The present work was submitted to the LuFG Theory of Hybrid Systems

MASTER OF SCIENCE THESIS

ACCELERATING SYMBOLIC SIMULATION TO ANALYZE THE EFFECT OF DELAYS IN TRAIN TIMETABLES

Niklas Kotowski

Examiners:
Prof. Dr. Erika Ábrahám
Prof. Dr. Thomas Noll
Additional Advisor:
Rebecca Haehn

Aachen, 30.09.2021

Abstract

Railway traffic is a substantial part of public transport and its safe execution and reliability is a significant factor in the timetable performance. To improve reliability and quality of service, symbolic simulation is deployed to analyze the propagation of train delays in railway systems. Inside of the timetable execution trains are depicted by stochastic cases representing train instances. The precise computation of train delay propagation leads to inevitable growth of the state space slowing down the simulation to an impractical state.

This thesis aims at accelerating the procedure and improving the overall algorithm scalability by applying different reduction techniques. The thesis begins with theoretical foundations and a detailed illustration of the simulation process. Then, the first reduction method in form of a set of rules is introduced and implemented in a reduction framework. As the abstraction of train instances comprises an extended potential of reducible instances not covered by these reduction rules, an efficient SAT encoding extending the reduction rules is defined and solved in form of a bounded model checking approach. Finally the implementations are evaluated on a set of examples varying in size and complexity. The experimental evaluation shows significant reductions of the number of stored instances and correspondingly the runtime of symbolic simulation. On large inputs the approach has a maximal runtime improvement of 99%.

This thesis took a huge step in the direction of efficient execution of the approach on large railway timetables allowing even the computation of input examples previously not terminating in a period of hours.

Erklärung

Hiermit versichere ich, dass ich die vorgelegte Arbeit selbstständig verfasst und noch nicht anderweitig zu Prüfungszwecken vorgelegt habe. Alle benutzten Quellen und Hilfsmittel sind angegeben, wörtliche und sinngemäße Zitate wurden als solche gekennzeichnet.

Niklas Kotowski
Aachen, den 30. September 2021

Acknowledgements

I would like to thank Prof. Dr. Erika Ábrahám for the opportunity to work on this research topic and the helpful thoughts and discussions. Furthermore, I want to thank Rebecca Haehn for the constant support and the many meetings. Additionally I want to thank my family and especially Jana for always supporting me.

Contents

Chapter 1

Introduction

In today's railway systems the importance of punctual train schedules is of substantial relevance. Public transportation and their performance have a significant effect on the global system they are part of e.g. punctual arrival at work. A complete rebuild and improvement of the infrastructure is exceeding realistic boundaries. Therefore, the focus is set on enhancing the current analysis tools in order to develop robust timetables. The idea of analyzing train delay propagation has shown to have promising potential to further increase efficiency and reliability of train timetables, as shown in [HÁN21] and [JNRSfTL06].

Railway timetables are mainly influenced by uncertain events that cannot be accurately predicted. Those events are frequently malfunctions of the underlying infrastructure, extreme weather conditions or external factors of diverse unplanned events. To examine the impact of disruptive factors on the remaining timetable, we decided to model those with primary delay distributions.

We simulate the railway timetable symbolically under special consideration of the primary delay values to analyze the impact of those on the complete schedule. The symbolic simulation models railway systems for a given time interval with the help of an exact representation in a macroscopic abstraction of the railway infrastructure. The merely reduced representation allows an exact simulation of the schedule while keeping track of all possible scenarios. As a short explanation, scenarios are abstract instances inside of the simulation to represent participating trains in certain probabilistic situations. In the further realization of timetables, the delayed trains can cause interferences with the remaining trains and induce additional delays. The symbolic simulation aims at identifying problematic trains initiating a non-negligible amount of additional delays while maintaining an exact representation of the possible train behavior. During the execution, the scenarios have to be split to model the exact conduct at conflicting infrastructure elements, shortly speaking stations or rails not allowing additional trains to enter due to capacity constraints. This together with the correct computation of train instances in varying stochastic cases increments the number of instances significantly. The large growth of the state space impairs the scalability and performance of the approach and prevents an exact analysis of realistic input data.

The goal of this thesis is to analyze possible acceleration techniques and reduction methods to slow down the explosive growth of the state space. Therefore, we explore reduction techniques to merge existing instances and improve the algorithm's running

time.

In Chapter 2, we introduce required definitions for the complete understanding of the symbolic simulation, the definition of train instances in form of scenarios and the exact procedure of simulating train timetables. This part follows a Section 3.0.1 defining the theoretical foundation of scenario reductions. This consists of formalized requirements for various reduction methods and corresponding illustrative examples. Moreover, in Section 3.2 the implementation part is described to full extent, presenting the differences between various reduction techniques and their theoretical reduction power. The chapter is then concluded by a correctness proof, verifying that the presented reduction framework is sound and complete. In Chapter 4 the defined reduction rules are modeled in a SAT encoding and applied in a bounded model checking approach to compute a more strategic reduction order. To evaluate the acceleration techniques and performance improvements, Chapter 5 presents an extensive experimental analysis of varying railway systems. Finally, in Chapter 6 the results are summarized and put into a realistic context, together with prospects of potential enhancements in an abstract about future work.

Chapter 2

Preliminaries

In the following chapter, the theoretical foundation for modeling and symbolically simulation railway systems is defined. Therefore, we begin by defining the railway system by representing the infrastructure with a directed graph and introduce a timetable to store all relevant information about participating trains. Afterwards, the symbolic train instances are defined as scenarios including definitions of probability theory to ensure the overall correctness of the approach. Finally in the last part of the chapter the symbolic simulation procedure is depicted. Note here that the illustrated symbolic simulation has been initially defined in [HÁN21].

2.1 Railway System

In today's railway systems, the demand for reliable train schedules is of significant importance. The objective is to reduce train delays and optimize the use of given resources. Optimizing timetables and minimizing delays can have a significant effect on the quality of service. In the following approach, the focus lies on analyzing the propagation of delays in a railway system.

To examine this, we analyze how trains that start their schedule with a delay affect other trains in the system. In the following, we denote this event as primary delay. To study this, we deploy a symbolic simulation. Typically, railway systems are represented microscopically, here we relax this level of detail and define a macroscopic representation. Consequently, various physical details of the railway system are omitted. We choose to reduce the precision, as a microscopic representation would slow down the simulation process to an impractical state. With the chosen representation, defined in Definition 2.1.1, small railway networks can be simulated efficiently. However, on large instances with an increased time horizon the current implementation is not able to efficiently simulate the system.

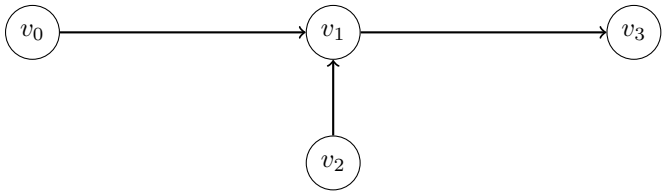
The idea of this thesis is to improve the scalability of the algorithm by investigating and implementing optimizations. The expected result is that with the help of a reduction even large networks can be efficiently simulated.

Definition 2.1.1 (Graph)**.** *The railway infrastructure is represented by a directed graph* $G = (V, E, c)$ *with* $c : V \cup E \to \mathbb{N}$*, which consists of a set of vertices* V *and edges* E*. The nodes* $v \in V$ *represent train stations and the edges* $e \in E$ *for* $E \subset \{(v, u) \in V \times V | v \neq u\}$ *depict the connections between those. Additionally,* c *assigns each infrastructure element* $x \in V \cup E$ *a capacity value* $c(x) \in \mathbb{N}$*.*

The capacity function restricts the simultaneous existence of train instances at an element x. For edges $e \in E$ the capacity models the number of parallel tracks available. While at stations $v \in V$ the value represents the number of existing halting points. Note here, that we simplify the representation of the railway structure by omitting additional physical parts, e.g. switches, et cetera. Despite this simplification, we assume that the abstraction is sufficiently accurate to fit the purpose of the simulation. Moreover, we introduce an example to illustrate the infrastructure graph.

Example 2.1.1.

$$G = (V, E, c),\ \forall x \in V \cup E.\ c(x) = 1 \textit{ with}$$
$$V = \{v_0, v_1, v_2, v_3\} \textit{ and } E = \{(v_0, v_1), (v_2, v_1), (v_1, v_3)\}$$



To define the schedules of participating trains we introduce timed paths.

Definition 2.1.2 (Timed path)**.**
Let $\pi = (v_1, a_1, d_1), ..., (v_n, a_n, d_n)$ be a timed path consisting of a set of triplets:

- *vertices $v_1, ..., v_n \in V$ such that $(v_i, v_{i+1}) \in E$ for $i = [1, ..., n-1]$.*
- *loop free $\forall i, j \in [1, ..., n]$ with $i \neq j$. $v_i \neq v_j$.*
- *an arrival $a_i \in \mathbb{R}$ and departure time $d_i \in \mathbb{R}$ for each vertex v_i.*

In addition to the definition of the railway infrastructure with G and a trains schedule with timed paths π_i, we define the complete timetable $\mathcal{T}$.

Definition 2.1.3 (Timetable)**.**
Timetable $\mathcal{T} := \{(type_1, \pi_1), ..., (type_n, \pi_n)\}$ for $T = [T_{min}, T_{max}]$:

- *$type_i$ for $i \in [1, ..., n]$ specifies the physical train type.*
- *a timed path π_i for $i \in [1, ..., n]$ specifies for each train i its timed schedule.*

The train's type is important for the actual simulation as it affects the trains stopping times and more importantly its priority. Examples for physical train types are e.g. ICE, IC, and RE.
The example is extended with a timetable:

Example 2.1.2. $\mathcal{T} = \{(ICE, \pi_1), (D, \pi_2), (RE, \pi_3)\}$ *for* $T = [0, 10]$

$$\pi_1 = (v_0, 0, 0), (v_1, 0, 2), (v_3, 3, 5$$
$$\pi_2 = (v_2, 4, 4), (v_1, 5, 6), (v_3, 7, 8)$$
$$\pi_3 = (v_0, 7, 7), (v_1, 8, 9), (v_3, 9, 9)$$

In the following, we will describe additional definitions and assumptions relevant for the symbolic simulation. During the simulation trains are represented by a theoretical abstraction. In order to define this, we present two principles of probability theory, following [FG13].

2.2 Scenarios

Definition 2.2.1 (Probability space)**.** *Let the triplet $(\Omega, \mathcal{F}, \mathcal{P})$ be a probability space. Then Ω is a non-empty set called sample space containing all possible outcomes of the corresponding random phenomenon. $\mathcal{F}$ is the set of events, such that events are arbitrary subsets of Ω. $\mathcal{P} : \mathcal{F} \to [0, 1]$ is the probability function assigning each event $\omega \in \mathcal{F}$ a real-valued probability.*

As in the simulation context the sample space is always finite, the set of events can be defined as the power set of Ω. In order to denote an event, we introduce the notion of a random variable.

Definition 2.2.2 (Random variable)**.** *Let p be a random variable in the probability space $(\Omega, \mathcal{F}, \mathcal{P})$, then Ω represents all values p can possibly take. In extension $\mathcal{F}$ is the power set of Ω. The function $\mathcal{P}$ defines the probability of p taking a value $\omega \in \mathcal{F}$. The following conditions have to hold for $\mathcal{P}$:*

$$\mathcal{P}(p = \omega) > 0 \tag{1}$$

$$\sum_{x \in \Omega} \mathcal{P}(p = x) = 1 \tag{2}$$

Given these fundamentals of probability theory, we can introduce the notion of primary delay.

Definition 2.2.3 (Primary delay)**.** *Let p_i for $i = [1, ..., n]$ be random variables in the probability space $(\Omega, \mathcal{F}, \mathcal{P})$, such that they represent the primary delay of a train i at its starting station. We define the sample space Ω of p_i as the finite set $\mathcal{D}(p_i) := \{n \in \mathbb{N} \mid \mathcal{P}(p_i = n) \neq 0\}$ further denoted as support set of p_i. Every variable p_i has a discrete probability distribution $\mathcal{P} : \mathcal{F} \to [0, 1]$, that expresses the probability of train i to become a given number of time units delayed, without the cause being another train.*

We refer to the set of all random variables as $P := \{p_i \mid i \in [1, ..., n]\}$, where n is the number of trains in the railway system. We assume that all of these variables are statistically independent since they represent primary delay. Here and subsequently, initial delay is considered as a synonym for primary delay. We extend the example introduced above by an initial delay set $\mathcal{D}$.

Example 2.2.1.

$$\begin{aligned} \mathcal{D}(p_i) &= \{0, 1, 2, 3\} \text{ for } i \in [1, 2, 3] \\ \mathcal{P}(p_i = 0) &= 0.5 \\ \mathcal{P}(p_i = 1) &= 0.2 \\ \mathcal{P}(p_i = 2) &= 0.2 \\ \mathcal{P}(p_i = 3) &= 0.1 \end{aligned}$$

Remark 2.2.1. *Moreover delays can be caused in case a scheduled train cannot enter an infrastructure element due to capacity constraints. This delay is further denoted as secondary delay. Additional primary delays induced during a train ride are ignored for now.*

Given the definition of primary delay, we require a formal structure to restrict the initial delay values of a train. Therefore, we introduce the definition of random inclusion.

Definition 2.2.4 (Random inclusion)**.** *A random inclusion c for $p_i \in P$ has the form $p_i \lhd D$ for some $D \subseteq \mathcal{D}(p_i), D \neq \emptyset$.*

With the help of random inclusions we can now restrict the initial delay of trains.

Example 2.2.2.

$c_1 : \{p_1 \lhd \{0\}\}$ - *train$_1$ starts its schedule without delay.*

$c_2 : \{p_2 \lhd \{0, 1\}\}$ - *train$_2$ starts without delay or with one time unit delayed.*

Given the theoretical foundation restricting initial delay sets, we introduce a formalization for to define stochastic cases for trains. Inside of the simulation, trains are represented by scenarios, which consist of random inclusions in which each train's random variable is restricted at most once. The definition is taken from [HÁN21].

Definition 2.2.5 (Scenario)**.** *A scenario S is a set that contains exactly one random inclusion for each random variable. Let $\mathcal{S}$ be the set of all scenarios. For $S \in \mathcal{S}$ and $(p_i \lhd D) \in S$ we define $S(p_i) = D$, and set $\mathcal{P}(S) = \prod_{c \in S} \mathcal{P}(c)$. We call a scenario S complete iff $|S(p_i)| = 1$ for each $p_i \in P$. We say that $S \in \mathcal{S}$ refines $S' \in \mathcal{S}$ (written $S \preceq S'$) iff $S(p_i) \subseteq S'(p_i)$ for all $p_i \in P$; we also say that S' contains S. We call S and S' compatible iff $S(p_i) \cap S'(p_i) \neq \emptyset$ for all $p_i \in P$. For two compatible scenarios S and S' we define $S \triangle S'$ as the scenario $\{p \lhd (S(p_i) \cap S'(p_i)) \mid p \in P\}$. Likewise two scenarios S and S' are incompatible, iff $\exists p_i \in P.\ S(p_i) \neq S'(p_i)$, denoted by $S|_i S'$.*

A non-complete scenario includes a number of complete scenarios as sub-scenarios. As the random inclusions for non-complete scenarios do not have to fix values, but only restrict them.

Remark 2.2.2. *In the following, a scenario refers to a non-complete scenario, while complete ones are explicitly denoted as such.*

Scenarios denote in which stochastic case a train exists at $x \in V \cup E$ during t. Inside of the simulation we formalize this as train instances.

Definition 2.2.6. *Train instance*
A train instance is a triplet (i, S, t_{ep}) at $x \in V \cup E$:

- *$i \in [1, ..., n]$ is the train id.*
- *S is a scenario, describing the stochastic case in which the train exists.*
- *$t_{ep} \in \mathbb{R}$ is the earliest possible departure time.*

Example 2.2.3.

$$(1, S_1, 2) \text{ at } v_2 \text{ with } S_1 : \{p_1 \lhd \{0\}, p_2 \lhd \{0\}, p_3 \lhd \{2\}\}$$

Train$_1$ is during $t = 2$ at vertex v_2, if train one and two are punctual, while train three is delayed with two time units.

$$(2, S_2, 4.5) \text{ at } v_3 \text{ with } S_2 : \{p_1 \lhd \{1\}, p_2 \lhd \{1\}, p_3 \lhd \{1\}\}$$

Train 2 is during $t = 4.5$ at vertex v_3, if train one, two and three start their train ride one time unit delayed.

Definition 2.2.7 (Set of scenarios)**.** *Let $\mathcal{S} := \{S_1, ..., S_m\}$ for $m \in \mathbb{N}$ be the set of all scenarios. Adding an index i to the notation $\mathcal{S}_i$ restricts the set to contain only scenarios for the train with id i.*

2.2.1 Invariant

To get correct results, we have to formalize a certain invariant that has to hold during symbolic simulation. Therefore, we ensure that in case a train is part of the current schedule, it has to exist in form of scenarios summing up to a probability of one. The invariant is defined as the computed sum of probabilities, such that for each currently participating train holds:

$$\sum_{S \in \mathcal{S}_i} \mathcal{P}(S) = 1 \pm \epsilon, \tag{2.1}$$

where ϵ is an infinitesimal number covering rounding errors occurring due to the usage of doubles in the implementation. We assume this invariant to hold throughout the simulation and additionally we implemented that Equation 2.1 has to be satisfied at each time step during the simulation to ensure that the later presented reduction techniques do not violate correctness of the approach.

2.3 Timetable Execution

This section briefly introduces a timetable execution and corresponding properties of further interest. The execution of a timetable gives insights regarding trains potentially causing other trains to be further delayed and infrastructure elements at which many trains have to halt due to capacity conflicts. These important results can then further be used to adapt the timetable accordingly, thus decreasing secondary delays and improving the efficiency of the railway system.

The timetable is executed according to the contained schedules in form of timed paths and starts with an initially defined set of primary delays. During the simulation, secondary delays can occur due to conflicts at infrastructure elements. Those postponements cannot be prevented and have to be stored correctly to capture all possible stochastic cases that are induced by them. Additionally, all trains drive as early as they can, which means that in case the element they want to move to has a free capacity slot and the earliest possible departure time (epdt) is smaller than the current time step, the train moves to its next vertex. In case more than one train wants to enter an element and the capacity restricts the operation, a priority order has to be introduced. In the implementation, simplified priority handling is defined. Moreover, in the execution of a timetable delays are shortened by rerouting and applying certain optimizations. In this simulation halting times are reduced and delayed trains are sped up, rerouting is not applied.

2.3.1 Symbolic simulation

The exact computation of delay propagation inside of a railway system is highly demanding. Therefore, continuous analysis of all complete scenarios is not practical. Nevertheless, a symbolic simulation considering non-complete scenarios and a refined subset of time steps is applicable. Due to the timetable and corresponding strict train

schedules, only a limited subset of time points is of relevance, as time steps without train movement do not produce new insights. This is explained by the introduced abstraction that we only model a simplified position of trains being at an infrastructure element neglecting where it is exactly. To cover up all relevant time points, the implementation keeps track of upcoming time steps at which a train movement is scheduled. The exact computation is explained below in more detail, the symbolic simulation is further depicted in [HÁN21].

In the initialization process, all trains enter the system with a previously defined set of primary delays. Thus a train starting its schedule is inserted into the simulation as a set of train instances. Consequently, it is possible to analyze what impact a delayed starting time has on its schedule and more importantly on the other participants.

Additionally, while executing the timetable, the movement of a train is mainly influenced by the occupations of the next stations in the system. Blocked infrastructure elements induce secondary delays and impact the whole timetable reliability. To correctly compute train instances that cannot directly enter the next vertex a split of instances is deployed. The split method divides scenarios into parts representing a probabilistic case in which the movement is possible and one in which the train has to halt at the current infrastructure element. Note that the symbolic simulation and especially the initial set of primary delays is a simplified abstraction. Nevertheless is the symbolic simulation an efficient and precise approach to gather realistic results on train delay propagation and eventually the identification of problematic trains.

Initialization

The symbolic simulation initializes a wide set of global variables required to correctly store information during the execution. In the following, those are shortly listed and explained as they are of significant importance for the understanding of the simulation and the later explained reduction methods.

The input given to the simulation is an infrastructure graph $G = (V, E, c)$, a timetable $\mathcal{T}$, a time period $[T_{min}, T_{max}]$, a measure for a safety distance $\delta \in \mathbb{R}$ given in time units and a set of random variables $P := \{p_i | \text{ for } i \in [1, ..., n]\}$. Then, the initialization process begins by connecting all vertices to a new source vertex v_s with $c(v_s) = \infty$ and $\forall v \in V.\ c((v_s, v)) = \infty$. This ensures that the simulation can model that a train is not able to start on schedule on its initial vertex in case it is already fully occupied. Moreover, for each vertex $v \in V$ an edge connecting it to a target vertex v_t with $c(v_t) = \infty$ and $\forall v \in V.\ c((v, v_t)) = \infty$ is included. Thus a train, which completed its schedule can move to a position without a capacity limit to not block any of the remaining train instances.

First, the procedure computes for each infrastructure element $x \in V \cup E$ the set of scenarios $req[x]$ that head to the current infrastructure element x as the next position in their timed path. Consequently, during the execution train instances occupy an infrastructure element by halting or driving on it and they block it by having left it and not having exceeded the safety distance δ. In addition analogous to the request set, for each element in the network sets of scenarios $occ[x]$, $block[x]$ occupying or blocking the element are defined. As a measure for intuitive handling of the split method, a data structure $cap[x]$ stores for each element the number of train instances blocking and occupying it together with the corresponding scenarios. This map is required to correctly schedule trains in conflict situations, where conflicts occur in

Algorithm 1 Simulation Algorithm

```
1: procedure SIMULATE(G, 𝒯, P)
2:     for each t ∈ times do
3:         for each x ∈ V ∪ E do
4:             req[x] ← COMPREQ(t, x);
5:             occ[x] ← COMPOCC(t, x);
6:             for each r ∈ req[x] do
7:                 UPDATE(t, x, r);
```

case a train cannot enter its next routing point. Finally, the set of time steps at which the simulation updates is declared as $times \subset \{t \mid t \in T\}$ and initially contains all initial time points for all participating trains $i \in [1, ..., n]$.

