Otto-von-Guericke-Universität Magdeburg

Fakultät für Informatik

Bachelorarbeit

Modellbasierte Sicherheitsanalyse eines Bahnhofsstellwerkes

Autor:

Ludwig Bedau

31. März 2017

Betreuer:

Prof. Frank Ortmeier

Institut für Intelligente Kooperierende Systeme
Lehrstuhl für Softwaretechnik

Bedau, Ludwig:
Modellbasierte Sicherheitsanalyse eines Bahnhofsstellwerkes
Bachelorarbeit, Otto-von-Guericke-Universität Magdeburg, 2017.

Inhaltsangabe

Stellwerke stellen eine zentrale Komponente für den sicheren Schienenverkehr dar. Um die korrekte Funktionsweise sicherzustellen und sicherheitskritische Zustände auszuschließen, sind formale Methoden angebracht.
In dieser Arbeit soll gezeigt werden, wie in der Modellierungssprache SAML (System Analaysis and Modeling Language) ein Baukastensystem erstellt wird, mit dem sich beliebige Stellwerke modellieren lassen. In Rahmen der Arbeit wird beschrieben, wie das reale System abstrahiert wurde, um das zu untersuchende Verhalten herauszustellen. Die Arbeit beschreibt weiterhin die Sicherheits- und Liveness-Spezifikationen, die zur Verifikation der generierten Bahnhofsstellwerke genutz werden. Im Anschluss wird an einem Beispiel die Modellierung eines Bahnhofsstellwerkes auf Basis des vorgestellten Baukastensystems gezeigt. Bei der Verifikation wurde eine kleine Auswahl an Model-Checking-Tools genutzt. Ein Vergleich dieser Tools in Bezug auf das hier gestellte Problem sowie eine Auswertung der Erfahrungen, die während des Projektes gesammelt werden konnten, sollen den Abschluss bilden.

Abstract

Interlocking systems are a meaningful component for the safe operation of a railway system. To be sure about the correct behavior of interlocking systems and to exclude saftycritical states, the use of formal methodes is appropriate.
In this thesis will be shown how to design a modular construction system using SAML (System Analaysis and Modeling Language), so that you can model any arbitrary interlocking system with it. There for we will show, how to abstract the real system to underline the behavior which should be examine. There will be described the safety- as well as the liveness-specifications, which will be prooved with a selection of model checkers. Following this, there will be shown the implementation of an example railway yard based on the presented modular system. A comparison between the results of the used model checkers in relation to this problem and the evaluation of the experience, made with this project, will finalize this thesis.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quelltextverzeichnis

1. Einführung

Abgesehen von Strecken mit Einzugbetrieb sind Stellwerke für einen zuverlässigen und vor allem sicheren Zugbetrieb unerlässlich. Da es bei Fehlverhalten zu Entgleisungen sowie Kollisionen zwischen Zügen und damit zu hohem Personen- und Sachschaden kommen kann, sind zur Überprüfung der korrekten Funktionsweise formale Methoden angebracht.
Der derzeitige Stand der Technik weicht davon jedoch deutlich ab. Die Überprüfung von Stellwerkstechnik erfolgt in langwieriger und fehleranfälliger Handarbeit durch dazu staatlich bestellte Gutachter. Eine frühzeitige Fehlererkennung, die Zeit und Kosten sparen könnte, ist so praktisch unmöglich.
Formale Methoden können eine Lösung für dieses Problem sein. Deren Einsatz erfordert jedoch ein hohes fachliches Wissen aus den beiden Fachgebieten Schienenverkehrstechnik und formale Methoden. Da für die Anwender die größte Hürde im Bereich der formalen Methoden liegt, bietet es sich an hier unterstützend anzusetzen.

1.1 Motivation

Die Entwicklung von Stellwerkslogiken ist nach wie vor durch viel Handarbeit geprägt. Fehler sind dabei nahezu unvermeidbar, stellen aber auch kein Problem dar, wenn sie frühzeitig und zuverlässig gefunden werden können. Der aktuelle Stand der Technik sieht dazu eine Simulation von ausgewählten Betriebssituationen vor. Auf Basis dieser Tests wird die Zulassung durch einen dafür bestellten Gutachter gegeben. Bei diesem Verfahren werden jedoch nicht alle Situationen abgedeckt. Dadurch ist es möglich, dass Fehler unerkannt bleiben und im realen Stellwerk zu sicherheitskritischen Situationen führen. Dies kann zu schwerwiegenden Unfällen mit hohem Sach- und Personenschaden führen.
Im Zuge des Streckenausbaus und der Ausrüstung mit moderner Signal- und Sicherungstechnik werden zur Personaleinsparung immer mehr Stellwerke in großen computergestützten Betriebszentralen zusammengefasst. Diese elektronischen Stellwerke sichern den Eisenbahnverkehr mehrerer Regionen und Ballungszentren ab. Bei der manuellen Überprüfung der dafür nötigen Software ist es schwer, den Überblick über das System zu behalten.

Bei der Betrachtung der Folgen von Fehlfunktionen zeigt sich also, dass Tests in Simulationsumgebungen nicht die benötigte Gewissheit über die korrekte Funktion bieten können. Darüber hinaus werden mögliche Fehler erst bei der Zertifizierung gefunden, was zu sehr hohen Kosten sowie zu längeren Bearbeitungszeiten führt.
Die Einführung formaler Methoden ist in diesem Bereich seit Längerem angestrebt, scheiterte jedoch oft an Rechen- und Speicheraufwand sowie an dem nötigen Fachwissen im Bereich der formalen Methoden auf der Seite der Anwender. Das Hauptaugenmerk der Forschung lag bis jetzt auf der Verbesserung der Rechenzeit und des Speicherverbrauches. Durch Optimierungen an verschiedensten Stellen konnten hier Verbesserungen erzielt werden. Dabei wurden jedoch häufig Kenntnisse über spezielle örtliche Situationen genutzt, um bestimmte Einschränkungen annehmen zu können. Damit ist eine generische Erstellung von Stellwerksmodellen nicht möglich. Zudem benötigt die Modellierung und Verifikation Personal mit Kenntnissen aus dem Bereich der formalen Methoden sowie aus dem Bereich der Schienenverkehrstechnik. Darüber hinaus muss auch der Gutachter der Zertifizierungsstelle Kenntnisse im Bereich der formalen Methoden besitzen, um die Umsetzung des realen Problems in ein Modell nachvollziehen zu können.
Um die Verifikation mit formalen Methoden im Bereich der Stellwerkslogik zu etablieren, muss die Hürde der Fachkenntnisse über die Schienenverkehrstechnik hinaus abgeschwächt werden. Ideal ist eine verständliche, einmalig zu zertifizierende Methode, die automatisiert die Stellwerkslogik sowie die örtlichen Bedingungen und Gleisgeometrien in ein Modell überführt und anhand von generischen Sicherheitskriterien verifiziert.