2.3.2 Simulate

The simulate function controls the complete procedure of the symbolic simulation. In Algorithm 1, the functionality is depicted and in the following we will explain the implementation stepwise.

The initialization has already been explained, thus we proceed with the main method. The SIMULATE$(G, \mathcal{T}, P)$ method is called with the infrastructure graph G, the timetable $\mathcal{T}$ and the set of variables P. The simulation begins with entering the first loop in line 2, taking out the smallest time step and the first vertex $v \in G$ in line 3. Note here, that the implementation processes all vertices before working on the edges, as this enables trains to directly drive through a station. Then, the algorithm computes the set of requests $req[x]$ and occupiers for x at the given time step t.

Next, in line 6 of Algorithm 1 we loop over the set of requesting instances $req[x]$ and call UPDATE(t, x, r) with the current time step t, the current infrastructure element x and the requesting instance r. Shortly speaking, the update method tries to move the requesting instance to the current infrastructure element x, however due to capacity constraints it can occur that the train instance cannot move to x. In such cases, we split the train instance, therefore we divide the contained scenario if possible in a stochastic case in which the train can move and in one case in which it has to halt at the previous element stored in $pre[x]$.

The SIMULATE function runs until T_{max} is reached. After the execution symbolic simulation has obtained all relevant information about the timetable with the initially defined probabilistic distribution for primary delays. This includes data about arrival times and induced secondary delays. Owing to the design of the simulation, the set of scenarios is growing during the execution. However, at no point inside the analysis, the set of scenarios is reduced or merged, thus leading to significant growth of the state space. This is one cause why the simulation is slowing down throughout the computation. Correspondingly, in larger railway systems and especially in simulations over an extended period, an efficient execution is no longer feasible. This restricts the current application to small inputs and an imprecise discretization of probabilistic distributions modeling the primary delay to not overload the computation. To tackle this issue and improve the scalability of the simulation, we introduce a reduction approach in Chapter 3.

Algorithm 2 Attempts to Update Trains Position

1: **procedure** UPDATE($t \in times, x \in V \cup E, r = (i, S, t_{ep}) \in [1, n] \times \mathcal{S} \times T$)
2: $\mathcal{S} \leftarrow \emptyset$;
3: **if** ($|occupy[x] \cup block[x] \cup request[x]| < c(x)$) **then** $\mathcal{S} \leftarrow \{S\}$;
4: $\mathcal{S} \leftarrow$ AVAILABLE(x, r);
5: **for each** $S \in \mathcal{S}$ **do**
6: $occ[pre(x)] \leftarrow occ[pre(x)] \setminus (i, S, t_{ep})$; ▷ remove instance from predecessor
7: $occ[x] \leftarrow occ[x] \cup (i, S, t + t')$; ▷ add instance to x
8: $block[pre(x)] \leftarrow block[pre(x)] \cup (i, S, t + \delta)$; ▷ update blocking instances
9: $times \leftarrow times \cup t + t'$; ▷ update times
10: $\mathcal{S}'' \leftarrow$ DIFFERENCE($S, \mathcal{S}$); ▷ compute remaining instances
11: **for each** $S \in \mathcal{S}''$ **do**
12: $occ[pre(x)] \leftarrow occ[pre(x)] \cup (i, S, t_{ep})$; ▷ instances that could not move

2.3.3 Occupation

The function COMPREQ(t, x) collects all predecessor elements for the given infrastructure element x. Then we check, if the epdt t_{ep} of their train instances is below the given time step t. All train instances fulfilling this are inserted into $req[x]$ to update the given set of requesting instances.
After updating the set of requests $req[x]$, we have to compute occupying and blocking train instances for the current infrastructure element x . Therefore, we execute the COMPOCC function to correctly collect the blocking and occupying train instances.

The method COMPOCC(t, x) computes the current set of blocking train instances by including all new blocking trains and removing all previously blocking instances that have exceeded the safety distance δ. Then, it collects all train instances which are currently moving through or halting at x and stores all computed instances in $occ[x]$.

2.3.4 Update

In the following we further explain UPDATE(t, x, r), which is illustrated in Algorithm 2. The update method begins with the definition of an empty scenario set $\mathcal{S}$, which is required to store the scenarios that are scheduled to move to x. Thus, we first check if the requested instance can move to the current infrastructure element x by comparing the number of currently blocking, occupying and requesting instances with the capacity of x (line 3). In case the train instance can realize the requested movement no further splitting is needed and $\mathcal{S}$ is updated to solely contain S. However, if the capacity is not sufficient for the set of instances, we call AVAILABLE(x, r). As a short note, the function does compute a set of scenarios $\mathcal{S}$ in which the train of the given request r can be moved to x. The exact procedure is explained below and depicted in Algorithm 3.

After gathering the set of scenarios $\mathcal{S}$ defining the stochastic cases in which the requesting instance can be moved to x, we loop over each scenario that is contained in the computed set $S \in \mathcal{S}$(line 5). Next, we remove the occupying instances of the predecessor element $occ[pre(x)]$ by removing the requested instance r. Then, the occupying instances of the current element x are updated to include the scheduled instance r with the constructed scenario S. The epdt of the instance is set to $t_{ep} =$

Algorithm 3 Computes Scenarios in which Request can be scheduled.

```
1: procedure AVAILABLE(x ∈ V ∪ E, r ⊆ {(i, S, t)|i ∈ [1,n]})
2:     S' ← ∅; cap_t ← ∅;
3:     for each (S', u) ∈ cap[x] do
4:         if (u ≥ c(x) ∨ S incompatible with S') then
5:             cap_t ← cap_t ∪ (S', u);
6:             CONTINUE;
7:         S' ← S' ∪ {S'△S};
8:         if (S' ⪯ S) then                ▷ decide if the scenario is included in S
9:             cap_t ← cap_t ∪ (S', u + 1)          ▷ capacity usage is incremented
10:        else                                  ▷ or if a separation is required
11:            cap_t ← cap_t ∪ (S△S', u + 1);                          ▷ update cap
12:            cap_t ← cap_t ∪_{S_d ∈ DIFFERENCE(S, S')} (S_d, u);
13:    cap[x] ← cap_t;
14:    return S';
```

$t+t'$, where t' computes the time the instance has to stay at the current element. Note here, that we omit additional details of the exact computation of t_{ep} as they are not of significant importance for the thesis, interested readers are referred to [HÁN21]. In line 8 the blocking states for the predecessor are updated to contain the instances that are moved to x. Finally, the set of time steps is updated to contain the newly computed t_{ep}. In case the request could not be scheduled conflict-free (line 3), the set of scenarios in which the request can be scheduled does not contain all scenarios of the requesting instance. Therefore, we have to compute the difference of S and $\mathcal{S}$ to determine the set of scenarios that have to halt at the predecessor of x. To compute the stochastic cases in which the train instance cannot move to x we call DIFFERENCE(S, $\mathcal{S}$) constructing the set of scenarios, which are contained in S but not in any of the instances of $\mathcal{S}$. The resulting scenarios are then added to the occupying states of the predecessor (line 12). Those instances will attempt to move to x in the next iteration phase. To this extent, the next call is invoked for the next request.

For completeness we extend the functionality by explaining the procedure of the AVAILABLE(t, x) method. The availability function is required to evaluate if and in which stochastic scenarios a given request r can be successfully executed. The whole procedure is depicted in Algorithm 3.

2.3.5 Available

The AVAILABLE(x, r) function returns a set of scenarios in which the requesting train instance can be moved to the current element x. We begin by defining an empty set of scenarios $\mathcal{S}'$ to store all scenarios in which the request r can be conflict-free scheduled and a temporary set . To determine this set, in line 3 we loop through the set $cap[x]$ which contains all scenarios currently occupying or blocking x together with the number of elements (S', u) with $u \in \mathbb{N}$. Next in line 4 we check if the number of instances u is greater or equal to the capacity of x or if the corresponding scenario S' is incompatible with S. Then if the condition is satisfied the scenario is not suited to be scheduled in S', thus we include (S', u) in cap_t and continue with the next element in $cap[x]$. In the other case, that the capacity is sufficient and the scenario S' is compatible with S, $S \triangle S'$ is stored in $\mathcal{S}'$. Moreover, in the

lines 8-12 a case distinction is implemented to update cap_t correctly depending on the refinement relation between S and S'. Afterwards, $cap[x]$ is modified to equal cap_t, which stores the correctly updated capacity usage for each included scenario. The resulting scenario set $\mathcal{S}$ is returned and further used in UPDATE(t, x, r).

Chapter 3

Iterative Reduction

The symbolic simulation presented in the previous chapter propagates in each relevant time step the position of train instances in the railway system. During this procedure, conflicts occur hindering train instances to move to the next infrastructure element. To correctly handle this, the stochastic cases depicted in the scenarios are split to define states in which the train has to halt and those in which the instance can move to the next scheduled infrastructure element. Additionally, every time a train is inserted into the system, the simulation includes multiple train instances to cover the initial delay set. In the ongoing execution, those newly included or split instances induce additional conflicts, which increase the number of instances even further. This prevents an evaluation of larger railway systems with realistic initial delay distributions, as the number of train instances is growing at fast pace. As the growth of the state space is not tackled, the simulation slows down during its execution and is impractical to be applied on a larger scale.

As the current simulation does store scenarios that can be compressed without losing information, we introduce the theory for various reduction methods. In the following, we will set up notation and terminology to develop the theory behind the reduction approach. First, tuples of scenarios are separated into reducible and non-reducible ones. Second, the reduction rules are formalized and further illustrated in examples.

As a general notation for the following declarations, we denote S, S' as scenarios and $\mathcal{S}, \mathcal{S}'$ as sets of scenarios. Besides, we introduce a notion to define the set of all random variables p_i for a given scenario S that are part of nontrivial inclusions. Note here, that trivial random inclusion are those not restricting the value space, formalized as $S(p_i) \subsetneq \mathcal{D}(p_i)$. The set of variables is formalized as:

$$P := \{p_i \mid p_i \in \{p_1, ..., p_n\} \text{ with } p_i \lhd S(p_i) \wedge S(p_i) \subsetneq \mathcal{D}(p_i)\}. \tag{3.1}$$

Additionally, we define that all random variables p_i with $i \in [1, ..., n]$ used for upcoming examples have equal support sets of size four: $\mathcal{D}(p_i) = \{0, 1, 2, 3\}$

3.0.1 Requirements

Reductions cannot be applied to all scenario tuples, certain requirements have to be met to ensure the correct merging procedure not altering any information contained in the scenarios. Therefore, we begin by defining scenario pairs not suitable for a reduction. Important to note here is that a scenario pair cannot be reduced, if the following conditions are not satisfied ensuring that only scenarios can be reduced

which are treated equally by the simulation. A scenario is always contained in a train instance with additional parameters regarding the train id $i \in [1, ..., m]$ and the earliest possible departure time $t_{ep} \in T$. The train instances (i, S, t_{ep}) and (j, S', t'_{ep}) have to represent the same train and the epdt has to be equal or below the current time step t as only if those requirements are fulfilled we can merge the instances without modifying the train schedule defined in $\mathcal{T}$.

$$(i == j) \wedge ((t_{ep} \leq t \wedge t'_{ep} \leq t) \vee (t == t')) \tag{3.2}$$

In the following, we assume that the underlying train instances for given scenario tuples fulfill these requirements. The below-stated requirements try to measure the similarity of the given scenarios, simply speaking the first reduction approach is only able to merge scenarios which are nearly identical. Therefore, we initially have to sort out pairs, in which a more complex procedure is required. This does not mean that those scenarios cannot be reduced, rather the reduction rule is not capable of doing it. However, in the process of this chapter, the solving capacity of the techniques is increased to cover the complete reduction potential. This structure is mainly owed to the chosen research direction and it allowed us to step-wise analyze the impact of an increased complexity on the running time. We begin by defining three conditions that decide, if a scenario tuple is suitable to be reduced.

Scenario size First, the scenarios have to be of equal size in order to be reducible. The cardinality of variables is sufficient to measure this, as we ensured in the definition of P (Equation 3.1), that the set contains solely variables which are part of non-trivial inclusions.

$$|P| = |P'| \tag{3.3}$$

Example 3.0.1.

$$S = \{p_1 \triangleleft \{0\}, p_2 \triangleleft \{0\}\}$$
$$S' = \{p_2 \triangleleft \{1\}\}$$

The first scenario S restricts the initial delay set for p_1 and p_2. However, S' restricts it only for p_2, thus they cannot be reduced.

$$S = \{p_1 \triangleleft \{0\}, p_2 \triangleleft \{0\}\}$$
$$S' = \{p_2 \triangleleft \{1\}, p_3 \triangleleft \{2\}\}$$

In this case the scenarios would satisfy the first condition.

Variables The second condition states that the set of restricted variables has to be equal. Note here, that this condition implies the first one, accordingly, it is theoretically sufficient. Nevertheless, as Equation 3.3 can be verified more efficiently, we state it as a separate requirement. Inside of the implementation, we also make use of the above-stated condition to reject non-suited pairs faster.

$$P = P'$$

Example 3.0.2. *We use the example from before and evaluate if it satisfies the second condition.*

$$S = \{p_1 \triangleleft \{0\}, p_2 \triangleleft \{0\}\}$$
$$S' = \{p_2 \triangleleft \{1\}, p_3 \triangleleft \{2\}\}$$

The first scenario restricts the initial delay set of p_1 and p_2, while S_2 restricts the space for p_2 and p_3. Hence, the scenarios are too dissimilar to be merged trivially.

$$S = \{p_1 \triangleleft \{0\}, p_2 \triangleleft \{0\}\}$$
$$S' = \{p_1 \triangleleft \{1\}, p_2 \triangleleft \{2\}\}$$

The modified example does fulfill the first and second requirement.

Random inclusions The last premise for the reduction rule states, that the scenarios only differ in exactly one random inclusion. Therefore, we introduce a set $C_{\neq}$ covering the set of distinct random inclusions for a scenario tuple (S, S'). In the below equation we denote by c_i, c_i' $i \in [1, ..., n]$ the random inclusion for $p_i \in [1, ..., n]$ and we require $\forall p_i \in P.\ p_i = p_i'$.

$$C_{\neq} := \{(c_i, c_i') | c_i \neq c_i' \text{ for } c_i \in S \text{ and } c_i' \in S'\}$$

$$|C_{\neq}| = 1$$

The scenarios have to differ in exactly on random inclusion in order to be reducible.

Example 3.0.3. *Next, the modified example is checked to fulfill the final condition.*

$$S = \{p_1 \triangleleft \{0\}, p_2 \triangleleft \{0\}\}$$
$$S' = \{p_1 \triangleleft \{1\}, p_2 \triangleleft \{2\}\}$$

However, here S, S' differ in more than one random inclusion, thus S, S' cannot be reduced.

$$S = \{p_1 \triangleleft \{0\}, p_2 \triangleleft \{0\}\}$$
$$S' = \{p_1 \triangleleft \{1\}, p_2 \triangleleft \{0\}\}$$

The example is modified another time and is now suitable to be reduced.

3.1 Reduction Rules

The main idea of the reduction approach is to collect all scenarios matching Equation 3.2 at certain time steps in the simulation and try to reduce those. Given a set of scenarios $\mathcal{S}$ the reduction procedure returns a potentially modified and optimally reduced set of scenarios $\mathcal{S}'$ without loosing any information.

In the remainder of this section we assume all scenario pairs to be of equal size $|P| = |P'|$, that constraint the same variables $P = P'$, differ in exactly one inclusion $|C_{\neq}| = 1$ and are non-empty.

We begin by defining the reduction rule for suitable scenario tuples fulfilling all required conditions.

The condition for two scenarios S, S' to be reducible written as $S \simeq_R S'$ can be formalized as:

$$\begin{aligned} &S \simeq_R S', \text{ iff} \\ &\exists p \in P.\ S(p) \cap S'(p) = \emptyset \wedge \\ &\forall p' \in P \setminus \{p\}.\ S(p') = S'(p'). \end{aligned} \tag{3.4}$$

As all trivial inclusions are identical we omit them. The reduction result is further denoted by $\hat{S}$ and formalized by:

$$\tilde{S} := (S \setminus \{p \lhd S(p)\}) \triangle \{p \lhd S(p) \cup S'(p)\} \tag{3.5}$$

Let $S, S' \in \mathcal{S}$ and $S \simeq_R S'$, then $\mathcal{S}' := (\mathcal{S} \setminus \{S, S'\}) \cup \{\hat{S}\}$, where $\hat{S}$ is the merged scenario. The intuitive idea of the reduction rule is, that the given scenarios are nearly equal and only differ for one random inclusion and can thus be compressed into one scenario without losing any precision. The intersection between the differing inclusion pair has to be empty as in the other case some complete scenario would be contained twice. As this would be a violation of the earlier defined invariant (in Equation 2.1), such a case cannot occur.

$$S(p) \cap S'(p) = \emptyset \tag{3.6}$$

The reduction rule then computes the union between $S(p)$ and $S'(p)$ to define the resulting random inclusion covering the information contained in both scenarios.

$$\tilde{c} : p \lhd S(p) \cup S'(p)$$

Example 3.1.1. *Below we depict two examples, illustrating the reduction procedure on different scenario tuples. In the second example, the reduction leads to the construction of a trivial inclusion for variable p_1, which is removed in the resulting scenario.*

$$S_1 : \{p_1 \lhd \{0\}, p_2 \lhd \{0,1\}\},\ S_2 : \{p_1 \lhd \{1\}, p_2 \lhd \{0,1\}\} \Rightarrow S' : \{p_1 \lhd \{0,1\}, p_2 \lhd \{0,1\}\}$$
$$S_1 : \{p_1 \lhd \{0,1\}, p_2 \lhd \{1\}\},\ S_2 : \{p_1 \lhd \{2,3\}, p_2 \lhd \{1\}\} \Rightarrow S' : \{p_2 \lhd \{1\}\}$$

3.1.1 Relaxed reduction

As in the current approach possible reduction opportunities are missed out, we extend the rules by a more involved reduction technique. We further denote this second rule as *relaxed reduction*, as it relaxes the strict requirements depicted in Definition 3.4. First, we define an example, that is not reducible at the current state.

Example 3.1.2.

$$\begin{aligned} S_1 &: \{p_1 \lhd \{0,1\}, p_2 \lhd \{0\}\} \\ S_2 &: \{p_1 \lhd \{0\}, p_2 \lhd \{1\}\} \\ S_3 &: \{p_1 \lhd \{1\}, p_2 \lhd \{2\}\} \end{aligned}$$

S_1 contains a scenario suitable to be reduced with S_2, however, as we only consider scenarios with exactly one differing inclusion the tuple is not considered.

In order to consider these additional scenario tuples the approach is extend by the rule presented below. For the given example an application of the relaxed reduction enables another basic reduction and thus a strict instance decrease, the example will be completed after formalizing the method. For the relaxed reduction we have to lift the stated conditions. The number of differing random inclusions is fixed to exactly two with the condition that the set of one inclusion is contained in the other.

$$|C_{\neq}| = 2$$

Additionally, the number of variables $|P|$ does not have to be identical anymore. However, it has to differ at most by one as we consider only one trivial inclusion in the reduction process. The contained random variables are also allowed to differ in exact one position as the random variable of a trivial inclusion is not contained in P.

$$|P'| - 1 \leq |P| \leq |P'| + 1 \tag{3.7}$$

The exact requirements for the relaxed reduction are formalized as follows:

$$\begin{aligned} &S \simeq_{RR} S', \text{ iff} \\ &\exists p \in P.\ S(p) \neq S'(p) \wedge S(p) \cap S'(p) = \emptyset \wedge \\ &\exists p' \in P.\ p \neq p' \wedge (S(p') \subsetneq S'(p') \vee S'(p') \subsetneq S(p')) \wedge \\ &\forall \hat{p} \in P \setminus \{p, p'\}.\ S(\hat{p}) = S'(\hat{p}) \wedge \\ &\forall p \notin P.\ S(p) = \mathcal{D}(p). \end{aligned} \tag{3.8}$$

This includes the condition that exactly two constraints are unequal, while in one case the included values are completely distinct and in the other case, one value set has to be a strict subset of the other. Important to note here is that the relaxed reduction does not remove a scenario, but rather modifies the given ones and thus eventually enables another application of the basic reduction rule. Consequently, the approach only works with the subsequent application of the reduction rule (Equation 3.5) by possibly extending the set of reducible scenario pairs.

In Equation 3.8 we defined the requirements for the relaxed reduction, further we define the scenarios resulting from the application of the method. Let (S, S') be a tuple of scenarios satisfying $S \simeq_{RR} S'$, and let $p \in P$ with $S(p) \neq S'(p) \wedge S(p) \cap S'(p) = \emptyset$. Then, we assume w.l.o.g. that $p' \in P$ with $p \neq p' \wedge (S(p') \subsetneq S'(p'))$. The resulting scenarios are denoted by $\tilde{S}$ and $\tilde{S}'$ and formalized as:

$$\tilde{S} = (S \setminus \{p \triangleleft S(p)\}) \triangle \{p \triangleleft S(p) \cup S'(p)\},\ \tilde{S}' = S' \triangle \{p' \triangleleft S'(p') \setminus S(p')\} \tag{3.9}$$

The relaxed reduction rule removes the contained scenario required for the reduction out of S' and applies the reduction step to that scenario and S. Note that the application of the reduction rule is identical to Equation 3.5.