1.2 Zielstellung der Arbeit

Ziel der Arbeit ist es, ein Pattern zur Modellierung von Stellwerken in der Modellierungssprache SAML (System Analysis and Modeling Language) zu beschreiben und geeignete Verfahren zur Verifikation der daraus abgeleiteten Modelle zu finden. Dazu sollen an einem beispielhaften Stellwerk generalisierbare Komponenten ausfindig gemacht werden und in einer Template-Struktur umgesetzt werden.
Damit soll es möglich sein, Modelle von beliebigen Stellwerken (teil)-automatisiert zu erzeugen. Ebenfalls sollen generische Sicherheitsspezifikationen (teil)-automatisiert an das spezielle Stellwerk angepasst werden können. Damit wird das Ziel verfolgt, die Hemmschwelle für den Einsatz formaler Methoden zu reduzieren. Denn mit der Anwendung sind zumeisten Personen betraut, die zwar Fachwissen auf dem Gebiet der Schienenverkehrstechnik, nicht aber im Bereich der formalen Methoden haben.
SAML eignet sich dafür besonders gut, da sie über die Möglichkeit verfügt, mit Templates das Verhalten bestimmter Komponentenklassen generisch festzulegen. Dadurch wird die Basis für eine (teil-)automatisierte Generierung von einer Vielzahl verschiedener Bahnhofsstellwerke mit den unterschiedlichsten Topologien geschaffen.
Dabei bietet die Model-Checking-IDE VECS (Verification Enviroment for Critical Systems[1]) eine sehr gute Hilfestellung bei der Modellierung und Verifikation solcher komplexen Systeme.

[1] https://cse.cs.ovgu.de/vecs/

1.3 Gliederung der Arbeit

Grundlegende Informationen zur Funktionsweise und Entwicklung der Stellwerkstechnik sowie zur verwendeten Modellierungsumgebung, der Modellierungssprache und den Verifikations-Algorithmen werden in Kapitel 2 beschrieben. In Kapitel 3 folgt die Beschreibung des Modellierungsansatzes mit den Basiskomponenten des Stellwerkes. Eine genaue Beschreibung der Verifikationsbedingungen und Spezifikationen findet sich in Kapitel 4. Aus diesem Baukastensystem wird in Kapitel 5 das Bahnhofsstellwerk von Braine l'Alleud exemplarisch modelliert. Mit diesem Modell wurden einige Versuche durchgeführt. Die Beschreibung und die Ergebnisse finden sich, wie die Schlussfolgerungen ebendort. Ein Ausblick auf andere Arbeiten zu diesem Thema wird in Kapitel 6 gegeben. Ein zusammenfassung mit Ausblicken auf weiterführende Themen bietet Kapitel 7.

2. Grundlagen

2.1 Grundlagen der Stellwerkstechnik

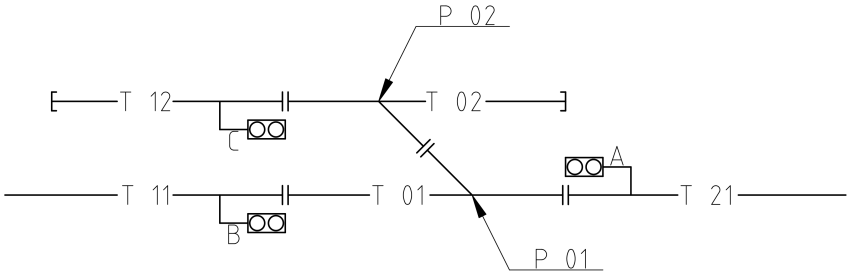


Abbildung 2.1: Fiktiver Gleisplan eines exemplarischen Bahnhofes

Das gesamte Schienennetz ist in eine Vielzahl von Bereichen eingeteilt. Jeder dieser Bereich wird durch ein Stellwerk kontrolliert. In Abbildung 2.1 ist ein fiktiver Gleisplan eines solchen (sehr kleinen) Bereiches dargestellt. Zu sehen sind dort die untereinander verbundenen **Gleisabschnitte** (Tracks) (T 01, T 02, T 11, T 12 und T 21), auf denen die Züge verkehren. Dabei ist z. B. T 12 eine Sackgasse (Stumpfgleis) und endet nach links an einem Prellbock. Über die Gleise T 11 und T 21 können Züge in die Wirkungsbereiche anderer Stellwerke hineinfahren. Der Gleisabschnitt T 01 besitzt die Weiche P 01 und der Gleisabschnitt T 02 besitzt die Weiche P 02, dort ist es möglich, Züge in verschiedene Richtungen weiter fahren zu lassen. Die **Weichen** (Switches oder auch Points) P 01 und P 02 können sich in normaler Lage befinden, dann verbinden sie die Gleise T 11 und T 21, außerdem wird T 12 von der durchgehenden Strecke (T 11 $\leftrightarrow$ T 01 $\leftrightarrow$ T 21) getrennt. Stehen sie in abzweigender Lage, verbinden sie T 12 und T 21 miteinander. Wenn sich die Weichen unbeabsichtigt verstellen, kann dies zu Entgleisungen von Zügen führen. Desswegen müssen sie mechanisch in ihrer Lage verriegelt werden, bevor sie durch einen Zug befahren werden können. Um den Zügen zu signalisieren, ob eine sichere Weiterfahrt möglich ist, gibt es **Signale** (A, B und C). Sie dürfen nur auf freie Fahrt (grün, auch „Hp1“ genannt) stehen, wenn die darauffolgenden Gleise sicher