Example 3.1.3.

$$\begin{aligned} S_1 &: \{p_1 \triangleleft \{0, 1\}, p_2 \triangleleft \{0\}\} \\ S_2 &: \{p_1 \triangleleft \{0\}, p_2 \triangleleft \{1\}\} \\ S_3 &: \{p_1 \triangleleft \{1\}, p_2 \triangleleft \{2\}\} \end{aligned}$$

At this point the basic reduction rule is not applicable, however S_1 and S_2 satisfy Equation 3.8. Thus, we apply Equation 3.9.

$$\begin{aligned}\tilde{S_1} &: \{p_1 \lhd \{1\}, p_2 \lhd \{0\}\}\\ \tilde{S_2} &: \{p_1 \lhd \{0\}, p_2 \lhd \{0,1\}\}\\ S_3 &: \{p_1 \lhd \{1\}, p_2 \lhd \{2\}\}\end{aligned}$$

$\tilde{S_1}$ and $\tilde{S_2}$ are the result of the relaxed reduction and in combination with S_3 another basic reduction step is applicable.

$$\begin{aligned}\tilde{S_1} &: \{p_1 \lhd \{1\}, p_2 \lhd \{0,2\}\}\\ \tilde{S_2} &: \{p_1 \lhd \{0\}, p_2 \lhd \{0,1\}\}\end{aligned}$$

This is the final reduction result, as no rule is applicable for the given pair. Notice here, that the number of scenarios could be reduced by exactly one, which would not have been possible without the relaxed reduction.

In a practical application of the algorithm, this extended reduction rule showed to have a significant effect on the number of applied reductions, while improving the overall performance. As the relaxed reduction is not directly reducing the set of scenarios rather shifting the contained information, it is only applied in case the basic reduction is not applicable, further details on this are presented in Section 3.2.

3.1.2 Split

A non-complete scenario contains multiple complete scenarios. Thus, in various scenarios are sub-scenarios stored that are not directly accessible with the presented reduction methods. The basic reduction method can only be applied to a scenario tuple if the strict requirements are satisfied. However, the relaxed reduction extends the space of applicable pairs by allowing to make use of a sub-scenario contained in one of the scenarios. In the following subsection, a split approach is defined allowing to exploit all information stored in a set of scenarios. The intuitive idea of the split is that non-complete scenarios can be divided recursively into their complete sub-scenarios.

The requirement for a scenario to be split is that the scenario is non-complete containing at least two complete scenarios. Thus a scenario is suitable to be split, in case one of the random inclusion includes at least two elements. Note here, that this is also satisfied for all scenarios, which do not restrict all variables. The split technique is bound directly to a random variable p, which is not fixed in S. This can be formalized as:

$$S \text{ can be split, iff } \; p \in P.\ |S(p)| > 1$$

We assume w.l.o.g. that $|S(p)| > 1$, then the result of the split operation can be described by:

$$\begin{gathered}S, S' \text{ with}\\ \forall p' \in P \setminus \{p\}.\ S''(p') = S'(p') = S(p') \ \wedge S'(p) \subset S(p) \ \wedge \\ S''(p) \subset S(p) \ \wedge S'(p) \cup (S''(p)) = S(p) \wedge S'(p) \cap S''(p) = \emptyset.\end{gathered} \tag{3.10}$$

Note here, that the split operation alone does not reduce any instance, it even increments the number of scenarios. However, in combination with the reduction rule, the split operation does allow the approach to compute the optimal reduction result.

The optimal result is here defined as the minimal number of instances and likewise the maximal amount of applied reductions. This does not necessarily mean that computing the optimal result corresponds to having the smallest running time. The idea of this thesis is to find the level of reduction, which leads to the maximally reduced running time. The trade-off between required reduction time and saved time in the simulation is further analyzed and adjusted to correctly fit the research interest. This question is further analyzed in Chapter 4 and depicted in the experimental evaluation in Chapter 5.

Additionally, it is important to mention that the result of the relaxed reduction can be reconstructed by split and reduction rules. The relaxed reduction is a compressed application of split and the direct reduction of one of the resulting split instances. Nevertheless the relaxed reduction is defined as a single rule as it has shown to improve the performance of the symbolic simulation significantly as opposed to the split rule. Moreover, the split operation requires a more complex and strategic application to lead to a reduced result. Therefore, the split operation is only invoked in the bounded model checking approach in Chapter 4. A satisfiability solver evaluates if a split leads to a possible reduction of the input set and thus applies it only if it is beneficial.

To depict the limits of the reduction and relaxed reduction, we add an example that can only be reduced with the help of multiple splits. For this example we have to extend the support to $\mathcal{D}(p_i) := \{0, 1, 2, 3, 4\}$ for $i \in [1, 2, 3]$.

Example 3.1.4.

$$
\begin{aligned}
S_1 &: \{p_1 \lhd \{3\}, p_2 \lhd \{2, 3\}, p_3 \lhd \{2, 3\}\} \\
S_2 &: \{p_1 \lhd \{0\}, p_2 \lhd \{0, 2, 3\}, p_3 \lhd \{0, 2, 3\}\} \\
S_3 &: \{p_1 \lhd \{1\}, p_2 \lhd \{0\}, p_3 \lhd \{2, 3\}\} \\
S_4 &: \{p_1 \lhd \{2\}, p_2 \lhd \{0\}, p_3 \lhd \{0\}\} \\
S_5 &: \{p_1 \lhd \{4\}, p_2 \lhd \{2, 3\}, p_3 \lhd \{0\}\}
\end{aligned}
$$

At this point, neither the reduction rule nor the relaxed reduction is applicable. However, we can split S_2 in S_2' and S_6.

$$
\begin{aligned}
S_1 &: \{p_1 \lhd \{3\}, p_2 \lhd \{2, 3\}, p_3 \lhd \{2, 3\}\} \\
S_2' &: \{p_1 \lhd \{0\}, p_2 \lhd \{0, 2, 3\}, p_3 \lhd \{2, 3\}\} \\
S_3 &: \{p_1 \lhd \{1\}, p_2 \lhd \{0\}, p_3 \lhd \{2, 3\}\} \\
S_4 &: \{p_1 \lhd \{2\}, p_2 \lhd \{0\}, p_3 \lhd \{0\}\} \\
S_5 &: \{p_1 \lhd \{4\}, p_2 \lhd \{2, 3\}, p_3 \lhd \{0\}\} \\
S_6 &: \{p_1 \lhd \{0\}, p_2 \lhd \{0, 2, 3\}, p_3 \lhd \{0\}\}
\end{aligned}
$$

In the next step, we split S_2' another time in S_2'' and S_7.

$$
\begin{aligned}
S_1 &: \{p_1 \lhd \{3\}, p_2 \lhd \{2,3\}, p_3 \lhd \{2,3\}\} \\
S_2'' &: \{p_1 \lhd \{0\}, p_2 \lhd \{0\}, p_3 \lhd \{2,3\}\} \\
S_3 &: \{p_1 \lhd \{1\}, p_2 \lhd \{0\}, p_3 \lhd \{2,3\}\} \\
S_4 &: \{p_1 \lhd \{2\}, p_2 \lhd \{0\}, p_3 \lhd \{0\}\} \\
S_5 &: \{p_1 \lhd \{4\}, p_2 \lhd \{2,3\}, p_3 \lhd \{0\}\} \\
S_6 &: \{p_1 \lhd \{0\}, p_2 \lhd \{0,2,3\}, p_3 \lhd \{0\}\} \\
S_7 &: \{p_1 \lhd \{0\}, p_2 \lhd \{2,3\}, p_3 \lhd \{2,3\}\}
\end{aligned}
$$

After extending the set of scenarios by two splits, two reductions are applied. We reduce S_1 with S_7 and S_2'' with S_3.

$$
\begin{aligned}
S_1 &: \{p_1 \lhd \{0,3\}, p_2 \lhd \{2,3\}, p_3 \lhd \{2,3\}\} \\
S_2'' &: \{p_1 \lhd \{0,1\}, p_2 \lhd \{0\}, p_3 \lhd \{2,3\}\} \\
S_4 &: \{p_1 \lhd \{2\}, p_2 \lhd \{0\}, p_3 \lhd \{0\}\} \\
S_5 &: \{p_1 \lhd \{4\}, p_2 \lhd \{2,3\}, p_3 \lhd \{0\}\} \\
S_6 &: \{p_1 \lhd \{0\}, p_2 \lhd \{0,2,3\}, p_3 \lhd \{0\}\}
\end{aligned}
$$

Then we proceed by splitting S_6 in S_6' and S_8.

$$
\begin{aligned}
S_1 &: \{p_1 \lhd \{0,3\}, p_2 \lhd \{2,3\}, p_3 \lhd \{2,3\}\} \\
S_2'' &: \{p_1 \lhd \{0,1\}, p_2 \lhd \{0\}, p_3 \lhd \{2,3\}\} \\
S_4 &: \{p_1 \lhd \{2\}, p_2 \lhd \{0\}, p_3 \lhd \{0\}\} \\
S_5 &: \{p_1 \lhd \{4\}, p_2 \lhd \{2,3\}, p_3 \lhd \{0\}\} \\
S_6' &: \{p_1 \lhd \{0\}, p_2 \lhd \{2,3\}, p_3 \lhd \{0\}\} \\
S_8 &: \{p_1 \lhd \{0\}, p_2 \lhd \{0\}, p_3 \lhd \{0\}\}
\end{aligned}
$$

Finally, we reduce S_4 with S_8 and S_5 with S_6'. Then in the overall process the set of scenarios has been reduced to four instances.

$$
\begin{aligned}
S_1 &: \{p_1 \lhd \{0,3\}, p_2 \lhd \{2,3\}, p_3 \lhd \{2,3\}\} \\
S_2'' &: \{p_1 \lhd \{0,1\}, p_2 \lhd \{0\}, p_3 \lhd \{2,3\}\} \\
S_4' &: \{p_1 \lhd \{0,2\}, p_2 \lhd \{0\}, p_3 \lhd \{0\}\} \\
S_5' &: \{p_1 \lhd \{0,4\}, p_2 \lhd \{2,3\}, p_3 \lhd \{0\}\}
\end{aligned}
$$

Without introducing the split operation, exactly those combinations cannot be reduced. The efficiency and direct improvements are further analyzed in Chapter 5.

3.2 Implementation

In the previous chapter, theoretical reduction rules have been defined including a basic and a relaxed reduction rule. In this chapter, the implementation and corresponding inclusion into the symbolic simulation are outlined. The reduction is split into two different modes that can be evaluated separately.

First, a reduction approach in form of a priority queue with two reduction rules is specified. Second, the reduction problem is solved with Bounded Model Checking

(BMC) to use the complete reduction potential. The intuitive idea of the acceleration process is to apply the reduction at a heuristically adjusted set of time steps, slowing down the growth of the state space.

Requirements

As defined in the preliminaries chapter, the symbolic simulation abstracts trains in form of scenarios. Those scenarios are stored in triplets (i, S, t_{ep}), with $i \in [1, ..., n]$, a scenario S and the epdt $t_{ep} \in T$ and are in each time step bound to an infrastructure element $x \in V \cup E$.

The reduction approach is not allowed to interfere with or influence the outcome of the symbolic simulation. Therefore, the implementation has to exactly define sets of instances suitable to be reduced.

The first requirement is, that the scenarios have to be at the same infrastructure element x. In addition, t_{ep} has to be identical or smaller than the current time step t for all scenarios that are reduced. Identical earliest possible departure times are trivially allowed, time values smaller than the current time step are also suitable as they are both allowed to leave the element. Then, of course, they have to represent the same participating train i. In case these properties are fulfilled, the corresponding scenarios can be given to the reduction method.

Application

The reduction method is invoked at two points inside the execution, first it is executed at every r^{th} time step in SIMULATE$(\mathcal{S}, G, \mathcal{T})$. Therefore, we define a subset of time steps at which the method is executed as $times_r \subseteq times$. To match and satisfy the stated requirements, the algorithm loops over all infrastructure elements and filters suitable subsets of scenarios, which are then given to the currently deployed reduction technique, the exact procedure is depicted in REDUCEELEMENT(t, x).

Algorithm 4 Reduction Application

```
1: procedure SIMULATE(S, G, T)
2:     for each t ∈ times do
3:         for each x ∈ V ∪ E do
4:             if (t ∈ times_r) then REDUCEELEMENT(t, x); ▷ Reduce instances at x
5:             req[x] ← COMPREQ(t, x);
6:             occ[x] ← COMPOCC(t, x);
7:             for each r ∈ req[x] do
8:                 UPDATE(t, x, r);
```

The REDUCTION$(\mathcal{S}, H)$ method is further explained in the next section. However, it is important to clarify, that even if the reduction techniques are different the interface with the remaining implementation is identical. In addition the reduction is applied after each execution of AVAILABLE(x, r), as especially in an analysis of the effectiveness of reductions, this additional application was found to be of significant importance. The REDUCTION$(\mathcal{S})$ call is inserted in between line 4 and 5 of UPDATE(t, x, r).

Algorithm 5 Filter and Apply Reduction

```
1: procedure ReduceElement(t, x)
2:     S ← ∅;
3:     for each i ∈ [1, ..., n] do
4:         for (i, S, t') ∈ occ[x] with (t' ≤ t) do        ▷ Loop over all trains
5:             occ[x] ← occ[x] \ (i, S, t');               ▷ Collect suitable train instances
6:             S ← S ∪ {S};
7:         S ← Reduction(S);                               ▷ Reduce scenario set
8:         for each S ∈ S do
9:             occ[x] ← occ[x] ∪ (i, S, t);                ▷ Update occupiers
```

Reduction approach

In the first implementation of a reduction approach, the reduction possibilities are implemented as rules. The algorithm works in two parts, initially, scenario tuples are collected that are either reducible or relaxed reducible. The corresponding information is stored in a custom data structure q_{entry} containing a priority value and a tuple of scenarios (S, S'). Then, for Algorithm 5 to be loop-free, it has to keep track of

Data structure: Queue entry

1: $priority \in \mathbb{Q}$, heuristic value to decide about the reduction order
2: $pair$, stores reducible scenario tuples with $(S \simeq_R S' \vee S \simeq_{RR} S')$

scenario tuples used in a relaxed reduction. This has to be done, as the result of the relaxed reduction can be reapplied in a consecutive application of the rule directly undoing the last change. In order to prevent this, we store the participating scenarios in H and reset H as soon as a basic reduction is applied as the set of scenarios has possibly changed. We decided to not track the changed scenario tuples and thus only reset those entries of H, but rather reset the complete set as we evaluated the additional effort to be not efficient. This ensures that the execution is loop-free as we do not consider scenario tuples included in H in Line 2 of Algorithm 9. Therefore, we specify two global variables. The set H storing scenario tuples recently used in a relaxed reduction and the set $\mathcal{S}$, which is the current set of given scenarios further defined in Algorithm 6.

3.2.1 Reduction algorithm

Initialization

We begin by defining the initialization of the reduction procedure. The initialization is required to compute heuristic values to determine priority values for scenario tuples. The deployed heuristic favors small values and order scenario tuples dependent of their size and the variable over which is reduced. Hereby we prioritize large scenarios and small variable indices, where the scenario size is weighted higher. For the initialize function we introduce the sets $vars$ and $sizes$.

The set $vars := \{p_i \mid \text{for } S \in \mathcal{S}. \text{ with } S(p_i) \neq \mathcal{D}(p_i)\}$ stores all variables that are part of non-trivial inclusions for all scenarios in the given set $\mathcal{S}$. Additionally, $sizes := \{|S| \mid \text{for } S \in \mathcal{S}.\}$ contains the sizes of given scenarios. We define the two

Algorithm 6 Variables

1: $H := \{(S, S') \mid (S, S') \in \mathcal{S}^2\}$, set of scenario tuples that have been part of a relaxed reduction since the last basic reduction.
2: $\mathcal{S} := \{S \mid (i, S, t_{ep})$ with $t_{ep} \leq t\}$, set of scenarios at the current $x \in V \cup E$ and time step $t \in times_r$ for train i.
3: $q := \{(prio, (S, S')) \mid prio \in \mathbb{Q} \wedge (S \simeq_R S' \vee S \simeq_{RR} S')\}$.

Algorithm 7 Reduction algorithm

1: **procedure** REDUCTION($\mathcal{S}, H$)
2: **if** ($|\mathcal{S}| == 1$) **then return** $\mathcal{S}$; ▷ Termination criterion
3: INITIALIZE($\mathcal{S}$);
4: $q \leftarrow$ UPDATEQUEUE($\mathcal{S}, H$); ▷ Inserts suitable scenario tuples
5: **if** ($|q| > 0$) **then**
6: $(S, S') \leftarrow q[0].pair$;
7: $\mathcal{S} \leftarrow$ APPLYREDUCTION($(S, S'), H$); ▷ Applies reduction step recursively
8: **return** $\mathcal{S}$;

sets $vars, sizes$ to contain only distinct elements and be sorted in an ascending order. We introduce an example set of scenarios to illustrate the initialization procedure.

Example 3.2.1.

$$S_0 : \{p_1 \lhd 2\},\ S_1 : \{p_1 \lhd 0, p_2 \lhd 2\},$$
$$S_2 : \{p_1 \lhd 1, p_3 \lhd 0\},\ S_3 : \{p_1 \lhd 1, p_2 \lhd 1, p_3 \lhd 1\}$$

The procedure further depicted in Algorithm 8 begins in line 2 with defining two maps $w_v : vars \to \mathbb{Q}$, $w_s : sizes \to \mathbb{Q}$, which assign each variable and each scenario size a heuristic value. Furthermore, the preprocessing collects all distinct variables and scenario sizes to compute separate priority values (Lines 3-6).

$$vars = \{p_1, p_2, p_3\},\ sizes = \{1, 2, 3\}$$

Definition 3.2.1 (Heuristic values)**.** *The heuristic values are used to balance the variable selection order. The first deployed function primarily selects reduction pairs with the largest scenario size and only considers the reduced variable if the first criterion is not deciding.*

$$x_s = 0.8$$
$$x_v = min(0.1, \frac{x_s}{|vars|})$$

In line 7 the heuristic values $x_v, x_s \in \mathbb{Q}$ deciding over the priority of variable id and scenario size are computed. The implemented heuristic ensures that the algorithm always prioritizes the scenario size before considering the variable index. The constructed sets are then further used to define the mappings w_v and w_s. In the lines 8-9 and 10-11, the priority maps are defined. The maps are required to map each variable and scenario size to its computed priority.

Algorithm 8 Initialization of Heuristic.

```
1: procedure INITIALIZE(S)
2:     map<int, double> w_s, w_v ← {};
3:     for each S ∈ S do                          ▷ Compute size and variable set
4:         for each {p ◁ S(p)} ∈ S do             ▷ Loop through all random inclusions
5:             vars ← vars ∪ {p};
6:         sizes ← sizes ∪ |S|;
7:     x_v ← 0.8, x_s ← min(0.1, x_v/|vars|);
8:     for i ∈ [1, ..., |sizes|] do               ▷ Define a priority value for each size
9:         w_s.INSERT(sizes[i], x_s·i/|sizes|);
10:    for i ∈ [1, ..., |vars|] do                ▷ Define a priority value for each variable
11:        w_v.INSERT(vars[i], x_v·i/|vars|);
```

Definition 3.2.2 (Weight maps)**.**

$$w_v(i) = \frac{x_v \cdot j}{|vars|},\ \textit{where } i \in vars \textit{ and } j \in 0, ..., |vars| \textit{ with } vars(j) = i$$
$$w_s(i) = \frac{x_s \cdot j}{|sizes|},\ \textit{where } i \in sizes \textit{ and } j \in 0, ..., |sizes| \textit{ with } sizes(j) = i$$

For the above introduced example, the resulting maps are:

$$w_s := \begin{cases} 1 \mapsto 0.266 \\ 2 \mapsto 0.532 \\ 3 \mapsto 0.798 \end{cases}, \quad w_v := \begin{cases} 1 \mapsto 0.033 \\ 2 \mapsto 0.066 \\ 3 \mapsto 0.099 \end{cases} \tag{3.11}$$

The example illustrates the core functionality of the deployed heuristic (Algorithm 8). We decided to use this heuristic to reduce the set of scenarios with a constant order. Additionally based on practical tests during the implementation, the by this computed order defines a reduction order superior to applying reductions without a defined order. Therefore, the heuristic is solely based on practical tests merging same sized scenarios before decreasing the size of a reduced scenario.

Algorithm 9 Update the Priority Queue

```
1: procedure UPDATEQUEUE(S, H)
2:     for each (S, S') ∈ S² with S ≠ S' ∧ (S, S') ∉ H do
3:         if (S ≃_R S' or S ≃_RR S') then
4:             q_e.pair ← (S, S')
5:             q_e.prio ← PRIO(S, S')          ▷ Compute priority value based on heuristic
6:             q.INSERT(q_e)
7:     return p_q
```

The next step of Algorithm 7 is to update the queue q with suitable scenario tuples. This procedure is realized with the function UPDATEQUEUE(S, H) further illustrated in Algorithm 9.

Update Queue

The data structure q_{entry} has two attributes. A priority value $prio \in \mathbb{Q}$, and a scenario pair $(S, S') \in S^2$. The computation of the priority value considers some characteristics of the reduction pair to ensure a consistent execution order.

To ensure that merging steps are applied such that we keep the largest set of suitable scenarios, the presented reduction rules are prioritized individually. Besides the weighting of scenario size and merged variable id, the heuristic additionally includes characteristics of the applied rule and the merging result. Therefore, we introduce an additional function $h(S, S')$, that can take four different values depending on the given reduction instance. The function h orders the possible reduction cases, such that we always merge the information, while not reducing the size of the resulting scenario.
Therefore, we define four cases:

$$h(S, S') := \begin{cases} 1, & \text{for a reduction with a restrictive merging} \\ 2, & \text{for a reduction in which a restriction is completely removed} \\ 3, & \text{for a relaxed reduction with a restrictive merging} \\ 4, & \text{for a relaxed reduction in which a restriction is completely removed} \end{cases}$$

Note here that the step size between the cases is exactly one, this ensures that h is the first deciding factor determining the reduction order before considering variable and scenario size, as the summed mapping values in Equation 3.11 are strictly lower than 0.9. To illustrate the reduction cases we extend the formalization by an example, we use the same support set as earlier $\mathcal{D}(p) = \{0, 1, 2, 3\}$ with $p \in \{p_1, p_2\}$

Example 3.2.2.

$$\begin{aligned} h(S, S') = 1, \ \textit{with}\ & S : \{p_1 \lhd \{0\}, p_2 \lhd \{1\}\}, S' : \{p_1 \lhd \{1\}, p_2 \lhd \{1\}\} \\ & S'' = \{p_1 \lhd \{0, 1\}, p_2 \lhd \{1\}\} \\ h(S, S') = 2, \ \textit{with}\ & S : \{p_1 \lhd \{0, 3\}, p_2 \lhd \{1\}\}, S' : \{p_1 \lhd \{1, 2\}, p_2 \lhd \{1\}\} \\ & S'' = \{p_2 \lhd \{1\}\} \\ h(S, S') = 3, \ \textit{with}\ & S : \{p_1 \lhd \{0, 1\}, p_2 \lhd \{0\}\}, S' : \{p_1 \lhd \{0\}, p_2 \lhd \{1\}\} \\ & \tilde{S} = \{p_1 \lhd \{1\}, p_2 \lhd \{0\}\}, \tilde{S'} : \{p_1 \lhd \{0\} p_2 \lhd \{1\}\} \\ h(S, S') = 4, \ \textit{with}\ & S : \{p_1 \lhd \{0, 1\}, p_2 \lhd \{0, 2\}\}, S' : \{p_1 \lhd \{0\}, p_2 \lhd \{1, 3\}\} \\ & \tilde{S} = \{p_1 \lhd \{1\}, p_2 \lhd \{0\}\}, \tilde{S'} : \{p_1 \lhd \{0\}\} \end{aligned}$$

Given those preliminary parts, we introduce the complete heuristic.

Definition 3.2.3 (Heuristic)**.** *Given a scenario pair (S, S') and a variable p over which we reduce then* $\textsc{Prio}(S, S') \in \mathbb{Q}$ *computes the resulting priority value.*
$\textsc{Prio}(S, S') = h(S, S') + w_v(p) + w_s(|S|)$, *for S, S' with $S \simeq_R S' \vee S \simeq_{RR} S'$.*

After completing the definition of the heuristic, we begin by introducing $\textsc{Update-Queue}(\mathcal{S}, H)$. The function is called with the set of scenarios $\mathcal{S}$ and the set of scenario

Algorithm 10 Reduction Operation

```
1:  procedure ApplyReduction((S, S'), H)
2:      S ← S \ {S, S'}
3:      if (S ≃_R S') then
4:          H ← ∅                                          ▷ Reset set H
5:          Ŝ ← (S \ {p ◁ S(p)}) △ {p ◁ S(p) ∪ S'(p)}      ▷ Apply reduction rule
6:          S ← S ∪ {Ŝ}
7:      else if (S ≃_RR S') then                           ▷ Tuple is relaxed reducible
8:          H ← H ∪ (S, S')                                ▷ Add pair to H
9:          if S(p') ⊊ S'(p') then                         ▷ Case distinction
10:             S̃ ← (S \ {p ◁ S(p)}) △ {p ◁ S(p) ∪ S'(p)}
11:             S̃' ← S' △ {p' ◁ S'(p') \ S(p')}
12:         else if S'(p') ⊊ S(p') then
13:             S̃' ← (S' \ {p ◁ S'(p)}) △ {p ◁ S'(p) ∪ S(p)}
14:             S̃ ← S △ {p' ◁ S(p') \ S'(p')}
15:         S ← S ∪ {S̃, S̃'}
16:     return Reduction(S, H)
```

tuples H that have been recently part of a relaxed reduction. Then in line 3, we loop trough all scenario tuples with differing elements which are not included in H. For each tuple we check in line 3, if the tuple is reducible or relaxed reducible. In case one of the conditions is met, we introduce a new queue entry q_e and set the pair and priority parameter in lines 4-6. After processing all suitable tuples, we return the priority queue q.

In the consecutive step in Algorithm 7, we check if the given queue q does contain at least one element. In case the priority queue is empty, the algorithm has not found a new scenario tuple that is reducible thus the algorithm returns the current set of scenarios and terminates. Otherwise, the q is sorted in an ascending order, and the first queue entry is taken out. Then ApplyReduction$((S, S'), H)$ is called with the in the queue entry contained scenario tuple (S, S') and the current set H (line 7).

Apply Reduction

In the following section we will explain the method ApplyReduction$((S, S'), H)$, which is further depicted in Algorithm 10. The function begins by removing the given scenario tuple out of the set of scenarios $\mathcal{S}$. Next, in line 2 it is evaluated if the given tuple is reducible. In case the basic reduction rule can be applied, we reset H and compute the reduction result in line 5. The computed scenario is then added to $\mathcal{S}$ and we recursively call Reduction$(\mathcal{S}, H)$ in line 16. In the other case, that the scenario tuple is not suited to be reduced with the basic reduction rule, it has to be relaxed reducible. Then the execution jumps to line 8 and includes the current scenario tuple in H.

Next in line 9 we compute the direction of the required containment for the relaxed reduction. Therefore, we check if the support set for the p' of S is contained in the support set for S'. Note here, that we only illustrate the first case, as the other direction is analogous. Then in lines 10-11 the resulting scenarios $\tilde{S}$ and $\tilde{S}'$ are computed as defined in Equation 3.9. Finally, the two newly constructed scenarios $\tilde{S}$ and $\tilde{S}'$ are added to the set of scenarios $\mathcal{S}$. Likewise to the first case, we then

enter the recursion and repeat the procedure until either the set of scenarios contains only one element (line 2 of Algorithm 7) or the priority queue is empty (line 5 of Algorithm 7). The resulting scenario set $\mathcal{S}$ is then returned to ReduceElement(t, x) and in lines 8-9 of Algorithm 5 the current set of occupiers is updated according to the reduced scenario set.

3.3 Proof of Correctness

To deploy the reduction framework in the symbolic simulation, while ensuring the correct execution of the timetable, we have to prove correctness of it. The main invariant of the simulation is that at each time point all participating trains exist with an exact probability of one. Therefore, it has to be ensured that the application of the reduction only compresses the representation of scenarios and it does not alter the information stored in them. In addition to the theoretical proof, we implemented several checks to verify that certain sufficient conditions are always satisfied.

In the following, we assume that during the simulation and before applying the first reduction step, the properties we want to prove are initially satisfied. The main idea of the approach is that given a set of scenarios $\mathcal{S}$, the method returns a possibly reduced set $\mathcal{S}'$.

In order to proof correctness of the approach, we shortly recap the definition of compatible scenarios. Two scenarios S and S' are compatible in case their included values intersect for all variables in P, formally defined as:

$$S \text{ and } S' \text{ are compatible} \leftrightarrow \forall p \in P.\ S(p) \cap S'(p) \neq \emptyset \tag{3.12}$$

Incompatibility can thus be expressed as:

$$S \text{ and } S' \text{ are incompatible} \leftrightarrow \exists p \in P.\ S(p) \cap S'(p) = \emptyset \tag{3.13}$$

Additionally as the proof concept is based on the definition of complete scenarios we include and formalize the definition. A complete scenario has a random inclusion for each variable in the variable set of the simulation and each one is a singleton.

Then, we formalize that a complete scenario s is contained in a given scenario S as:

$$s \blacktriangleleft S \Leftrightarrow s \subseteq \{p \lhd s(p) |\ p \in P.\ |s(p)| = 1 \wedge s(p) \subseteq S(p)\}$$

We extend this notation to define the containment of a complete scenario s in a set of scenarios $\mathcal{S}$.

$$s \blacktriangleleft \mathcal{S} \Leftrightarrow s \blacktriangleleft S_i, \text{ for some } S_i \in \mathcal{S}$$

We have to prove the below-stated properties to show the correctness of the approach. The first property states that neither a complete scenario is lost during the execution nor a new one is constructed.

Property 3.3.1. *In case a complete scenario is part of the given scenario set before the reduction, it has to be part of it afterwards and each complete scenario contained in the reduction result has to be in the input set.*

$$s \blacktriangleleft \mathcal{S} \leftrightarrow s \blacktriangleleft \mathcal{S}'$$

Proof. The reduction rule is applied iteratively, therefore if the property holds for one step, it also holds for the entire reduction process, thus it suffices to prove one iteration step. We begin with a short recap of the reduction result, formally specified in Equation 3.5. In addition, it has to be noted, that the reduction operations for the normal and relaxed reduction are implemented exactly as specified in Equation 3.5 and in Equation 3.9 and can be found in Algorithm 10. Therefore, we directly use the definitions of Chapter 3 to ensure the correctness of the basic reduction rule (Equation 3.5). The result of the basic reduction rule on a given set $\mathcal{S} := \dot{\mathcal{S}} \cup \{S\} \cup \{S'\}$ with $S \simeq_R S'$, can be formalized as:

$$\mathcal{S}' := \dot{\mathcal{S}} \cup \{\tilde{S}\}, \text{ where all scenarios in } \dot{\mathcal{S}} \text{ stay unchanged.}$$
$$\tilde{S} := (S \setminus \{p \lhd S(p)\}) \triangle \{p \lhd S(p) \cup S'(p)\}$$

As $\dot{\mathcal{S}}$ is not modified, the set of included complete scenarios is unchanged. Therefore, we consider only those, which are part of the reduction process and omit $\dot{\mathcal{S}}$.

We begin by proving ($\Rightarrow$) and then add ($\Leftarrow$) to show that the equivalence holds:

($\Rightarrow$) Assume $s \blacktriangleleft \mathcal{S}$:

$$s \blacktriangleleft \mathcal{S} \Rightarrow s \blacktriangleleft \{S\} \cup \{S'\} \cup \dot{\mathcal{S}}$$
$$\Rightarrow s \blacktriangleleft S \triangle \{p \lhd S(p)\} \vee s \blacktriangleleft S' \triangle \{p \lhd S'(p)\} \vee \dot{\mathcal{S}}$$

(I.) $s \blacktriangleleft S \triangle \{p \lhd S(p)\} \Rightarrow s \blacktriangleleft S \setminus \{p \lhd S(p)\} \triangle \{p \lhd S(p) \cup S'(p)\}$
as $(\{p \lhd S(p)\} \preceq \{p \lhd S(p) \cup S'(p)\})$,
$\Rightarrow s \blacktriangleleft \tilde{S} \Rightarrow s \blacktriangleleft \mathcal{S}'$

(II.) $s \blacktriangleleft S' \triangle \{p \lhd S'(p)\} \Rightarrow s \blacktriangleleft S \setminus \{p \lhd S(p)\} \triangle \{p \lhd S(p) \cup S'(p)\}$
as $(\{p \lhd S'(p)\} \preceq \{p \lhd S(p) \cup S'(p)\})$,
$\Rightarrow s \blacktriangleleft \tilde{S} \Rightarrow s \blacktriangleleft \mathcal{S}'$

(III.) $s \blacktriangleleft \dot{\mathcal{S}} \Rightarrow s \blacktriangleleft \mathcal{S}$

($\Leftarrow$) Assume $s \blacktriangleleft \mathcal{S}'$:

$$s \blacktriangleleft \mathcal{S}' \Rightarrow s \blacktriangleleft S \setminus \{p \lhd S(p)\} \triangle \{p \lhd S(p) \cup S'(p)\} \vee s \blacktriangleleft \dot{\mathcal{S}}$$
as $(S(p) \subseteq (S(p) \cup S'(p)) \wedge S'(p) \subseteq (S(p) \cup S'(p)))$,
$$\Rightarrow s \blacktriangleleft S \cup S' \Rightarrow s \blacktriangleleft \mathcal{S}$$

The trivial inclusion of $s \blacktriangleleft \dot{\mathcal{S}}$ in $\mathcal{S}$ has been omitted in the proof of the second direction. □

Property 3.3.2. *Before and after the execution of the reduction each complete scenario is contained at most once. We formalize this as a pairwise incompatibility over all tuples in $\mathcal{S}$, respectively $\mathcal{S}'$.*

$$\forall (S, S') \in \mathcal{S}^2.\ \ (S \neq S' \ \rightarrow S \text{ and } S' \text{ are incompatible}) \wedge$$
$$\forall (S, S') \in \mathcal{S}'^2.\ \ (S \neq S' \ \rightarrow S \text{ and } S' \text{ are incompatible})$$

Proof. As this property is one invariant of the symbolic simulation we can surely assume that the premise does hold before the reduction is applied. Therefore, we proof that the conclusion always follows.

Assume that the result of the reduction $\mathcal{S}'$ does contain a scenario pair, which is compatible. By definition of the structure of $\mathcal{S}'$, we have an unchanged set of scenarios $\dot{\mathcal{S}}$ and the reduction result $\tilde{S}$. As $\dot{\mathcal{S}}$ is not modified, all elements are still pairwise incompatible. The only violation possibility is that some scenario $\dot{S} \in \dot{\mathcal{S}}$ becomes compatible with $\tilde{S}$. In the following p' denotes the reduced random variable.

$$\begin{aligned}
\exists \dot{S} \in \dot{\mathcal{S}}.\ \forall p \in P.\ \dot{S}(p) \cap \tilde{S}(p) \neq \emptyset \Rightarrow \exists \dot{S} \in \dot{\mathcal{S}}, \forall p \in P \setminus \{p'\}.\ \dot{S}(p) \cap S(p) \neq \emptyset \wedge \\
\dot{S}(p) \cap S'(p) \neq \emptyset \wedge \\
\dot{S}(p') \cap \{S(p') \cup S'(p')\} \neq \emptyset \\
\Rightarrow \exists \dot{S} \in \dot{\mathcal{S}}, \forall p \in P.\ \dot{S}(p) \cap S(p) \neq \emptyset \vee \\
\forall p \in P.\ \dot{S}(p) \cup S'(p) \neq \emptyset \\
\Rightarrow \dot{S} \text{ and } S \text{ or } \dot{S} \text{ and } S' \text{ are compatible. } \lightning
\end{aligned}$$

Given that each complete scenario is exactly represented by one scenario in the initial set and the assumption that this does not hold for the reduction result, we were able to derive a contradiction. Therefore we have proven, that the stated premise induces that the resulting scenario set represents each complete scenario at most once. □

Relaxed reduction

Similarly, we prove the correctness of the relaxed reduction. As this proof is more involved, we have to clarify the containment of complete scenarios. The notation we choose denotes any complete scenario contained in a scenario. For the first proof, this did not need to be considered any further as the inclusion sets are completely disjoint. However, the relaxed reduction requires this sort of containment. For the notation of complete scenario, this means, that given an inclusion with at least two elements, we have at minimum two complete scenarios induced by it. This property will be required to prove the correctness of the technique. As before, we have to prove the properties 3.3.1 and 3.3.2, where it is sufficient to prove the properties for one iteration step as the reduction is incremental.

W.l.o.g we assume, for p holds $S(p) \neq S'(p) \wedge S(p) \cap S'(p) = \emptyset$ and for p' with $p' \neq p$ it holds, that $S(p') \subsetneq S'(p')$. Then, for all $\hat{p} \in P \setminus \{p, p'\}$. $S(\hat{p}) = S'(\hat{p})$.

We begin proving that each complete scenario s contained in $S \cup S'$ has to be contained in $\tilde{S} \cup \tilde{S}'$. Formally defined as:

$$s \blacktriangleleft S \cup S' \Rightarrow s \blacktriangleleft \tilde{S} \cup \tilde{S}' \tag{3.14}$$

Proof. The idea of the proof is to show that all complete scenarios contained in $S \cup S'$ are also included in the resulting scenarios $\tilde{S} \cup \tilde{S}'$. Note here that it is sufficient to prove this condition for all variables part of the transformation, thus neglecting all random variables $\hat{p}$, which stay unchanged during the reduction process. In the following, we shortly recap the transformation step of the relaxed reduction to correctly specify $\tilde{S}$ and $\tilde{S}'$ as:

$$\begin{aligned}
\tilde{S} &= (S \setminus \{p \triangleleft S(p)\}) \triangle \{p \triangleleft S(p) \cup S'(p)\}, \\
\tilde{S}' &= S' \triangle \{p' \triangleleft S'(p') \setminus S(p')\}
\end{aligned}$$

further details can be found in Equation 3.8 and Equation 3.9. As the definition of complete scenarios states that for a complete scenario all random variables are

fixed to one value, it is sufficient to prove that the support set values of p and p': $S(p), S'(p)$ and $S(p'), S'(p')$ are contained in $\tilde{S}(p) \cup \tilde{S}'(p)$ and $\tilde{S}(p') \cup \tilde{S}'(p')$ thus the reduction result can construct the same set of complete scenarios as included in the input scenarios set.
We begin by proving the first direction ($\Rightarrow$) of the equivalence in (3.3.1):

$$(\Rightarrow) \text{ Assume } s \blacktriangleleft S \cup S', \text{ then}$$
$$s \blacktriangleleft S \cup S' \Rightarrow \; s \blacktriangleleft S \triangle \{p \lhd S(p)\} \triangle \{p' \lhd S(p')\} \vee \quad (3.15)$$
$$s \blacktriangleleft S' \triangle \{p \lhd S'(p)\} \triangle \{p' \lhd S'(p')\}$$

We split the proof of disjunction 3.15 to separately verify the required containment. The first case is trivially verified, as the complete scenarios s contained in

$$S \triangle \{p \lhd S(p)\} \triangle \{p' \lhd S(p')\}$$

are completely covered by scenario $\tilde{S}$.

$$\textbf{(I.)} \; s \blacktriangleleft S \Rightarrow \; s \blacktriangleleft (S \triangle \{p \lhd S(p)\} \triangle \{p' \lhd S(p')\})$$
$$\textbf{(def.} \; \tilde{S} = S \setminus \{p \lhd S(p)\} \triangle \{p \lhd S(p) \cup S'(p)\})$$
$$\text{as } ((S(p) \subseteq (S(p) \cup S'(p)) \wedge S(p') = S(p')),$$
$$\Rightarrow s \blacktriangleleft \tilde{S} \Rightarrow s \blacktriangleleft \tilde{S} \cup \tilde{S}'$$

In the following we prove the second part of 3.15

$$\textbf{(II.)} \quad s \blacktriangleleft S' \Rightarrow \; s \blacktriangleleft (S' \triangle \{p \lhd S'(p)\} \triangle \{p' \lhd S'(p')\})$$

To correctly prove that all complete scenarios contained in S' are included in the resulting scenarios $\tilde{S}, \tilde{S}'$, we have to show that $S'(p)$ and $S'(p')$ are covered by $\tilde{S}, \tilde{S}'$. We begin with showing that $S'(p)$ is contained in $\tilde{S}$ and $\tilde{S}'$:

$$\text{as } (S'(p) \subseteq (S(p) \cup S'(p)) \wedge S'(p) = S'(p))$$
$$\Rightarrow S'(p) \subseteq \tilde{S} \wedge S'(p) \subseteq \tilde{S}'(p)$$

The second containment holds as S' is unchanged for p due to the construction of $\tilde{S}' := S' \triangle \{p' \lhd S'(p') \setminus S(p')\}$. Next, we have to prove that $S'(p')$ is covered by $\tilde{S}$ and $\tilde{S}'$. Other variables $\hat{p} \in P \setminus \{p, p'\}$ can be neglected in this proof as they stay unchanged during the reduction.

$$\text{as } (S(p') \subseteq S'(p')) \Rightarrow S'(p') = (S'(p') \setminus S(p')) \cup S(p'))$$
$$\tilde{S}(p') \cup \tilde{S}'(p') = (S'(p') \setminus S(p')) \cup S(p'))$$
$$\Rightarrow S'(p') \subseteq \tilde{S}(p') \cup \tilde{S}'(p')$$

thus the support set of p' in S' is covered by $\tilde{S}$ and $\tilde{S}'$. As the support sets for complete scenarios are fixed to size and the required containment of the support sets has been shown we have proven that all complete scenarios that have been in the relaxed reduction input are included afterwards.

Likewise, we prove the second direction ($\Leftarrow$) of Equation 3.14, that every complete scenario included after applying the relaxed reduction had to be in there initially. To

prove this, we show that all complete scenarios contained in $\tilde{S}$ and $\tilde{S}'$ have been contained in the input tuple S, S'.

$$(\Leftarrow) \text{ Assume } s \blacktriangleleft \tilde{S} \cup \tilde{S}'$$
$$s \blacktriangleleft \tilde{S} \cup \tilde{S}' \Rightarrow s \blacktriangleleft (S \setminus \{p \lhd S(p)\}) \triangle \{p \lhd S(p) \cup S'(p)\} \vee$$
$$s \blacktriangleleft S' \triangle \{p' \lhd S'(p') \setminus S(p')\}$$

(I.) $s \blacktriangleleft (S \setminus \{p \lhd S(p)\}) \triangle \{p \lhd S(p) \cup S'(p)\}$

At this point it is sufficient to show the required containment:

$$\text{as } (S(p) = S(p) \wedge S(p') \subsetneq S'(p') \text{ (assumption)})$$
$$\Rightarrow s \blacktriangleleft S \cup S'$$

(II.)
$$s \blacktriangleleft S' \triangle \{p' \lhd S'(p') \setminus S(p)\}$$
$$\text{as } (\{p' \lhd S'(p') \setminus S(p)\}) \preceq \{p' \lhd S'(p')\})$$
$$\Rightarrow s \blacktriangleleft S \cup S'$$

□

In the final section of the proof, we verify that the relaxed reduction does not violate the invariant that each complete scenario is at each time step included at most once in the set of scenarios.

Proof. Let's assume, the given set of scenarios $\mathcal{S}$ is pairwise incompatible. Then as we are able to apply a relaxed reduction, the input set consists out of a scenario tuple involved in the reduction (S, S') and all remaining scenarios $\dot{\mathcal{S}}$.

$$\forall (\dot{S}, \dot{S}') \in \dot{\mathcal{S}}^2.\ \dot{S}|_i \dot{S}' \wedge \forall \dot{S} \in \dot{\mathcal{S}}. \dot{S}|_i S \wedge \dot{S}|_i S' \wedge S|_i S'$$

The reduction method in line 15 of Algorithm 5 constructs a new set $\mathcal{S}' := \dot{\mathcal{S}} \cup \{\tilde{S}\} \cup \{\tilde{S}'\}$. As $\dot{\mathcal{S}}$ is unchanged, all elements are still pairwise incompatible. Additionally, $\tilde{S}'$ is refined to include a smaller set of values, thus it still has to be incompatible with any scenario in $\dot{\mathcal{S}}$. Then, what is left is to prove that $\tilde{S}$ is incompatible with $\tilde{S}'$ and any scenario in $\dot{\mathcal{S}}$. The given reduction result $\tilde{S}, \tilde{S}'$ is incompatible, which can be trivially verified by the structure of the modified scenarios.

$$S(p) \cap (S'(p') \setminus S(p')) = \emptyset$$

The last possible violation of the invariant can occur if $\tilde{S}$ is compatible with an element in $\dot{\mathcal{S}}$. Thus, we have to prove that $\tilde{S}$ is pairwise incompatible with the set of unchanged scenarios. In order to prove this, we assume after applying the relaxed reduction, a scenario $\dot{S}' \in \dot{\mathcal{S}}$ exists, which is compatible with $\tilde{S}$. This would imply that a complete scenario is included twice, which is a violation of the simulations invariant.