befahren werden können. Ansonsten zeigen sie Halt (rot, auch „Hp0“ genannt). Um kollisionsfreie Fahrbewegungen von Zügen garantieren zu können, werden auf diesem Gleisplan Fahrstraßen definiert.

Fahrstraße
In den Weichenbereichen der Fahrstraßenknoten (Bahnhöfe, Abzweig- und Überleitstellen) verkehren Züge [. . .] grundsätzlich auf technisch gesicherten Fahrwegen, den so genannten Fahrstraßen. Die Sicherung einer Fahrstraße für Züge muss folgenden Sicherheitsanforderungen genügen:

- *Sicherstellung der richtigen Lage aller beweglichen Fahrwegelemente vor Zulassung einer Zugfahrt*
- *Verhinderung des Umstellens von beweglichen Fahrwegelementen im freigegebenen Fahrweg eines Zuges*
- *Verhinderung der Zulassung gefährdender (so genannter „feindlicher“) Fahrten*
- *Verhinderung, dass Fahrzeuge in den freigegebenen Fahrweg eines Zuges gelangen können*
- *Verhinderung der Zulassung von Zugfahrten in besetzte Gleisabschnitte.*

aus [Pac11]

Eine Fahrstraße beginnt dabei an einem Signal und führt auf ein Gleis. In der Beispiel-Station gibt es 4 mögliche Fahrstraßen:

- von A über T 01 nach T 11, wobei P 01 und P 02 in normaler Lage sein müssen
- von A über T 01 und T 02 nach T 12, wobei P 01 und P 02 in abzweigender Lage sein müssen
- von B über T 01 nach T 21, wobei P 01 und P 02 in normaler Lage sein müssen
- von C über T 02 und T 01 nach T 21, wobei P 01 und P 02 in abzweigender Lage sein müssen

Eine Fahrstraße zu stellen darf nur möglich sein, wenn sich die Weichen, die befahren werden sollen, in der korrekten Lage befinden und die zu befahrenden Gleise frei von anderen Zügen sind, sowie nicht durch andere Fahrstraßen beansprucht werden.
Der Sinn der Weiche P 02 ergibt sich aus einem weiteren Aspekt. Eine typische Anwendung des Gleises T 12 ist ein Abstellgleis, wo Wagen z. B. zum Be- und Entladen abgestellt werden. Um zu verhindern, dass es durch einen unbeabsichtigt ins Rollen geratenen Wagen zu einem Unfall auf der Weiche P 01 kommt, muss für die Fahrstraßen B→T 21 und A→T 11 nicht nur die Weiche P 01, sondern auch die Weiche P 02 in normale Lage gebracht werden. Ein rollender Wagen fährt so nicht in die Flanke eines Zuges, sondern gegen den Prellbock am geraden Ausgang von P 02. Diese Sicherheitsvorkehrung ist als Flankenschutz bekannt und je nach Situation durch Regelwerke, wie die Eisenbahn Bau- und Betriebsordnung, vorgeschrieben.

Nicht immer müssen dazu jedoch extra Weichen, wie in diesem Beispiel, eingebaut werden.
Um die Menge an Fahrstraßen, Gleisabschnitten, Weichen und Signalen zu koordinieren, gibt es verschiedene Stellwerkskonzepte. Die Ersten zeichneten sich durch ein sehr restriktives Verhalten aus. Fahrstraßen konnten nur vollständig gestellt werden und auch erst aufgelöst werden, wenn sie komplett passiert waren. Um den Durchsatz der Bahnhöfe und damit die Leistungsfähigkeit der Eisenbahn zu erhöhen, wurden Konzepte entwickelt, die es möglich machen, Fahrstraßen schrittweise hinter dem Zug aufzulösen und bereits freigefahrene Abschnitte für andere Fahrstraßen zu nutzen. Dieser technische Zustand, in dem sich das Stellwerk der anschießenden Fallstudie (Braine l'Alleud) befindet, soll modelliert werden. Zielstellung für die nächsten Jahre wird der Übergang vom fixen Blockabstand hin zu wandernden Blockabständen. Dies bietet die Möglichkeit, Fahrstraßen ungeachtet von Signalen bis an den vorausfahrenden Zug heran zu stellen. Diese Entwicklung soll in dieser Arbeit nicht bedacht werden. Weiterführende Erklärungen zu Sicherungstechniken im Eisenbahnwesen finden sich in [Pac11].
Um das Modell eines Bahnhofsstellwerkes auf sicherheitskritische Fehler zu überprüfen, müssen Sicherheitskriterien definiert werden. Deren Einhaltung wird während der Verifikation geprüft. Im Rahmen dieser Arbeit sollen die folgenden Sicherheitskriterien verifiziert werden:

Kollisionsfreiheit

Kollisionsfreiheit bedeutet in unserem Modell, dass sich zwei Züge nie gleichzeitig auf dem selben Gleis befinden. Aus diesem Grund werden in dieser Arbeit Rangierfahrten vernachlässigt, denn zur Zugbildung muss das Befahren von belegten Gleisen möglich sein.

korrekte Weichenlage

Um Entgleisungen zu verhindern, müssen alle Weichen, wenn sie von Zügen befahren werden, sich in der korrekten Lage befinden und gegen Verstellen gesichert sein. Es darf nicht vorkommen, dass Weichen aufgeschnitten werden (stumpfes Befahren von falsch gestellten Weichen) und dass Weichen unter einem Zug verstellt werden.

Flankenschutz

> **Flankenschutz**
> *Flankenschutzmaßnahmen sollen verhindern, dass ein Zug durch in seinen Fahrweg einmündende Fahrten (so genannte Flankenfahrten) gefährdet wird.*
>
> aus [Pac11]

Insbesondere auf Hauptgleisen ist es also wichtig, fahrende Züge vor durchrutschenden Zügen oder rollenden Wagen zu sichern. In dieser Arbeit soll sich auf den Flankenschutz durch Schutzweichen beschränkt werden. Eine Schutzweiche ist eine Weiche, die nicht Teil des zu stellenden Fahrweges ist, jedoch in eine abweisende Lage gebracht wird, um Flankenschutz zu bieten.

2.2 Modellierung mit SAML in VECS

SAML ist eine transitionsregelbasierte Modellierungssprache. Da sie als Beschreibungssprache für Kripke-Strukturen verstanden werden kann, ist eine eindeutige Übersetzung in Eingabeformate für diverse Model-Checker möglich. Die im Rahmen dieser Arbeit verwendete Übersetzung von SAML nach SMV ist darüber hinaus mathematisch bewiesen korrekt. SAML erlaubt es komplexe Systeme als eine Menge von parallelen Automaten zu modellieren, die zeitdiskret ausgeführt werden. Dazu werden für jeden Automaten die Übergangsrelationen angegeben. In dieser Arbeit soll ein kleiner Einblick in die Syntax von SAML genügen. Weitere Informationen dazu finden sich in [GO10] und [Güd11] sowie in der Onlinedokumentation[1].
VECS ist ein Tool, dass Modellierung in SAML mit IDE-Unterstützung, Model Checking (unter Anderem NuSMV bzw. NuXMV) und Simulationsmöglichkeiten verbindet. Die Entwicklung findet am Lehrstuhl für Software Engineering an der Otto-von-Guericke Universität Magdeburg statt. Idee hinter dem System ist es, eine einfache, leicht verständliche, jedoch auch mächtige Möglichkeit zur Modellierung und Verifikation zu bieten, die ohne tiefgreifende Kenntnisse aus dem Bereich der formalen Methoden bedienbar ist.