Assume $\exists \dot{S}' \in \dot{\mathcal{S}}$, with $\forall \hat{p} \in P.\ \dot{S}'(\hat{p}) \cap \tilde{S}(\hat{p}) \neq \emptyset$, according to the reduction rule (3.9), then in particular $\dot{S}'(p) \cap (S(p) \cup S'(p)) \neq \emptyset$ has to hold. Given that and $S(p) \cap S'(p) = \emptyset$ (3.8), we have to make a case distinction:

1. Either scenario S is intersecting with $\dot{S}'$ over p:

$$S(p) \cap \dot{S}'(p) \neq \emptyset \tag{3.16}$$

Then due to the assumption $\forall \hat{p} \in P.\ \dot{S}'(\hat{p}) \cap \tilde{S}(\hat{p}) \neq \emptyset$, this has to hold for all variables $\hat{p} \in P$ including p' in S that stay unchanged in the reduction process. Together with Equation 3.16 we can conclude that $\dot{S}'$ had to be compatible with S already before applying the relaxed reduction, which contradicts our assumption.

2. Or scenario S' intersects with $\dot{S}'$:

$$S'(p) \cap \dot{S}'(p) \neq \emptyset : \tag{3.17}$$

Arguing analogous to in the first case, Equation 3.17 together with assumption $\forall \hat{p} \in P.\ \dot{S}'(\hat{p}) \cap \tilde{S}(\hat{p}) \neq \emptyset$ implies that $\dot{S}', S'$ would already have been compatible. As in the first case we were able to derive a contradiction assuming the invariant is violated.

Therefore have proven that in case the given input set is pairwise incompatible, the relaxed reduction does not alter this. Thus, we have proven that both properties do hold also for the relaxed reduction. □

Split operation

In the final section of the proof, the presented split operation is verified to not violate any of the stated properties. Even though the presented approach does not invoke the split operation it is implemented in the bounded model checking approach and thus has to be proven. Nevertheless, as the operation is rather intuitive the proof is short. The complete definition of the split operation is depicted in Equation 3.10. For the proof, we shortly recap the induced transformation. Given a scenario S with a random inclusion containing at least two values for a random variable p, then the scenario can be split into two sub-scenarios with two disjoint inclusions for p covering the complete inclusion of S.

Proof. We begin by proving property 3.3.1, stating that each complete scenario contained in S before the split has to be contained in the resulting scenarios $S' \cup S''$ after the split operation.

$$s \blacktriangleleft S \Leftrightarrow s \blacktriangleleft S' \cup S''$$

Assume that $s \blacktriangleleft S$ holds. Due to the definition of the split operation, the random inclusions of the output scenarios S' and S'' are identical for all variables $p' \in P \setminus \{p\}$. As the definition of complete scenarios states that each random variable has to be fixed to one value, this has to hold also for p. Therefore, we only have to prove that all fixed valuations that can be constructed with $S(p)$ can be reproduced by $S'(p) \cup S''(p)$. As this holds trivially due to $S(p) = S'(p) \cup S''(p)$, the first part of the proof is completed. To complete the proof, we have to extend it by showing the other direction. Therefore, we begin with $s \blacktriangleleft S' \cup S''$ and derive the conclusion that all complete scenarios have to be included in S. The reasoning is analogous to the first direction. □

Next, we have to prove that the split operation does not violate the second property (3.3.2), stating that after applying the operation no complete scenario is contained twice.

Proof. Let's assume that the result of the split operation S', S'' contains a complete scenario twice. Then, there has to be a scenario tuple that is compatible. Assume S' or S'' is compatible with an scenario $\dot{S}$ in the set of scenarios not part of the split operation $\dot{\mathcal{S}}$. In the following w.l.o.g we assume S' to be compatible with $\dot{S}$. Then, $\forall p \in P.\ S'(p) \cap \dot{S}(p) \neq \emptyset$, however $S'(p) \preceq S(p) \Rightarrow S(p) \cap \dot{S} \neq \emptyset$. This would imply that the initial set of scenarios already violates the invariant, which contradicts our premise. Next assume that the split result S', S'' is compatible, then $S'(p) \cap S''(p) \neq \emptyset$. However, per definition of the split operation $S'(p) \cap S''(p) = \emptyset$. $\lightning$

Thus, as we were able to derive in both cases a contradiction assuming the split result violates the property, the premise has to hold. □

Termination

Note here, that the split operation is omitted in the following proof, as it is not part of the incremental reduction approach. Further, details on the termination of the split operation are given in Chapter 4. To prove completeness, we have to ensure that the approach terminates on each input instance. As the algorithm is based on recursion, we define an order relation between different input instances. The relation considers the size of the given set $\mathcal{S}$, which is decremented each time a reduction is applied and the set of already applied relaxed reductions H. However, in case only a relaxed reduction is executed the cardinality of the set is not lowered, rather the set of tuples considered for a relaxed reduction is restricted further. This is due to H storing already applied tuple combinations, since the last basic reduction application.

Definition 3.3.1. *Order relation Let* $\mathcal{S}, H \ll \mathcal{S}', H'$*, iff* $|\mathcal{S}| < |\mathcal{S}'| \vee |\mathcal{S}| = |\mathcal{S}'| \wedge |H| > |H'|$*, where* H *respectively is the set of scenario pairs already applied in an earlier relaxed reduction.*

Proof. To prove termination we step-wise analyze the procedure. In each Reduction$(\mathcal{S}, H)$ call, the heuristic values for the current set of scenarios are computed. Inside of Initialize$(\mathcal{S})$, we loop over the set of scenarios to collect all required information. Thus, as the input set is always finite, the execution of the initialize procedure runs in a limited time. Then, the algorithm executes UpdateQueue$(\mathcal{S}, H)$. The purpose of Algorithm 9 is the addition of all suitable scenario tuples to a global priority queue. Note here, that those are scenario tuples that are either reducible or relaxed reducible. As depicted in the pseudo code in line 3, the algorithm runs over all scenario tuples and appends all pairs fulfilling 3.4 or 3.8. The termination of this procedure is trivial, as there exist only limited scenario tuples for the given input set $\mathcal{S}$. The application of the reduction and corresponding recursion found in Algorithm 7 is more complex and requires a deeper analysis.

To prove the termination of the recursion, the stopping conditions of the algorithm have to be analyzed. The first halting point is met in case the given scenario set is a singleton (line 2 of Algorithm 7). Note here, that in the implementation this can only occur inside of the recursion stack as the reduction method is not invoked for single scenarios.

Intuitively, the algorithm also terminates in case the priority queue is empty and no tuples were found to be reducible (5 of Algorithm 7). When the approach is limited to the application of the basic reduction rule, this is a sufficient termination criterion. As the reduction cannot be undone with the given methods, there are only finitely many reduction steps possible. However, in case the relaxed reduction

is allowed the execution gets more complicated. This is mainly because the result of the relaxed reduction is suitable to be used in another relaxed reduction. As the resulting scenarios are again fulfilling Equation 3.8 another addition into q has to be prevented. To restrict this redundant behavior we introduce the set H storing all scenario tuples that were recently part of a relaxed reduction. The set H is reset as soon as the algorithm applies a normal reduction, as in this case participating scenarios are modified and can thus be part of not yet tried relaxed reductions. This strategy could be enhanced by only removing scenario tuples if one of the two elements is modified. However, we decided to not increase the complexity of the algorithm further, as the rather small input examples rarely produce such cases. In future work, this could, however, be further improved. In all other cases the priority queue is not empty (line 2) and the set of scenarios is greater than one (line 5). Then, the next reduction step is applied. To cover the whole functionality of the technique, we separate the proof in a part for the normal (Equation 3.5) and relaxed reduction (Equation 3.9).

We further prove that according to Definition 3.3.1 each recursive call results in a strictly smaller instance. In line 7 of Algorithm 7 the reduction method is given the currently prioritized pair (S, S'). Let $S \simeq_R S'$ hold, then the reduction method reduces the size of the given scenario set $\mathcal{S}$ by exactly one. The resulting set $\mathcal{S}'$ is then given to a recursive execution of the reduction method. As the cardinality of $\mathcal{S}$ is reduced $\mathcal{S}' \ll \mathcal{S}$ holds.

Otherwise, if $S \simeq_{RR} S'$ holds, a relaxed reduction is applied, which does not remove a scenario rather modifying the existing ones. In line 8 of Algorithm 5 the used combination is then added to H to ensure that the applied change is not undone in the successive application. Inside of UpdateQueue$(\mathcal{S}, H)$ the possible reduction pairs are verified to not be in H. Hence, by appending elements to H the set of tuples not suitable for a reduction is decreasing up to the point where no further operation is applicable. In addition, according to Definition 3.3.1 the expansion of H leads to a strictly smaller execution instance. In case no new pair can be queued the current set of scenarios is returned and the recursive stack is resolved. Consequently, each possible operation constructs a strictly smaller instance according to Definition 3.3.1. And as the set of scenarios and the number of possible scenario tuples are finite, the algorithm terminates in finite time. □

Chapter 4

Bounded Model Checking Approach

In the previous chapter, we illustrated an approach to reduce scenarios. The basic reduction technique together with the relaxed version showed promising results in accelerating the symbolic simulation while increasing scalability by decreasing the set of states. Nevertheless, the reduction approach could not be extended to exploit the complete reduction potential without introducing complex loop prevention, optimized heuristics and a backtracking algorithm. Therefore, we decided to implement another approach to evaluate the achieved acceleration of a reduction method making use of the complete reduction potential. We choose to transform the reduction methods into a SAT encoding. The SAT encoding is consecutively solved with a bounded model checking approach (BMC) using satisfiability checking (SAT) [CBRZ01].

The idea of bounded model checking is to construct a boolean formula that is true if and only if the underlying transition system can execute a finite sequence with a fixed number of steps leading to a predefined goal state. In case the transition system does not have such a sequence in the allowed iteration depth, normally one would increment the number of allowed steps and reapply the approach. However, as this can lead to a running time increase contradicting this thesis purpose, we evaluate different techniques to handle this, further explained in Section 4.2. After defining the boolean formula it is given to a satisfiability solver, which returns either SAT with a satisfying assignment or UNSAT. In the context of this thesis, we decided to use the Minisat implementation, as it provides an intuitive *C++* interface.

Important to note here is, that the satisfiability solving method is not complete, as the used hardware has boundaries that could be reached dependent on the formula size. This has to be kept in mind with special regard to the allowed unfolding steps for the transition system.

In order to correctly generate the input formula, we formalize the transition system as:

$$[\![M]\!]_K := I_0 \wedge \bigwedge_{k=0}^{K-1} T(k, k+1) \wedge goal_K,$$

where I_0 is the initial state encoding, $K \in \mathbb{N}$ is the iteration depth, $T(k, k+1)$ encodes possible transitions between iterations and $goal_K$ defines the goal state. In

the following chapter, the SAT encoding for the reduction operation is depicted and the corresponding transition system is constructed.

4.1 SAT Encoding

The encoding of reduction methods is not enough to correctly depict the procedure. Thus, we define an interface between simulation and reduction operation. As before, in the basic reduction approach, the method's input is a set of scenarios matching strict requirements. The given data structure has to be transformed into a boolean formula and is correspondingly encoded in I_0.

Consequently to encode the initial state, the given scenario set is transformed into a boolean clause set. The used variables are divided into different sets of states depending on the current iteration step of the bounded model checking. Therefore, we introduce an index $k \in [0, ..., K]$ denoting the current state, beginning with $k = 0$ and ending with a fixed bound K. Additionally, the variables are indexed according to the scenario $i \in [1, ..., n]$ with $n = |\{S_1, ..., S_n\}|$, the random variable $v \in [1, ..., m]$ with $m = |P|$ and the domain value $d \in \mathcal{D}(p_v)$.
The semantics are defined as follows:

$$a_{k,i,v,d} = \begin{cases} 1, & \text{if } d \in S_i(p_v) \text{ in step } k \\ 0, & \text{else} \end{cases}$$

For the initial state I_0, we add unary clauses with the corresponding variables $a_{0,i,v,d}$ to encode the given scenarios. The function $\phi(i,v,d)$ is required to define the initial state.

$$\phi(i,v,d) = \begin{cases} a_{0,i,v,d}, & \text{if } d \in S_i(p_v) \\ \neg a_{0,i,v,d}, & \text{else} \end{cases}$$

$$I_0 := \bigwedge_{i=1}^{n} \bigwedge_{v=1}^{m} \bigwedge_{d \in \mathcal{D}(p_v)} \phi(i,v,d)$$

Example 4.1.1.

$$\mathcal{D}(p_i) = \{0,1\} \textit{ for } i \in [0,1]$$
$$S_1 : \{p_1 \lhd \{0,1\}, p_2 \lhd \{1\}\}$$
$$S_2 : \{p_1 \lhd \{0,1\}, p_2 \lhd \{0\}\}$$

$$I_0 = a_{0,1,1,0} \wedge a_{0,1,1,1} \wedge \neg a_{0,1,2,0} \wedge a_{0,1,2,1} \wedge$$
$$a_{0,2,1,0} \wedge a_{0,2,1,1} \wedge a_{0,2,2,0} \wedge \neg a_{0,2,2,1}$$

Further, we need an encoding to state that scenario tuples are reducible in a specific iteration state k. Scenario indices are denoted by $i, j \in [1, ..., n]$ with $i < j$ to avoid symmetric cases. We introduce a set of boolean variables to define that a scenario pair (i, j) is reducible over a certain variable v. To define this condition, we include additional variables $eq_{k,i,j,v,d}$ stating that valuations for a scenario pair (i, j),

a variable v and an initial delay d are equivalent. This can then be further used to describe the exact reduction requirement.

$$\begin{aligned} equiv(k,i,j,v,d) :&(\neg eq_{k,i,j,v,d} \vee \neg a_{k,i,v,d} \vee a_{k,j,v,d}) \wedge \\ &(\neg eq_{k,i,j,v,d} \vee a_{k,i,v,d} \vee \neg a_{k,j,v,d}) \wedge \\ &(eq_{k,i,j,v,d} \vee \neg a_{k,i,v,d} \vee \neg a_{k,j,v,d}) \wedge \\ &(eq_{k,i,j,v,d} \vee a_{k,i,v,d} \vee a_{k,j,v,d}) \end{aligned}$$

The boolean encoding of the reduction rule states that the random inclusions for all variables v' except for one variable v have to be identical. This can be formalized by encoding that for all variables except one, the set of equivalence defining variables is satisfied. To encode this, we introduce a set variables $r_{k,i,j,v}$ for each iteration state k, a scenario tuple (i,j) and the variable v.

Owing to the global invariant of the algorithm, that no complete scenario is contained twice, the random inclusion for the excluded variable has to be disjoint. Thus, the formula below is only true for reducible scenario tuples, where p_v is part of the differing random inclusion.

$$\begin{aligned} reducible(k,i,j,v) : &\bigwedge_{\substack{v' \in [1,m] \\ v' \neq v}} \bigwedge_{d \in \mathcal{D}(p_{v'})} (eq_{k,i,j,v',d} \vee \neg r_{k,i,j,v}) \wedge \\ &\bigvee_{\substack{v' \in [1,m] \\ v' \neq v}} \bigvee_{d \in \mathcal{D}(p_{v'})} (\neg eq_{k,i,j,v',d}) \vee r_{k,i,j,v} \end{aligned}$$

We introduce additional variables to denote that a certain pair (i,j) of scenarios is reduced in step k $r_{k,i,j}$.

$$reduce(k,i,j) : (\neg r_{k,i,j} \vee \bigvee_{v=1}^{m} r_{k,i,j,v}) \tag{4.1}$$

The variable $r_{k,i,j}$ represents the possibility of applying a reduction to a specific pair and in case it is satisfied one variable in i and j has to be suitable for a reduction. This is ensured by the disjunction over all possible variables in Equation 4.1. Moreover, to determine for each scenario if it is reduced in the current iteration step a variable $r_{k,i}$ solely depending on iteration state k and scenario index i is introduced.

$$reduce(k,i) : (\neg r_{k,i} \vee \bigvee_{j=0}^{i-1} r_{k,j,i} \bigvee_{j=i+1}^{n} r_{k,i,j}) \wedge \bigwedge_{j=0}^{i-1} (\neg r_{k,j,i} \vee r_{k,i}) \wedge \bigwedge_{j=i+1}^{n} (\neg r_{k,i,j} \vee r_{k,i})$$

As we want to ensure that each scenario i is reduced only once in each iteration step, we begin by defining the set of reduction variables $R_{k,i}$ for the current step k. Then, we introduce an at most one constraint using the Bimander encoding [NM15], ensuring that no scenario is reduced twice. The set of reduction variables for a scenario i is defined as:

$$R_{k,i} := \{r_{k,i,j} |\ \forall j.\ i < j \text{ with } i,j \in [1,...,n]\} \cup \{r_{k,j,i} |\ \forall j.\ j < i \text{ with } i,j \in [1,...,n]\}$$

In the following we shortly depict the encoding of the at most once constraint, for further details we refer to [NM15].

Given the set of variables $x_1, ..., x_{\overline{n}} \in R_{k,i}$, we separate those in $\overline{m}$ disjoint groups $G_1, ..., G_{\overline{m}}$ of size $\overline{k}$ with $1 \leq \overline{m} \leq \overline{n}$, such that $\overline{k} = \lceil \frac{\overline{n}}{\overline{m}} \rceil$. By extending the indexing to $x_{i,h}$, we state that $x_{i,h}$ is the h^{th} element of group i.

Then, we encode that for each group at most one variable can be true. This is realized by applying the pairwise encoding to all groups $G_1, ..., G_{\overline{m}}$. The encoding technique often referred to as *naive* encoding is an intuitive strategy to ensure that at most one variable is assigned to true. The idea is to encode that in all possible combinations no two variables are satisfied at the same time. This is realized by encoding a set of clauses consisting of all negated variable tuples in G_i, as this encoding is rather trivial we abbreviate the construction by At-most-one$_{pw}$. As this is only applied to the defined subsets of fixed size, the rather simple encoding suits as an efficient method.

$$\bigwedge_{i=1}^{\overline{m}} (\text{At-most-one}_{pw}(G_i))) \tag{4.2}$$

Additionally, we introduce a constraint pair between each variable $x_{i,h} \in G_i$ of a group and a set of new variables further denoted as commander variables $B_1, ..., B_{\lceil log_2 \overline{m} \rceil}$. The commander variables are represented by:

$$\sigma(i,j) := \begin{cases} B_j, & \text{if bit } j \text{ of the binary string } i-1 \text{ is 1.} \\ \neg B_j, & \text{else} \end{cases}$$

$$\bigwedge_{i=1}^{\overline{m}} \bigwedge_{h=1}^{\overline{k}} \bigwedge_{j=1}^{\lceil log_2 \overline{m} \rceil} \neg x_{i,h} \vee \sigma(i,j) \tag{4.3}$$

Together Equation 4.2 and Equation 4.3 define, that at most one of the variables in $R_{k,i}$ is true, ensuring that each scenario is reduced at most once in each iteration step. We introduce the notation At-most-one$(R_{k,i})$ to abbreviate the described encoding.

In the following, we allow multiple reductions while reducing each scenario at most once in each step. We introduce a transition relation $T(k,k+1)$ consisting of all required constraints for transition step k to $k+1$.

$$\begin{aligned} T(k,k+1) := & \bigwedge_{i=1}^{n-1} \bigwedge_{j=i+1}^{n} \bigwedge_{v=1}^{m} reducible(k,i,j,v) \wedge \bigwedge_{i=1}^{n-1} \bigwedge_{j=i+1}^{n} reduce(k,i,j) \wedge \\ & \bigwedge_{i=1}^{n} reduce(k,i) \wedge \bigwedge_{i=1}^{n-1} \bigwedge_{j=i+1}^{n} \bigwedge_{v=1}^{m} \bigwedge_{d \in \mathcal{D}(p_v)} equiv(k,i,j,v,d) \wedge \bigwedge_{i=1}^{n-1} \bigwedge_{j=i+1}^{n} (\neg r_{k,i,j} \vee \\ & (\bigwedge_{v \in [1,m]} \bigwedge_{d \in \mathcal{D}(p_v)} (a_{k+1,i,v,d} \leftrightarrow (a_{k,i,v,d} \vee a_{k,j,v,d})) \wedge \neg a_{k+1,j,v,d})) \wedge \\ & \bigwedge_{i=1}^{n} (r_{k,i} \vee \bigwedge_{v \in [1,m]} \bigwedge_{d \in \mathcal{D}(p_v)} a_{k+1,i,v,d} \leftrightarrow a_{k,i,v,d}) \wedge \text{At-most-one}(R_{k,i}) \end{aligned}$$

$$\equiv \text{(transformation to CNF)}$$

$$T(k,k+1) := \bigwedge_{i=1}^{n-1} \bigwedge_{j=i+1}^{n} \bigwedge_{v=1}^{m} reducible(k,i,j,v) \wedge \bigwedge_{i=1}^{n-1} \bigwedge_{j=i+1}^{n} reduce(k,i,j) \wedge$$
$$\bigwedge_{i=1}^{n} reduce(k,i) \wedge \bigwedge_{i=1}^{n-1} \bigwedge_{j=i+1}^{n} \bigwedge_{v\in[1,m]} \bigwedge_{d\in\mathcal{D}(p_v)}$$
$$(\neg a_{k+1,i,v,d} \vee a_{k,i,v,d} \vee a_{k,j,v,d} \vee \neg r_{k,i,j}) \wedge$$
$$(a_{k+1,i,v,d} \vee \neg a_{k,i,v,d} \vee \neg r_{k,i,j}) \wedge$$
$$(a_{k+1,i,v,d} \vee \neg a_{k,j,v,d} \vee \neg r_{k,i,j}) \wedge$$
$$(\neg a_{k+1,j,v,d} \vee \neg r_{k,i,j}) \wedge$$
$$\bigwedge_{i=1}^{n} \bigwedge_{v\in[1,m]} \bigwedge_{d\in\mathcal{D}(p_v)} ((r_{k,i} \vee \neg a_{k+1,i,v,d} \vee a_{k,i,v,d}) \wedge$$
$$(r_{k,i} \vee a_{k+1,i,v,d} \vee \neg a_{k,i,v,d})) \wedge \text{At-most-one}(R_{k,i})$$

As we want to quantify a certain state of interest, which is the optimal amount of applied reductions, we include a cardinality constraint demanding empty scenarios. In the context of this thesis, the optimal amount of applied reductions is the sequence with the best trade-off between reduction result and required time. Empty scenarios are the direct result of applied reductions. For the following equation we assume that we have a set of variables E and $\neg E$ denotes the set in which all variables of E are negated. Additionally, we assume that E has exactly n elements. Then, this equivalence holds:

$$\text{At-most-}(n\text{-}c)(E) \equiv \text{ At-least-}(c)(\neg E).$$

To define the goal state we introduce variables representing non-empty scenarios, such that we can make use of the equivalence by encoding an at most n-c non-empty scenarios clause set to model at least c empty scenarios. To detect empty scenarios we include for each state k and each scenario i a variable $e_{k,i}$, which is true iff the first variable ($a_{k,i,1,d}$) is satisfied for at least one element of its support set.

$$e_{k,i} \leftrightarrow \bigvee_{d\in\mathcal{D}(p_1)} a_{k,i,1,d}$$

$$\equiv \text{(transformation to CNF)}$$

$$empty(k,i) := (\bigvee_{d\in\mathcal{D}(p_1)} \neg a_{k,i,1,d} \vee \neg e_{k,i}) \wedge \bigwedge_{d\in\mathcal{D}(p_1)} (e_{k,i} \vee \neg a_{k,i,1,d})$$

The introduced variables $e_{k,i}$ are true, if in step k in scenario i at least one element is inside of the random inclusion for p_1. The set of empty scenario variables in iteration depth k is defined as $E_k := \{e_{k,i} | \text{ for } i \in [1,...,n]\}$.

The final step of the encoding is the quantification of empty scenarios, thus we include a constraint stating that there have to be at least c empty scenarios in state K.

The encoding of at least c empty scenarios is realized with an encoding defining at most n-c non-empty scenarios.

The at-most n-c constraint is denoted by At-most-n-$c(x_1, ..., x_n)$ for $x_i \in E_{K,i}$ and realized with the help of a sequential counter encoding (SQ_{enc}). Note here, that we have to introduce an additional encoding as the Sequential encoding ([NM15]) showed to be more efficient for at-most-k constraints with $k > 1$. The SQ_{enc} is described further in the following section, for more specific information on the complexity and a corresponding evaluation of the encoding, we refer to [Sin05].

The basic idea of the encoding is a sequential counter that is incremented for each satisfied variable and violated in case more than $c' = n$-c variables are assigned true. The encoding requires a set of variables $s_1, ..., s_n$ in which the variables represent the partial sum of satisfied variables up to i: $\sum_{j=1}^{i} x_j$ in unary representation. The notation is extended to $s_{i,j}$ to represent the j^{th} bit of partial sum i.

In the first partial sum s_1 the first bit is set to x_1 and all other bits are trivially null:

$$(\neg x_1 \vee s_{1,1}) \tag{4.4}$$
$$\wedge\ (\neg s_{1,j}) \quad \text{for } 1 < j \leq c' \tag{4.5}$$

In case x_i is set to true, the first bit of s_i has to be set to true, independent of the previous entries.

$$\wedge\ (\neg x_i \vee s_{i,1}) \quad \text{for } 1 < i \leq n \tag{4.6}$$

Then to ensure the correct counting, partial sums have to be carried over between successive sums, with special consideration of overflow bits.

$$\wedge\ (\neg s_{i-1,1} \vee s_{i,1}) \quad \text{for } 1 < i \leq n \tag{4.7}$$
$$\wedge\ (\neg x_i \vee \neg s_{i-1,j-1} \vee s_{i,j}) \quad \text{for } 1 < i \leq n \tag{4.8}$$
$$\wedge\ (\neg s_{i-1,j} \vee s_{i,j}) \quad \text{for } 1 < j \leq c' \tag{4.9}$$

Finally, the encoding specifies that the c'^{th} bit of the partial sum s_i is never set to one together with a satisfied x_i, as this would indicate that more than c' variables are true. In those cases the encoding is unsatisfiable.

$$\wedge\ (\neg x_i \vee \neg s_{i-1,c'}) \quad \text{for } 1 < i \leq n \tag{4.10}$$

Given all encoding details we can specify the transition system M for the bounded model checking approach.

$$[\![M]\!]_K := I_0 \wedge \bigwedge_{k=0}^{K-1} T(k, k+1) \wedge \bigwedge_{i=1}^{n} empty(K, i) \wedge \text{ At-most-}(n\text{-}c)(E_k), \tag{4.11}$$
where $K \in \mathbb{N}$ is the iteration depth.

$[\![M]\!]_K$ is then unfolded up to K and given to a SAT Solver.

Split-Operation

We introduce a new set of variables $sp_{k,i,j,v}$ to denote that a scenario i is split and the sub-scenarios are stored in i and j. The condition for a scenario to be split is extended to a certain variable v.

For a better understanding the logical requirements for the split operation and the corresponding CNF formulas are listed down below:

First, we specify the requirements for a scenario to be split. As we have to store the split result in a distinct scenario, the scenario with index j has to be empty. Additionally, the random inclusion of v in i has to have a cardinality of at least two. The random inclusion for v in i is defined as: $D_{k,i,v} := \{a_{k,i,v,d} | \text{ for } d \in \mathcal{D}(v)\}$

$$\bigwedge_{d \in \mathcal{D}(v)} \neg a_{k,i,1,d} \wedge \text{at-least-two}(D_{k,i,v})$$

The encoding of at-least-two encodes all possible $|D_{k,i}| - 1$ sized clauses of variables in $D_{k,i,v}$ and conjuncts those. This results in $|D_{k,i}|$ clauses and ensures that by assigning one variable all but one clause are satisfied, thus requiring at least two satisfied variables. Given a scenario pair fulfilling the above condition, the split operation can be applied. As we require a general encoding without restricting possible split applications, we formalize the operation step by step. The first condition for the split result is that both scenarios i, j are non-empty afterwards.

$$\begin{aligned}&(\bigvee_{d \in \mathcal{D}(v)} a_{k+1,i,1,d}) \wedge \\ &(\bigvee_{d \in \mathcal{D}(v)} a_{k+1,j,1,d})\end{aligned}$$

The split operation cannot modify the contained complete scenarios, thus everything which was part of i in k has to be covered by i and j in $k+1$.

$$\bigwedge_{d \in \mathcal{D}(v)} a_{k,i,v,d} \leftrightarrow (a_{k+1,i,v,d} \vee a_{k+1,j,v,d})$$

Then, to ensure that no scenario is part of i and j a constraint specifies, that at least one of both has to be false.for the split variable v.

$$\bigwedge_{d \in D(v)} \neg a_{k+1,i,v,d} \vee \neg a_{k+1,j,v,d}$$

Finally, we have to ensure that for all other variables unequal to v, the resulting scenarios stay unchanged.

$$\begin{aligned}&\bigwedge_{\substack{v' \in [1,m] \\ v' \neq v}} \bigwedge_{d \in D(v')} (a_{k+1,j,v',d} \leftrightarrow a_{k+1,i,v',d}) \wedge \\ &\bigwedge_{\substack{v' \in [1,m] \\ v' \neq v}} \bigwedge_{d \in D(v')} (a_{k+1,i,v',d} \leftrightarrow a_{k+1,i,v',d}) \wedge\end{aligned}$$

The required conditions have to be connected to the split variable. Therefore, we define an implication from $sp_{k,i,j,v}$ to all split clauses. For completeness we formalize

the split-encoding below:

$$\begin{aligned} split(k,i,j,v) :=& sp_{k,i,j,v} \rightarrow \bigwedge_{d\in\mathcal{D}(v)} (\neg a_{k,i,j,1,v} \wedge \text{at-least-two}(D_{k,i,v})) \wedge \\ & (\bigvee_{d\in\mathcal{D}(v)} a_{k+1,i,1,d}) \wedge (\bigvee_{d\in\mathcal{D}(v)} a_{k+1,j,1,d}) \wedge \\ & \bigwedge_{d\in\mathcal{D}(v)} a_{k,i,v,d} \leftrightarrow (a_{k+1,i,v,d} \vee a_{k+1,j,v,d}) \wedge \\ & \bigwedge_{d\in D(v)} (\neg a_{k+1,i,v,d} \vee \neg a_{k+1,j,v,d}) \wedge \\ & \bigwedge_{\substack{v'\in[1,m]\\ v'\neq v}} \bigwedge_{d\in D(v')} (a_{k+1,j,v',d} \leftrightarrow a_{k+1,i,v',d}) \wedge \\ & \bigwedge_{\substack{v'\in[1,m]\\ v'\neq v}} \bigwedge_{d\in D(v')} (a_{k+1,i,v',d} \leftrightarrow a_{k+1,i,v',d}) \end{aligned}$$

Additionally to the defined constraints, we add a number of clauses to ensure the correctness with the remaining part of the SAT encoding. Therefore, we begin by adding variables for each scenario to denote that the scenario is split in state k. The variables are denoted by $sp_{k,i}$ and are assigned true in case i is part of a split.

$$split(k,i) := (sp_{k,i} \leftrightarrow (\bigvee_{j=1}^{i-1} \bigvee_{v\in[1,m]} sp_{k,j,i,v} \vee \bigvee_{j=i+1}^{n} \bigvee_{v\in[1,m]} sp_{k,i,j,v}))$$

In every iteration step k a scenario can be either split or reduced, not both.

$$\text{At-most-one}_a(k) := \bigwedge_{i=1}^{n} (\neg r_{k,i} \vee \neg sp_{k,i}) \tag{4.12}$$

The violet part of the transition has to be adjusted, as a scenario assignment only stays unchanged in case no reduction and no split is performed.

$$\bigwedge_{i=1}^{n} (r_{k,i} \vee sp_{k,i} \vee \bigwedge_{v\in[1,m]} \bigwedge_{d\in\mathcal{D}(p_v)} a_{k+1,i,v,d} \leftrightarrow a_{k,i,v,d})$$

Finally, analog to the reduction, it has to be ensured that no scenario is part of more than one split. Correspondingly, we add an at-most-one constraint using the Bimander encoding [NM15]. The used variable set is defined as:

$$\begin{aligned} SP_{k,i} :=& \{sp_{k,i,j,v} |\ \forall j \in [1,...,n], v \in [1,...,m].\ \text{with } i < j\} \cup \\ & \{sp_{k,i,j,v} |\ \forall j \in [1,...,n], v \in [1,...,m].\ \text{with } j < i\} \end{aligned}$$

The at-most-one encoding is abbreviated by:

$$\text{At-most-one}(SP_{k,i})$$

Before formalizing the transition system $[\![M']\!]_K$ including the split operation, we have to determine a sequence of empty scenarios that have to be included to store the split

scenarios. As a general rule, we add $\lceil \frac{n}{2} \rceil$ empty scenario in the first state $k = 0$ to restrict the split possibilities and improve the solving speed. Correspondingly, the at least c empty condition is incremented to hold for at least $c + \lceil \frac{n}{2} \rceil$, as the empty scenarios do not count in the reduction process.

While implementing and evaluating the SAT encoding, we realized that the split operation expanded the search space by a significant factor, hence we define a set of enhancements to rule out redundant parts of the solution space.

The first improvement is an additional constraint to rule out that a reduction is undone by applying a consecutive split and analogous a split should not be undone by a reduction. This is encoded by the following clauses.

$$Impr(k) := \bigwedge_{i=1}^{n-1} \bigwedge_{j=i+1}^{n} \bigwedge_{v \in [1,m]} (\neg r_{k,i,j,v} \vee \neg sp_{k+1,i,j,v}) \wedge (\neg r_{k,i,j,v} \vee \neg sp_{k+1,j,i,v}) \wedge (\neg sp_{k,i,j,v} \vee \neg r_{k+1,i,j,v}) \wedge (\neg r_{k+1,i,j,v} \vee \neg sp_{k+1,j,i,v}) \tag{4.13}$$

Moreover, we introduce a new variable τ_k for $k \in [1, ..., K]$, which is set to true in case no further reduction is applicable and no split is applied. Formally, τ_k is defined as:

$$(\tau_k \vee \bigvee_{i=1}^{n-1} \bigvee_{j=i+1}^{n} \bigvee_{v=1}^{m} r_{k,i,j,v} \vee \bigvee_{i=1}^{n-1} \bigvee_{j=i+1}^{n} \bigvee_{v=1}^{m} sp_{k,i,j,v}) \wedge (\bigwedge_{i=1}^{n-1} \bigwedge_{j=i+1}^{n} \bigwedge_{v=1}^{m} \neg \tau_k \vee \neg r_{k,i,j,v}) \wedge (\bigwedge_{i=1}^{n-1} \bigwedge_{j=i+1}^{n} \bigwedge_{v=1}^{m} \neg \tau_k \vee \neg sp_{k,i,j,v}), \tag{4.14}$$

This alone is not sufficient to improve the solving speed, hence we conjunct τ_k with all remaining parts of the encoding. The conjunction is trivial and no further explained. Thus in case no further reduction is possible in step k τ_k has to be satisfied and the remaining boolean clause set is trivially propagated to be satisfiable. Additionally, we have to include a constraint that ensures if τ_k is set to true, it has to be satisfied for all following iteration steps: $\forall k' \in [k+1, ..., K].\ \tau_{k'} = true$.

$$\bigwedge_{k=1}^{K} \neg \tau_{k-1} \vee \tau_k \tag{4.15}$$

As the current goal state of the bounded model checking is specified to hold at the last iteration step, we have to include a case distinction to correctly evaluate the assignment with respect to τ_k. Therefore, we adjust the condition to hold in the last iteration step in case all τ_k are assigned false. Otherwise, the constraint for at least c empty scenarios has to hold in the first iteration in which τ_k holds. These adjustments are encoded by:

$$\text{At-most*-}c(E_k) := \bigwedge_{k=2}^{K-1} ((\neg \tau_{k-1} \wedge \tau_k) \rightarrow \text{At-most-}c(E_k) \wedge (\neg \tau_K \rightarrow \text{At-most-}c(E_k))) \tag{4.16}$$

The complete improvement encoding is denoted by $Stop(k)$ and is a conjunction of the separately depicted formulas, note here that it additionally modifies the remaining encoding as described above.

$$Stop(k) := (4.14) \wedge (4.15)$$

Note here, that 4.16 is not included in this definition as it replaces the At-most-$c(E_k)$ condition in the transition system (4.11). The updated transition relation $T(k, k+1)$ can then be encoded by:

$$\begin{aligned} T(k,k+1) := & \bigwedge_{i=1}^{n-1} \bigwedge_{j=i+1}^{n} \bigwedge_{v=1}^{m} reducible(k,i,j,v) \wedge \bigwedge_{i=1}^{n-1} \bigwedge_{j=i+1}^{n} reduce(k,i,j) \wedge \\ & \bigwedge_{i=1}^{n} reduce(k,i) \wedge \bigwedge_{i=1}^{n} \bigwedge_{\substack{j\in[1,n]\\ j\neq i}} \bigwedge_{v=1}^{m} split(k,i,j,v) \wedge \bigwedge_{i=1}^{n} split(k,i) \\ & \bigwedge_{i=1}^{n-1} \bigwedge_{j=i+1}^{n} (\neg r_{k,i,j} \vee (\bigwedge_{v\in[1,m]} \bigwedge_{d\in\mathcal{D}(p_v)} (a_{k+1,i,v,d} \leftrightarrow (a_{k,i,v,d} \vee a_{k,j,v,d})) \wedge \\ & \neg a_{k+1,j,v,d})) \wedge \bigwedge_{i=1}^{n} (r_{k,i} \vee sp_{k,i} \vee \bigwedge_{v\in[1,m]} \bigwedge_{d\in\mathcal{D}(p_v)} a_{k+1,i,v,d} \leftrightarrow a_{k,i,v,d}) \wedge \\ & \text{At-most-one}(R_{k,i}) \wedge \text{At-most-one}(SP_{k,i}) \wedge \text{At-most-one}_a(k) \wedge \\ & Impr(k) \wedge Stop(k) \end{aligned}$$

Note here, that we could also merge the At-most-one conditions in one encoding, however we decided to introduce an additional encoding At-most-one$_a$ to encode that a scenario can only be part of exactly one rule application. This is reasoned by the fact that At-most-one$_a$ includes at most three clauses for each scenario (defined in Equation 4.12 and extended in Equation 4.25), whereas the Bimander encoding has a complexity of $\frac{\overline{n}^2}{2\overline{m}} + \overline{n}\lceil \log_2 \overline{m} \rceil - \frac{\overline{n}}{2}$. A further investigation of a more efficient encoding was out of scope and is listed as future work.

Relaxed reduction

As an additional reduction technique and to match the reduction framework, we decided to transform the relaxed reduction into a SAT encoding. This enables the direct comparison of the approaches to analyze the advantages and disadvantages between them.

The relaxed reduction is encoded below, we separate the encoding in conditions holding before and after the application. We introduce a set of variables to encode that a pair of scenarios $(i, j) \in [1, ..., n]^2$ with a tuple of random variables $(v, v') \in [1, ..., m]^2$ is relaxed reduced.

$$rr_{k,i,j,v,v'} \text{ for } k \in [1, ..., K],\ i, j \in [1, ..., n] \text{ and } v, v' \in [1, ..., m]$$

Prerequisites As explained in detail in an earlier chapter, the relaxed reduction can only be applied to tuples of scenarios inheriting a containment for the random inclusions of v and two disjoint inclusions for v'. Therefore, we encode that variable

v' models the containment of random inclusions, while the corresponding values for v have to be disjoint, modeling the reduction opportunity. This encoding strategy is rigid, however, it is required to correctly specify the corresponding transformations. Note here, that resulting formulas are encoded separately to improve the readability for the reader. However, in the implementation, the relaxed reduction is included as a single implication.

The containment is directed, hence the set corresponding to the random inclusion of $p_{v'}$ in j includes the one in i. Therefore, the tuple (i, j) differs in comparison to (j, i). The encoding is formalized as:

$$\bigwedge_{i=1}^{n} \bigwedge_{\substack{j\in[1,n]\\ j\neq i}} \bigwedge_{v\in[1,m]} \bigwedge_{\substack{v'\in[1,m]\\ v'\neq v}} (\neg rr_{k,i,j,v,v'} \vee \bigvee_{d\in D(v')} \neg eq_{k,i,j,v',d}) \wedge \tag{4.17}$$

$$\bigwedge_{d\in D(v')} (\neg rr_{k,i,j,v,v'} \vee \neg a_{k,i,v',d} \vee a_{k,j,v',d}) \wedge (\bigvee_{d\in D(v')} a_{k,i,v',d})$$

The idea of the encoding is that in case the relaxed reduction is applicable, one of the equivalences for containment variable v' has to be violated. Hereby, we ensure that the two inclusions for v' are unequal. Furthermore, the second clause in 4.17 secures that in case a value d is not in j, it cannot be in i. With the third clause we ensure that i is not empty and thus a real containment is present.

Additionally, we specify the structure of the remaining variables $\dot{v}$. For all variables $\dot{v} \in [1, ..., m]$ with $\dot{v} \in V \setminus \{v, v'\}$, the corresponding equivalence variables have to hold. This ensures that the scenarios are identical neglecting v and v'.

$$\bigwedge_{i=1}^{n} \bigwedge_{\substack{j\in[1,n]\\ j\neq i}} \bigwedge_{v\in[1,m]} \bigwedge_{\substack{v'\in[1,m]\\ v'\neq v}} \bigwedge_{\substack{\dot{v}\in[1,m]\\ \dot{v}\neq v,v'}} \bigwedge_{d\in D(\dot{v})} (\neg rr_{k,i,j,v,v'} \vee eq_{k,i,j,\dot{v},d}) \tag{4.18}$$

Note here, as the order of i and j is relevant for the relaxed reduction, the equivalence variables $eq_{k,i,j,\dot{v},d}$ require a case distinction, which is left out to shrink the formula size. In the implementation this distinction is considered.

The requirement that for v the random inclusions of i and j are disjoint is implied by the global invariant that no complete scenario is contained twice. In the other case, due to the restriction that for v' a containment holds and for all others the random inclusions are identical, a complete scenario would be included twice. Additionally, the first and second part of the encoding imply that both input scenarios have to be non-empty. Therefore, we shortly recap the definition of an empty scenario. A scenario is either empty, then for all variables the possible delay values are set to false, or for each variable at least one primary delay has to be included. Given the first two clauses defined in 4.17, it is ensured that scenario i is not identical to j, while j has to cover a larger set of values. Thus, considering the fact that j cannot be empty and taking 4.18 into account, we can prove that all input scenarios with a size bigger than two and fulfilling both conditions are non-empty. However for scenarios with exact two variables the condition is not fulfilled, as 4.18 is only constructed for additional variables. Therefore, we require an additional clause stating that at least one value in the contained set is assigned to true, this is the additional clause in 4.17.

Transformation Furthermore, we define the transformation of the participating scenarios in-between steps k and $k+1$. As the relaxed reduction makes use of the fact

that a reducible partner is contained in either i or j, we have to apply the reduction operation to one scenario and remove the used information out of the other. The complete encoding for the relaxed reduction-induced transformation procedure is split into four parts.