2.2.1 SAML-Syntax

Im Folgenden sollen die in dieser Arbeit verwendete Elemente der SAML-Syntax kurz vorgestellt werden:

Zustandsvariable

In SAML lassen sich Systeme durch parallele Automaten modellieren. Ein Automat wird dabei durch eine Zustandsvariable abgebildet. Der Zustand des Systems beschreibt sich durch die Zustände aller Variablen des Modells. Diesen Zustandsvariablen können dabei ganze Zahlen oder Enumerationen zugewiesen werden. Jede Zustandsvariable hat einen Startzustand (Initialzustand). Der Startzustand des Systems ergibt sich demzufolge aus den Startzuständen aller Variablen.

```
1 <variablename1> : [<range from>..<to>] init <number in range>;
2 <variablename2> : <enumname> init <enumname>.<statename>;
```

Quelltext 2.1: SAML-Syntax Zustandsvariable

In Quelltext 2.1 sind beide möglichen Varianten der Variableninitialisierung angegeben. Bei der Angabe von ganzzahligen Bereichen ist darauf zu achten, dass ein Bereich inklusive der unteren Grenze und exklusive der oberen Grenze generiert wird. Für das hier beschriebene Modell sollen jedoch nur Enumerationen verwendet werden, da sie eine verständlichere Zustandsbezeichnung bieten.

Enumeration

Um eindeutigere Bezeichnungen von Zuständen eines Systems zu haben, können in SAML anstelle von Zahlen Zustandsnamen definiert werden. Dies lässt sich über Enumerationen erzielen.
Eine Aufzählung beschreibt dabei den Zustandsraum einer Variablenklasse, also eines Automatentypen.

[1] https://cse.cs.ovgu.de/saml-sde/wiki/documentation

```
enum <enumname>  := [<value1>, <value2>, ...];
```

Quelltext 2.2: SAML-Syntax Enumeration

Zuweisung

Die Zuweisungen (Assignments) sind Regeln, die für jeden Zustand einen Folgezustand angeben, also abhängig von der aktuellen Variablenbelegung allen Variablen neue Werte zuweisen.

```
<condition> -> <variable1>' = <new value 1>
               & <variable2>' = <new value 2>
               & ... ;
```

Quelltext 2.3: SAML-Syntax Zuweisung

Nicht immer sind die Übergänge deterministisch. Ein Zug kann z. B. stehen bleiben oder seine Fahrt fortsetzen. Beide Optionen müssen in dem Modell bedacht werden. Dazu können in Zuweisungen optionale Zustandsübergänge angegeben werden. Sie werden mit dem Schlüsselwort **choice** angegeben. Damit lassen sich nichtdeterministische Übergänge mit beliebig vielen Zuweisungsblöcken angeben.

```
<condition> -> choice (<variable1>' = <new value 1>
                       & <variable2>' = <new value 2>
                       & ... )
             + choice (<variable1>' = <new value 1>
                       & <variable2>' = <new value 2>
                       & ... );
```