The first part is the reduction part, merging the information stored in two random inclusions into one. The merged set is then stored in scenario i for the reduction variable v.

$$\bigwedge_{i=1}^{n} \bigwedge_{\substack{j\in[1,n]\\ j\neq i}} \bigwedge_{v\in[1,m]} \bigwedge_{\substack{v'\in[1,m]\\ v'\neq v}} \bigwedge_{d\in D(v)} (\neg a_{k+1,i,v,d} \vee a_{k,i,v,d} \vee a_{k,j,v,d} \vee \neg rr_{k,i,j,v,v'}) \wedge \tag{4.19}$$

$$(a_{k+1,i,v,d} \vee \neg a_{k,i,v,d} \vee \neg rr_{k,i,j,v,v'}) \wedge (a_{k+1,i,v,d} \vee \neg a_{k,j,v,d} \vee \neg rr_{k,i,j,v,v'})$$
$$(\neg a_{k+1,i,v,d} \vee a_{k,i,v,d} \vee a_{k,j,v,d} \vee \neg rr_{k,i,j,v,v'})$$

The second part ensures that the complete scenario required for the reduction is removed from scenario j.

$$\bigwedge_{i=1}^{n} \bigwedge_{\substack{j\in[1,n]\\ j\neq i}} \bigwedge_{v\in[1,m]} \bigwedge_{\substack{v'\in[1,m]\\ v'\neq v}} \bigwedge_{d\in D(v')} (\neg a_{k+1,j,v',d} \vee a_{k,j,v',d} \vee \neg rr_{k,i,j,v,v'}) \wedge \tag{4.20}$$

$$(\neg a_{k+1,j,v',d} \vee \neg a_{k,i,v',d} \vee \neg rr_{k,i,j,v,v'}) \wedge (a_{k+1,j,v',d} \vee \neg a_{k,j,v',d} \vee a_{k,i,v',d} \vee \neg rr_{k,i,j,v,v'})$$

The third part models that i stays unchanged for all variables not part of the reduction process. Here, a clear distinction is made between i and j as scenario j has to stay unchanged for all variables except those in the relaxation process.

$$\bigwedge_{i=1}^{n} \bigwedge_{\substack{j\in[1,n]\\ j\neq i}} \bigwedge_{v\in[1,m]} \bigwedge_{\substack{v'\in[1,m]\\ v'\neq v}} \bigwedge_{\substack{\dot{v}\in[1,m]\\ \dot{v}\neq v;}} \bigwedge_{d\in D(\dot{v})} (\neg a_{k+1,i,\dot{v},d} \vee a_{k,i,\dot{v},d} \vee \neg rr_{k,i,j,v,v'}) \wedge \tag{4.21}$$

$$(a_{k+1,i,\dot{v},d} \vee \neg a_{k,i,\dot{v},d} \vee \neg rr_{k,i,j,v,v'})$$

Finally, the last part models that all variables except v' stay unchanged for j.

$$\bigwedge_{i=1}^{n} \bigwedge_{\substack{j\in[1,n]\\ j\neq i}} \bigwedge_{v\in[1,m]} \bigwedge_{\substack{v'\in[1,m]\\ v'\neq v}} \bigwedge_{\substack{\dot{v}\in[1,m]\\ \dot{v}\neq v'}} \bigwedge_{d\in D(\dot{v})} (\neg a_{k+1,j,\dot{v},d} \vee a_{k,j,\dot{v},d} \vee \neg rr_{k,i,j,v,v'}) \wedge \tag{4.22}$$

$$(a_{k+1,j,\dot{v},d} \vee \neg a_{k,j,\dot{v},d} \vee \neg rr_{k,i,j,v,v'})$$

In addition, we have to ensure that in each iteration step at most on relaxed reduction is applied on a single scenario i. Beforehand, we define the set of variables for which at most one is allowed to be satisfied at the same state k.

$$RR_{k,i} := \{rr_{k,i,j,v,v'}, rr_{k,j,i,v,v'} |\ \forall j \in [1,...,n], v, v' \in [1,...,m]. \text{ with } v' \neq v \wedge i \neq j\}$$

Given this set, we introduce the at most one relaxed reduction constraint using as above the Bimander encoding.

$$\text{At-most-one}(RR_{k,i}) \tag{4.23}$$

The complete encoding of the relaxed reduction is a large set of clause sets. For completeness and to list the required modifications of the remaining encoding, we formalize it in the following. The set of prerequisites and the transformation are encoded as:

$$reduction_{rel}(k) := (4.17) \wedge (4.18) \wedge (4.20) \wedge (4.19) \wedge (4.20) \wedge (4.21) \wedge (4.22) \wedge (4.23)$$

In case we include the relaxed reduction into the encoding, the remaining parts of the clause set have to be adapted. Analogous to the split operation, we have to increase our variable set by defining variables $rr_{k,i}$ denoting that scenario i is part of a relaxed reduction in iteration step k.

$$\bigwedge_{i=1}^{n}(rr_{k,i} \leftrightarrow (\bigvee_{j=1}^{i-1} \bigvee_{v\in[1,m]} \bigwedge_{\substack{v'\in[1,m]\\v'\neq v}} rr_{k,j,i,v,v'} \vee \bigvee_{j=i+1}^{n} \bigvee_{v\in[1,m]} \bigwedge_{\substack{v'\in[1,m]\\v'\neq v}} rr_{k,i,j,v,v'}))$$

Additionally, we modify the constraint for unchanged scenarios in the transition step $(k, k+1)$. Due to the design of the encoding, we can simply adjust the left side of the disjunction by removing or including the corresponding reduction operations. This ensures that a scenario stays only unchanged in case that it is not part of any reduction operation.

$$\bigwedge_{i=1}^{n}((r_{k,i} \vee sp_{k,i} \vee rr_{k,i}) \vee \bigwedge_{v\in[1,m]} \bigwedge_{d\in\mathcal{D}(p_v)} a_{k+1,i,v,d} \leftrightarrow a_{k,i,v,d}) \tag{4.24}$$

As before, we have to ensure that a scenario can either be reduced, relaxed reduced or split. In any transition only one of these operations for a single scenario i can be executed. To dynamically define and adjust the encoding we included an at most one constraint for the set of allowed reduction techniques, verifying that at most one of these operations is applied to each scenario. We further denote this by At-most-one$_a$, where a is an abbreviation for action.

$$\text{At-most-one}_a(k) := \bigwedge_{i=1}^{n}(\neg r_{k,i} \vee \neg rr_{k,i}) \wedge (\neg r_{k,i} \vee \neg split_{k,i}) \wedge (\neg rr_{k,i} \vee \neg split_{k,i}) \tag{4.25}$$

To modify an encoding to only contain a subset of those operations, we remove those from 4.24 and 4.25. As a final note to this chapter, we introduce a set of operation modes, which are further evaluated in Chapter 5. The modes increase in complexity starting with the basic encoding only including the reduction rule, then we define an encoding with reduction rule and relaxed reduction, one with reduction rule and split operation and one containing all three techniques. The transition step is then

formalized as:

$$
\begin{aligned}
T(k,k+1) := & \bigwedge_{i=1}^{n-1} \bigwedge_{j=i+1}^{n} \bigwedge_{v=1}^{m} \mathit{reducible}(k,i,j,v) \wedge \bigwedge_{i=1}^{n-1} \bigwedge_{j=i+1}^{n} \mathit{reduce}(k,i,j) \wedge \\
& \bigwedge_{i=1}^{n} \mathit{reduce}(k,i) \wedge \bigwedge_{i=1}^{n} \bigwedge_{\substack{j \in [1,n] \\ j \neq i}}^{n} \bigwedge_{v=1}^{m} \mathit{split}(k,i,j,v) \wedge \mathit{reduction}_{rel}(k) \wedge \\
& \bigwedge_{i=1}^{n-1} \bigwedge_{j=i+1}^{n} (\neg r_{k,i,j} \vee (\bigwedge_{v \in [1,m]} \bigwedge_{d \in \mathcal{D}(p_v)} (a_{k+1,i,v,d} \leftrightarrow (a_{k,i,v,d} \vee a_{k,j,v,d})) \\
& \wedge \neg a_{k+1,j,v,d})) \wedge \bigwedge_{i=1}^{n} ((r_{k,i} \vee sp_{k,i} \vee rr_{k,i}) \vee \\
& \bigwedge_{v \in [1,m]} \bigwedge_{d \in \mathcal{D}(p_v)} a_{k+1,i,v,d} \leftrightarrow a_{k,i,v,d}) \wedge \\
& \text{At-most-one}(R_{k,i}) \wedge \text{At-most-one}(SP_{k,i}) \wedge \text{At-most-one}(RR_{k,i}) \wedge \\
& \mathit{Impr}(k) \wedge \mathit{Stop}(k) \wedge \text{At-most-one}_a(k)
\end{aligned}
$$

Finally, we define the complete transition system $[\![M'']\!]_K$, that is simply modifiable by including or removing the named parts.

$$
[\![M'']\!]_K := I_0 \wedge \bigwedge_{k=0}^{K-1} T(k,k+1) \wedge \bigwedge_{i=1}^{n} \mathit{empty}(K,i) \wedge \text{At-most*-}c(E_k),
$$

where $K \in \mathbb{N}$ is the iteration depth.

At-most*-$c(E_k)$ is taken from Formula 4.16 and can be replaced by At-most-$c(E_k)$ in case we disable the improvement clauses (4.13). This is required to correctly specify the iteration step k in which the reduction result is stored, which differs in case we include τ_k.

4.2 BMC

As initially mentioned, the implemented BMC approach does not solely increment the iteration depth K in case the current instance was unsatisfiable. Instead, the deployed goal condition of c empty scenarios is considered as an additional measure to adjust the solution space of the input instance. In the implementation, two main functions are instantiated. On one hand, an approach, which attempts to find the maximal reduction, and on the other hand a method trying to get the best trade-off between reduced instances and running time improvement.

The clear distinction between the approaches has to be stated as this significantly influences the running time of the SAT solver. This is mainly due to the growth of the formula when allowing a higher step range K. The approach computing the minimal output instance begins by determining the worst-case iteration depth required to reduce n instances. If only applying the basic reduction rule, the value can be fixed to $n-1$. However, when the encoding is extended by the relaxed reduction and split method additional steps have to be considered to cover all possible solutions. For the sake of completeness, we shortly mention the required computation steps, although

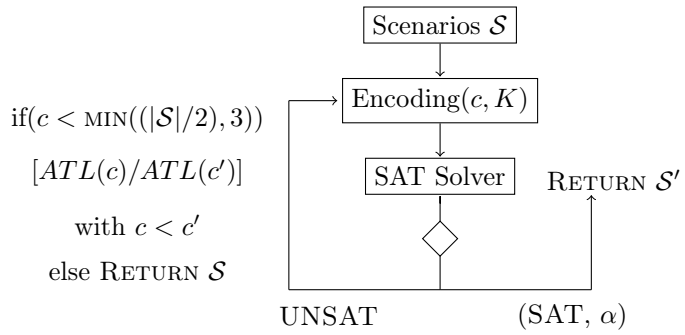


Figure 4.1: BMC procedure.

even an under approximated iteration depth leads to unsolvable instances. Note here, that by unsolvable instances we denote instances not solvable with the given hardware and time. As additionally, in the practical application instances requiring more than two splits occurred only occasionally, a strongly under approximated iteration depth is implemented here.

To compute the worst-case number of required iteration steps, we initially have to determine the number of splits needed to construct the set of all complete scenarios contained in the given scenario set. Thus, we have to multiply the cardinality of all random inclusions for each scenario and the sum of the resulting numbers subtracting the number of given scenarios. Next, knowing how often we have to split in the worst case, the corresponding number of reduction operations has to be calculated. To reduce the input instance the approach has to apply one more reduction than applied splits. It has to be noted here, that multiple reductions can be executed in each time step, the corresponding number of required iterations is thus lower. Finally, the computed numbers can be summed to determine the required iteration range K. Moreover, to allow the split operations, the encoding has to be extended by equivalently many empty scenarios. The defined procedure is thus the complete pre-processing for the bounded model checking approach, which reduces to a minimal output. This approach, however, is not applicable as the instances become to large to be solvable with the given setup.

The second approach also used in the experimental evaluation uses a heuristically defined number of assumed maximal reductions for each input. The amount of for an efficient execution required reductions is solely based on practical experiments. Note here, that in future applications of the reduction method, this heuristic has to be adapted based on the initial delay set $\mathcal{D}$ and the railway systems size. The idea of the approach is to set K to the size of the input scenarios $K := |\mathcal{S}|$, while initially setting the final number of empty scenarios to one and restricting them to the minimum of half of the input size and three $c := \text{MIN}(|\mathcal{S}|/2, 3)$. This constant of three has been computed by evaluating the on average applied reductions. Then during the execution, in case the input instance was unsolvable the number of required empty scenarios is, if possible incremented. This loop is repeated until either an unsatisfiable encoding is found or c cannot be increased without becoming greater than $\text{MIN}(|\mathcal{S}|/2, 3)$. The procedure is additionally depicted in Figure 4.1.

Chapter 5

Experimental Results

5.1 Contextual Setup

In the following chapter, we analyze the performance and especially the improvement induced by the work of this thesis. Due to the reduction of participating instances during the execution we assume to see a significant impact on the runtime of the symbolic simulation.

The reduction techniques are separated into various modes. The first algorithm is solely the basic reduction rule, then the second algorithm is the joint application of reduction and relaxed reduction. Furthermore, algorithms three, four, five, and six are different BMC configurations. The third encodes only the basic case, while four encodes also the relaxed reduction. Therefore, it will be interesting to analyze the direct comparison between the approaches. Configuration five includes the reduction rule and the split operation and method six contains all reduction techniques.

The infrastructure data is parsed from real world data for a railway system and vary in the number of participating trains and the simulated time interval. Moreover, the daytime of the schedule is modified in the example set, which highly influences the density of scheduled trains. Next, to make the simulation more realistic, we vary the set of initial delay values $\mathcal{D}$ and the corresponding probability distribution introduced in the simulation. This is expected to have a substantial effect on the number of train instances and correspondingly on the running time of symbolic simulation.

The compared parameters are the running time of the algorithm as decreasing this is our main motivation, as well as the number of instances after and during the simulation. The first property is a direct measure for the achieved acceleration. Additionally, the number of instances gives insights about potential scalability improvements. To illustrate the results, we show some diagrams about the running time improvement and some depicting the instance count during the execution, to further evaluate the state space growth without and with reduction.

5.1.1 Heuristic Evaluation

As stated in Section 3.2, the deployed heuristic is based on practical results, thus we introduce a short data set showing the its impact. The heuristic builds on the main idea, that before applying the relaxed reduction first all possible applications of the basic reduction rule are executed. Then further, the application order of reductions

	$\|trains\|$	T(in minutes)
rush hour_1	702	60
night_1	140	60
night_5	950	360
morning_3	1152	180
noon_2	762	120

Figure 5.1: Example Instances.

focuses on compressing information by first merging scenarios which do not decrease in size afterwards and only if this is not possible other reduction steps are selected. The idea of this is to prevent scenarios becoming unsuitable by a size decrease and thus impeding further reductions. On the night_1 example the final number of train instances stayed unchanged at 163 instances, however when applying the second reduction mode on rush hour_1 the instance count could be further decreased to 1086 showing a marginal improvement induced by the application order. On all other instances the reduction result was equal or slightly improved in comparison to neglecting the priority ordering considering scenario size and variable ids.

Note here, even though the heuristic enabled additional instance reductions it is mainly based on an intuitive idea to optimally reduce instances and can vary depending on the input size and density of scheduled trains.

5.2 Evaluation

The approach is tested on various inputs, while the characteristics of the used examples are illustrated in Figure 5.1. The relevant parameters are participating trains, time interval, and the number of infrastructure elements of the railway system those are omitted in the table as they are identical for all examples with $|V| = 2646$ and $|E| = 5622$. The examples are denoted as night_1, night_5, rush hour_1, morning_3 and noon_2. The railway systems and their corresponding timetables represent the train network in the north of Germany. Note here, that the train count and time interval T does not directly determine the complexity of an instance, as the number of conflicts and the density of scheduled trains is not part of the listed parameters. However, it is suited to give an indication for very small examples.

In the following we will evaluate the approaches on three different setups, varying the support set of primary delays. Initially, the delay set is set to $\mathcal{D}_1(p_i) = \{0, 1, 2\}$ and the probability distribution is defined as $\mathcal{P}_1(p_i = 0) = 0.2$, $\mathcal{P}_1(p_i = 1) = 0.5$, $\mathcal{P}_1(p_i = 2) = 0.3$ $\forall i \in [1, ..., n]$, where n depicts the number of participating trains. Moreover, the reductions are applied at every 7^{th} time step.

We divide the evaluation of the results into separate plots and a corresponding table to represent the insights to the full extend. Figure 5.4 illustrates on the left side the number of final instances on varying examples and for all different reduction techniques. The conducted example set varies in the density of scheduled trains, correspondingly the percentage and number of reduced instances varies and reaches a maximal reduction rate of 88% for the rush hour_1 input. In the resulting plot of example rush hour_1 and in addition in Table 5.2 and 5.3 it can be seen that the direct application of the reduction and the relaxed reduction are the most efficient

	no r.	r.	r., r.r.	$\text{bmc}_{(r.)}$	$\text{bmc}_{(r.,r.r)}$	$\text{bmc}_{(r.,s.)}$	$\text{bmc}_{(r,r.r.,s.)}$
rush hour_1	29.07	3.45	3.47	6.02	40.3	97.46	-
night_1	0.49	0.40	0.40	0.39	0.41	0.41	0.41
night_5	13.99	9.75	9.67	10.2	12.7	12.84	11.77
morning_3	49.21	37.41	37.51	35.98	35.76	36.29	36.13
noon_2	26.45	22.21	22.01	21.12	21.38	21.61	21.74

Figure 5.2: Running time for $\mathcal{D}_1$ and $\mathcal{P}_1$.

	no r.	r.	r., r.r.	$\text{bmc}_{(r.)}$	$\text{bmc}_{(r.,r.r)}$	$\text{bmc}_{(r.,s.)}$	$\text{bmc}_{(r,r.r.,s.)}$
rush hour_1	3749	1108	1089	1127	1120	1116	-
night_1	277	163	163	163	163	163	163
night_5	2185	1018	1018	1022	1018	1018	1018
morning_3	2139	947	947	947	947	947	947
noon_2	1453	665	665	665	665	665	665

Figure 5.3: Instance count for $\mathcal{D}_1$ and $\mathcal{P}_1$.

techniques. This can be seen in the reduced running time and the number of final instances. However, the BMC approach showed to be only reducing the running time of the simulation in the restricted setting only including the two basic rules. This is illustrated by the violet line in Figure 5.4. The extension considering the split operation theoretically covers the complete reduction potential, however, as the for this step required iteration depth prevents a terminating execution the current implementation cannot keep up with the direct reduction. This can additionally be seen in the running time for rush hour_1 with the extended SAT encoding illustrated in blue and turquoise. Note here, that in Table 5.2, 5.3, 5.5 and 5.6 certain entries are denoted by - as those instances did not terminate in a period of three hours. Not terminating instances did only occur in the application of the BMC approach. Generally, the gathered results showed the significant success of the reduction techniques and an efficient application of the simulation procedure.

In order to additionally analyze how the reduction procedure performs on inputs of an increased size and complexity, we modified the input delay set $\mathcal{D}_2(p_i) = \{0, 1, 2, 3\}$ and the probability distribution is defined as $\mathcal{P}_2(p_i = 0) = 0.2$, $\mathcal{P}_2(p_i = 1) = 0.5$, $\mathcal{P}_2(p_i = 2) = 0.2$, $\mathcal{P}_2(p_i = 3) = 0.1$ $\forall i \in [1, ..., n]$, where n depicts the number of participating trains. Due to the additional delay value the simulation initializes each train at its start element with an extra instance covering the new case. This and the fact that additionally inserted instances induce a notable amount of conflicts and thus splits, results in an increased state space. The remaining parts including the examples stay unchanged for the second experimental setup. As can be seen in Figure 5.7, the reduction approach is at least equally and in some cases even better performing on the extended delay values. The instance count and running time reduction is on average 44.4% and reaches with example rush hour_1 a rate of 99%, which can be seen in Table 5.5 and 5.6 . This significant reduction result is caused by the high train density of the given railway timetable, which leads to an explosive growth of train instances. The increased expansion of train instances slows down the procedure to a final simulation time of nearly two hours. The reduction approach,

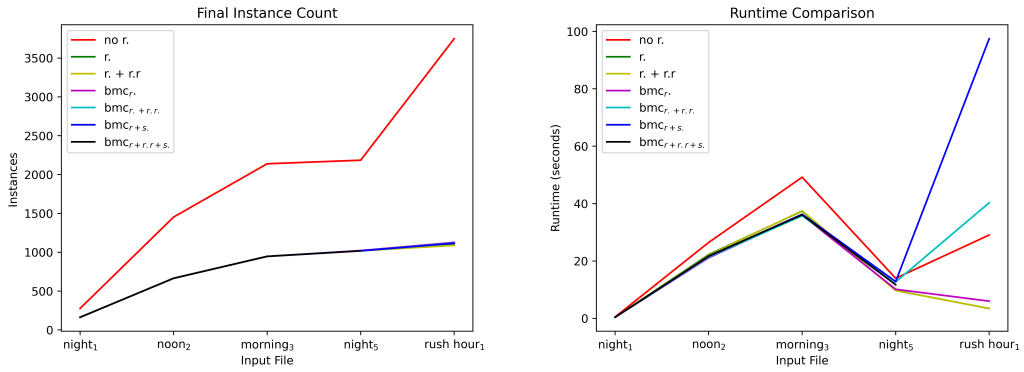


Figure 5.4: Comparison for $\mathcal{D}_1$ and $\mathcal{P}_1$.

	no r.	r.	r., r.r.	$\text{bmc}_{(r.)}$	$\text{bmc}_{(r.,r.r)}$	$\text{bmc}_{(r.,s.)}$	$\text{bmc}_{(r,r.r.,s.)}$
rush hour_1	6861.71	44.47	42.99	588.69	-	-	-
night_1	0.69	0.40	0.40	0.39	0.41	0.41	0.41
night_5	28.47	14.22	13.4	17.65	54.01	171.46	-
morning_3	90.31	63.02	63.11	63.28	63.11	63.46	63.21
noon_2	51.9	38.22	37.58	52.02	38.39	40.55	40.13

Figure 5.5: Running time for $\mathcal{D}_2$ and $\mathcal{P}_2$.

however, decreases explosive growth by directly merging suitable elements and thus improving the efficiency of the algorithm. When applying the reduction method, the running time can be reduced to approximately 44 seconds.

The second analysis demonstrated the relevance and the success of the implemented reduction techniques to even enable the simulation of railway systems previously not terminating in a period of hours. The direct and continuous merging strategy supports the scalability of the algorithm and detects a sufficient amount of instances that can be reduced. Finally, after evaluating the general impact of the reduction techniques, the applied methods have to be compared. The overall best performing implementation is the reduction framework including the relaxed and normal reduction. Due to the extended reduction capability requiring an acceptable complexity, further instances can be reduced leading to the optimal acceleration result. The bounded model checking approach showed to improve the algorithms performance for the basic encoding, while the SAT encoding including the split operation showed to be not suitable for a practical application. This can be mainly explained by the fact that a larger encoding leads to an increased variable and clause set harder to solve in efficient time. Thus, the required solving times exceed the time saved by the reduction. However, in a set of small examples, only the complex encoding was able to compute the minimal output result. Therefore the implementation is theoretically working correctly but for an accelerating effect, further enhancements are required.

In the final part of the chapter, we will directly analyze growth of the state space by illustrating two plots depicting the number of instances during the execution. The

	no r.	r.	r., r.r.	$\text{bmc}_{(r.)}$	$\text{bmc}_{(r.,r.r)}$	$\text{bmc}_{(r.,s.)}$	$\text{bmc}_{(r,r.r.,s.)}$
rush hour_1	31029	2166	1933	2595	-	-	-
night_1	412	222	222	222	222	222	222
night_5	4039	1390	1376	1395	1374	1374	-
morning_3	3099	1165	1165	1165	1165	1165	1165
noon_2	2102	829	823	828	823	823	823

Figure 5.6: Instance count for $\mathcal{D}_2$ and $\mathcal{P}_2$.

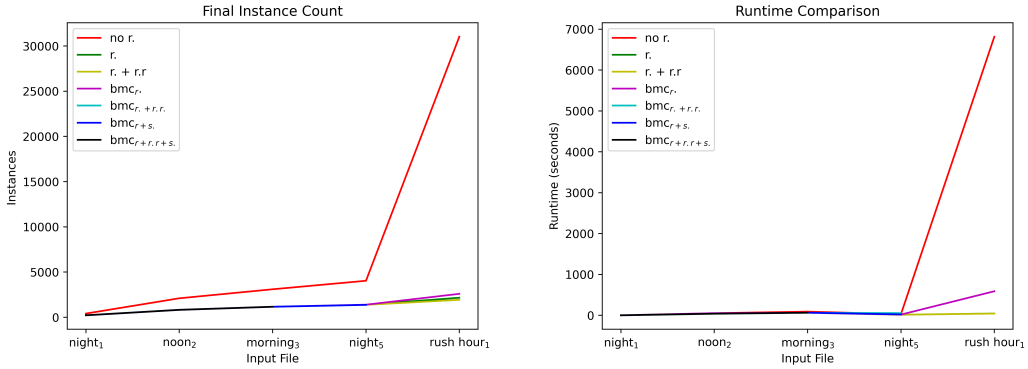


Figure 5.7: Comparison for $\mathcal{D}_2$ and $\mathcal{P}_2$.

selected example for this analysis is rush hour_1 as it showed the most significant running time and instance reduction. In addition to the unreduced simulation we include the approach with the basic reduction rule and the approach with both reduction techniques colored the same as before.

As we can see on the zoomed in plot on the left of Figure 5.8, the reduced application starts at approximately 1800 instances and in the first time steps the count decreases, whereas the unreduced simulation shows direct instance growth. Additionally, at around time step 250, the relaxed rule enabled the reduction of additional instances which is illustrated in the slight lower light green curve. This decrease is propagated throughout the rest of the execution resulting in a lower final count for the second approach and in a marginally improved runtime over the first reduction approach. In the right plot depicted in 5.8 the significant impact of the reduction method is shown. When applying the reduction method the number of instances stays nearly constant over the whole iteration range only growing by around 300 instances, while during the additional application of the relaxed reduction only by around 100 elements. For unreduced symbolic simulation, the red curve depicts an exponential growth of train instances only slowing down at the final iteration steps, as there for most of the trains the schedule is finished. This explosion of the state space can be explained by rising conflicts and splits due to a growing number of instances, which self induces additional splits. As, we implemented a direct reduction of the by the split generated instances, we can directly prevent the large growth induced by them, thus stopping the consequential conflict loop. Due to the implemented re-

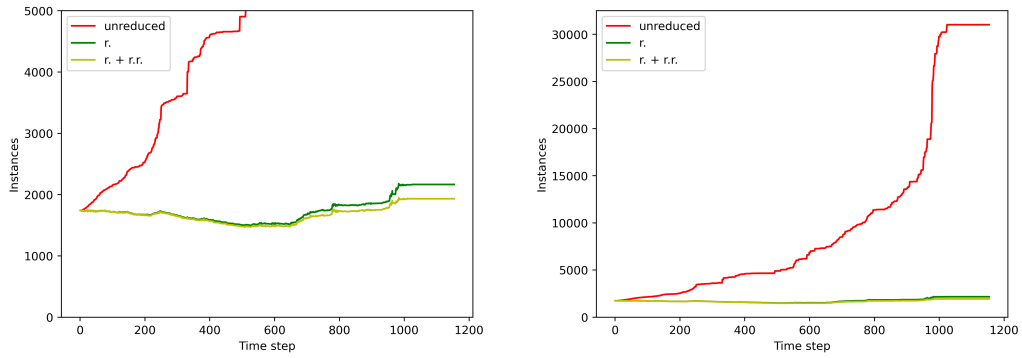


Figure 5.8: Instance growth (left image is zoomed in).

	no r.	r.	r. + r.r.
noon_2 (inst.)	5125	1286	1286
reduct. %		74.91%	74.91%
noon_2 (runt.)	3440.82	125.90	125.2
reduct. %		96.3%	96.4%

Figure 5.9: Noon_2 on $\mathcal{D}_3$.

duction methods instance growth can be decreased on those small examples, hence accelerating symbolic simulation to further analyze the train delay propagation in larger railway systems.

As the current example set only notes such significant reduction rates on the rush hour_1 input, we further modified the delay set to reproduce this result on another time table. Therefore, we choose example noon_2 with the delay set $\mathcal{D}_3(p_i) = \{0, 5, 10\}$. The evaluation results are illustrated in Figure 5.9 and show a significant running time improvement of 96.4% percent. In contrast to extending the delay set, we made it more realistic by increasing the time difference between delays. This lead to more frequent accumulations at train stations increasing the number of computed splits and thus train instances. As in the previous *problematic* instance, the reduction had a substantial impact on the running time of the algorithm by merging suitable train instances, showing a comparable running time reduction. We expect this running time improvement for all inputs at a certain size, thus massively improving the scalability of the algorithm.

5.2.1 Reduction Frequency

To complete the experimental evaluation, we introduce an experimental setup where the impact of varying reductions frequencies is analyzed. As explained in Chapter 3 in Algorithm 4, the reduction is only applied at a certain subset $times_r$ of time steps. Therefore, we choose a set of reduction frequencies, to show the direct impact on the instance evolution.

The evaluation is performed on the rush hour_1 input with $\mathcal{D}_1$ as delay set, the

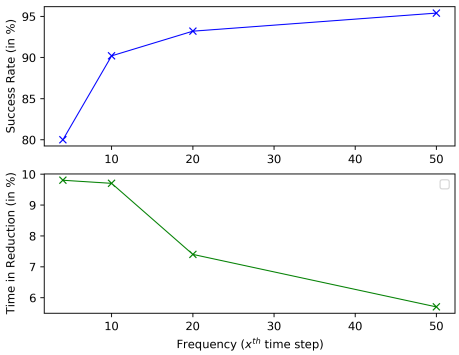


Figure 5.10: Reduction success rate and time percentage spend in reductions (r.).

Time steps	4^{th}	10^{th}	20^{th}	50^{th}
Running time	3.5s	3.5s	3.6s	3.8s

Figure 5.11: Running times for Figure 5.10.

frequencies are set to every $4^{th}, 10^{th}, 20^{th}, 50^{th}$ time step. Before pointing out the findings of the experimental evaluation we present an analysis covering the reduction success percentage and the percentage of time spend in the reduction method. The reduction success rate has been computed by counting the times the reduction method is applied and consequently the times in which the resulting set was strictly smaller. The results are illustrated in 5.10. The shown results have been used to improve the acceleration by enhancing the reduction frequency. On the subplot in the top row the success rate of the applied reductions is depicted and the corresponding x-axis defines the reduction frequency. As can be seen in the plot, the success rate is increasing when lowering the reduction rate as in those cases the input sets are larger and thus include additional reduction potential. Additionally the subplot in the bottom row shows that the time spend in the reduction method is decreasing when reducing the reduction frequency. This is the expected result as in case the frequency is lowered the absolute execution of reductions shrinks, thus the percentage of time spend in it is going down. To correctly evaluate the result, we have to evaluate the corresponding influence on the running time of the algorithm, thus the results are depicted in Table 5.11. The results further show, that the achieved reduction success rate does not mean that the overall approach is working any better, hence even slowing down the simulation by reducing a lower percentage of instances illustrated in 5.14.

Furthermore the instance count over the simulation period is illustrated in 5.14. The example is evaluated with the first and second reduction mode (left, right of 5.14), thus correspondingly the number of instances at the final time step varies in between the pictures. The first interesting insight is that due to a higher frequency than that (7^{th}) in the above complete evaluation of the algorithms in Table 5.2, the final instance count is lower. However, the number of final instances does not directly represent the achieved acceleration. As can be seen in the section above an increased reduction frequency corresponds directly to the percentage of time spent in the reduction.

	no r.	r.	r., r.r.	$\text{bmc}_{(r.)}$	$\text{bmc}_{(r.,r.r)}$	$\text{bmc}_{(r.,s.)}$	$\text{bmc}_{(r,r.r.,s.)}$
night_5	12.99	9.82	9.71	9.84	13.02	18.04	23.38

Figure 5.12: Running time for night_5 on $\mathcal{D}_1$ with frequency 4^{th}.

	no r.	r.	r., r.r.	$\text{bmc}_{(r.)}$	$\text{bmc}_{(r.,r.r)}$	$\text{bmc}_{(r.,s.)}$	$\text{bmc}_{(r,r.r.,s.)}$
night_5	2185	1018	1018	1020	1016	1016	1015

Figure 5.13: Instance count for night_5 on $\mathcal{D}_1$ with frequency 4^{th}.

Additionally Figure 5.14 illustrates the difference between varying reduction rates, which is mainly visualized by higher spikes in the state space induced by an increased number of time steps until the next reduction operation. Correspondingly the lower frequencies induce an increased instance count at the final time step. For the given example set (night_1, noon_2, morning_3, night_5, rush hour_1) the frequency of every 7^{th} time step showed to result in the best trade-off between time spent in reduction method and time saved by those. Important to note here, is that the frequency depends on many factors and has been optimized in the context of this thesis to suite an optimal result of the reduction framework. Despite that, especially when considering the BMC approach a lower frequency showed to be more efficient as the time spend in the solving process is comparably large and can thus be minimized. We shortly illustrate this on night_5 with $\mathcal{D}_1$ by applying the reduction at every 4^{th} time step in comparison to at every 7^{th} as used above, the results are depicted in Table 5.12 and 5.13. The results show basically two important insights. First, the running time when applying the extended SAT configurations doubled in comparison to the reduction at every 7^{th} time step, whereas the running time of the basic reductions is nearly the same. This shows that the impact of the frequency rate highly varies between the different techniques. Furthermore, the example also demonstrates the increased reduction potential of the BMC approach by reducing the number of final instances by one additional train instance in the mode including all reduction operations. However, as already expected the time required to find this additional reduction instance is not covering the time saved by it. Though the earlier mentioned optimal trade-off suiting the thesis purpose is found in the reduction framework including the basic and relaxed reduction rule.

Additionally, the heuristic is dependent on the structure and density of the railway timetable, thus there is still research that can be further evaluated. An improvement of an optimal reduction frequency could additionally boost the performance of the implementation. Due to time constraints this has not been further investigated and only optimized using the obtained analysis results. A further analysis is a possible future work topic.

Note here that the reduction success rate was on average around 85%, which on one side shows that the set of train instances contains a large set of reducible scenarios and on the other one induces that the implemented methods have been a substantial success in reducing instances.

In addition to the reduction frequency we evaluated the impact of modifying the lines at which reductions are applied. As explained in Section 3.2 the reduction method is called in $\textsc{Update}(t, x, r)$ to reduce instances at each infrastructure element

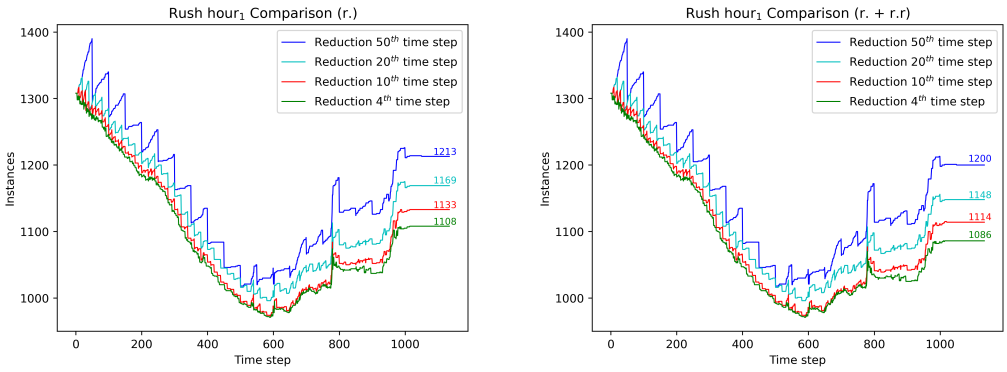


Figure 5.14: Instance growth varying the reduction frequency.

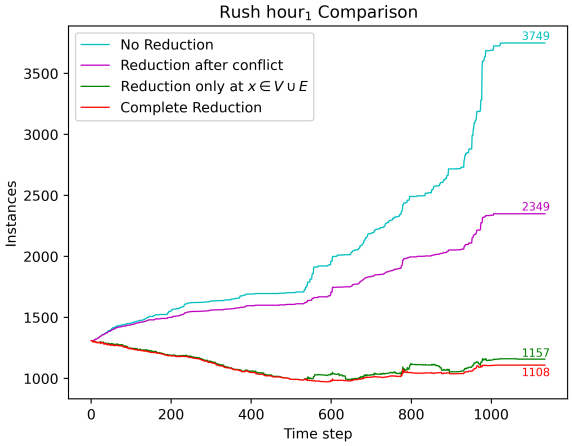


Figure 5.15: Compare instance growths when varying reduction application.

$x \in V \cup E$ and after a conflict in AVAILABLE(x, r). In the following we modify this to influence the corresponding impact. Therefore we selected four different modes, the unreduced application illustrated in light blue, the application of the reduction solely after a conflict in magenta, the reduction at every fourth time step in UPDATE(t, x, r) in green and the reduction framework applied in the update and available method. The corresponding examples are depicted in Figure 5.15.

As expected, the sole application of the reduction method in UPDATE(t, x, r) performed significantly well, as the additional instances of unreduced scenarios not caught by the execution in AVAILABLE(x, r) are at the latest reduced when the next reduction time step is reached. As the reduction is called at every fourth time step, this has no large impact. Moreover, the curve colored in magenta shows the reduction application only considering instances constructed during a conflict. Therefore, it cannot reduce instances never part of a conflict, however, the approach still reduced the number of instances by 63%. Especially in the later time steps and as already

seen in the large example above there is a time step at which the number of instances explodes at around 900 and here we can see a significant difference between the light blue and magenta curve as the reason for the growth is probably a station causing a large number of splits. Due to the reduction after conflicts, the second approach slows down growth of the state space and thus the final instance count. Consequently, also the running time varies, the running time of the unreduced case was 34.2s, for the application after conflicts 9.92s, for the reduction solely in $\textsc{Update}(t, x, r)$ the running time was 3.98s and the complete approach had a running time of 3.42s.

This and the difference between the green and red curve underlines the importance of the additional application of the reduction method in $\textsc{Available}(x, r)$ to further improve the reduction potential.

Chapter 6

Conclusion

6.1 Summary

In the following we recap the insights of this thesis with a section about promising future research topics. The work had a significant impact on the performance of the algorithm and enables an analysis of more realistic inputs. Through the continuous application of reduction techniques during the symbolic simulation, the number of generated instances has been reduced substantially. This has lead to a decreased growth of the simulations state space.

Throughout this thesis, the initially defined theory for reductions lead to intuitive and rather complex implementations. This was extended to implementing reduction methods covering the complete reduction potential. The reduction framework induced fast and promising results while causing various issues with endless loops in the context of undoing previous merging steps. The presented heuristics order the elements of the priority queue with an advanced loop detection to efficiently apply the basic reduction rule together with the relaxed reduction. The first reduction rule is extended by a relaxed version accessing even more reducible scenario tuples. This allowed to boost the performance of the algorithm by catching additional merging steps. However the first approach did not exploit the complete potential of possible reductions, thus it is extended by a more strategic approach.

The bounded model checking approach allows to effectively evaluate all possible reduction combinations until determining the optimal result. To cover the complete reduction potential the worst case number of required transaction steps has to be computed, resulting in a high iteration depth for the transition system. The high iteration depth produced larger instances which were hardly solvable and did slow down the simulation instead of accelerating it. From the theoretical perspective the method can compute the optimal reduction result. However, the extended variant covering the complete potential showed to be not practical for most of the input examples. Nevertheless, it gave us an important insight about the reduction efficiency.

The implemented framework does not always find the optimal reduction sequence, still it finds enough reductions, resulting in final instances numbers sufficiently close to the optimum. As the trade-off between finding those last additional reduction instances is not covering the required time to find those, the complete coverage of the reduction potential does not support the goal of this thesis. To detect the reduction steps needed to compute the minimal output the algorithm requires more solving time

than for the easily found merging operations. Therefore, the reduction framework with its effective and fast priority queue demonstrated to have to optimal trade-off between reductions and required time for those.

Overall the thesis covers every aspect of a theoretical reduction idea, in the context of merging suitable instances without losing any stochastic information.

As a general remark of the work, we state the significant reduction success, allowing to solve input instances previously unsolvable in a period of hours, while slowing down growth of the state space to improve the scalability of the simulation. However, there exist still problematic instances, that are even hard to solve with the application of the reduction technique. These are especially instances in which the delay values are wider separated and increased, as thus instances generate many conflicts and have a high growth rate even between single time steps. This corresponds to an increased reduction application as after each conflict a direct reduction is applied, resulting in an additional time duration spend in the defined procedure. The realistic occurrence of such large input delay sets is not considered any further and has to be evaluated in future work.

6.2 Future Work

In the process of this thesis various potentially promising approaches were mentioned but not yet further researched. We shortly recap topics and ideas suitable for a deeper investigation. Especially, during the analysis of implemented approaches, the time spent in different parts of the algorithm (e.g. generation of the encoding, solving of the encoding) has shown significant potential to be further enhanced. As currently the generation of the SAT encoding occupies a large proportion of the reduction time this could be further improved in future adoptions.

Furthermore, the encoding can be enhanced to reduce the number of required clauses and variables by reformulating the used at-most and at-least constraints. As the smallest encoding of the BMC approach only including the basic reduction rule was able to compete with the reduction framework, we assume that additional changes and improvements in the efficiency can lead to comparable results regarding running times using the complete reduction potential by including all reduction techniques.

Moreover, a promising approach to improve the running time of the symbolic simulation is to apply a CEGAR-based approximation technique. In the simulation train instances are finely split into sub scenarios of negligible probability. Thus, deploying a strict under or over-approximation could help to further reduce the set of instances and hence improve the performance of the algorithm while losing an acceptable subset of information. However, the approximation state has to be verified to not violate a predefined threshold to ensure realistic results. The requirements for this research field are implemented as the probabilities for scenarios are continuously computed and verified to satisfy the global invariant.

Bibliography

[CBRZ01] Edmund Clarke, Armin Biere, Richard Raimi, and Yunshan Zhu. Bounded model checking using satisfiability solving. *Formal Methods in System Design*, 19:7–34, 01 2001.

[FG13] Bert Fristedt and Lawrence Gray. *A Modern Approach to Probability Theory*. Probability and Its Applications. Birkhäuser Boston, 2013.

[HÁN21] Rebecca Haehn, Erika Ábrahám, and Nils Nießen. Symbolic simulation of railway timetables under consideration of stochastic dependencies. In *Quantitative Evaluation of Systems*, pages 257–275, Cham, 2021. Springer International Publishing.

[JNRSfTL06] Yuan Jianxin, Infrastructure Netherlands Research School for Transport, and Logistics. *Stochastic Modelling of Train Delays and Delay Propagation in Stations*. TRAIL thesis series. Netherlands TRAIL Research School, 2006.

[NM15] Van-Hau Nguyen and Son T. Mai. A new method to encode the at-most-one constraint into SAT. In *Proceedings of the Sixth International Symposium on Information and Communication Technology*, SoICT 2015, page 46–53, New York, NY, USA, 2015. Association for Computing Machinery.

[Sin05] Carsten Sinz. Towards an optimal CNF encoding of boolean cardinality constraints. In Peter van Beek, editor, *Principles and Practice of Constraint Programming - CP 2005*, pages 827–831, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.