Quelltext 2.4: SAML-Syntax nichtdeterministische Zuweisung

Die Bedingung (Condition) jeder Zuweisung ist ein aussagenlogischer Ausdruck. Wichtig zu beachten ist dabei, dass alle Bedingungen eindeutig und vollständig sind, denn der Automat muss in jedem Zustand einen Folgezustand haben. Dieser Folgezustand muss aber, abgesehen von den angegebenen Indeterminismen, immer eindeutig sein. Es gilt also in jedem Zustand für die Conditions ϕ eines Modells:

$$\bigvee_i \phi_i \equiv true \text{ für alle Conditions } \phi$$

$$\forall i \neq j \ : \ \phi_i \wedge \phi_j \equiv false \text{ für jede Condition}$$

[GO10]

Komponenten

Komponenten dienen als strukturierendes Element in SAML und können mit einem Namespace verglichen werden. Sie können hierarchisch verwendet werden.
Darüber hinaus bilden Komponenten für die Modellierung eine abgeschlossene Umgebung. So müssen die Zustandsübergänge nur für Variablen der Komponente angegeben werden. Das gesamte Modell kann durch die Bildung des Kreuzproduktes aller Teilautomaten gebildet werden.

```
component <componentname>
  <enum declaration if necessary>
  <variable declarations if necessary>
  <assignments in relation to the variable declaration>
  <other components if necessary>
endcomponent
```

Quelltext 2.5: SAML-Syntax Komponente

Formeln

Zur besseren Strukturierung des Quelltextes können die aussagenlogischen Ausdrücke der Bedingungen in Formeln ausgelagert werden.

```
formula <formulaname> := <condition>;
```

Quelltext 2.6: SAML-Syntax Formel

Auf diese Art und Weise ist es möglich, den Quelltext besser zu strukturieren und Duplikate zu vermeiden. Eine weitere Nutzung von Formeln sind Templates.

Templates

Um mehrere Komponenten mit ähnlichem Verhalten modellieren zu können, ohne Duplikate im Quelltext zu bekommen, ist es in SAML möglich, Templates anzulegen und beliebig viele Komponenten davon abzuleiten. Um diese Templates beim Ableiten spezialisieren zu können, werden abstrakte Formeln eingebaut, die erst in der konkreten Komponente definiert werden.

```
template <templatename>
  formula <formulaname>;
  ...
  <variable declarations>
  <assignments>
endtemplate

component <componentname> instanceof <templatename> with
  <formulaname> := <condition>;
endcomponent
```

Quelltext 2.7: SAML-Syntax Template

Templates beschreiben damit einen Automaten mit seinen Zustandsübergängen. Lediglich die Bedingungen, unter welchen Umständen ein Zustandsübergang schaltet, sind offen und werden erst festgelegt, wenn Komponenten von dem Template abgeleitet werden.

2.3 Verifikation und Validierung

Zur Verifikation und Validierung sollen Verfahren des Model-Checkings genutzt werden. Dazu muss das SAML-Modell als Zustandsautomat aufgefasst werden. Jede Zustandsvariable im SAML-Modell stellt dabei für sich einen Automaten dar. Der

gesamte Automat ist das Produkt aller Teilautomaten. Die Analysen beim Model-Checking basieren auf der Beschreibung von Eigenschaften von Zuständen (Zustandsformel) oder von Pfaden (Pfadformel).
Um zu zeigen, dass das modellierte Stellwerk korrekt funktioniert – um es also verifizieren zu können – muss festgelegt werden, was „korrekt“ ist. Dazu werden die oben genannten Sicherheitskriterien als Invarianten beschrieben. Diese Invarianten geben Zustandsformeln in Form von aussagenlogischen Ausdrücken an. Bei der Verifikation müssen sie in jedem Zustand, den das Modell erreichen kann, zu „wahr“ ausgewertet werden können.
Da jedoch das reale Stellwerk verifiziert werden soll, muss gezeigt werden, dass sich Modell und reales System hinreichend ähnlich verhalten. Das heißt, das Modell muss validiert werden. Dazu muss das reale Verhalten eines Stellwerkes spezifiziert werden, um dann am Modell geprüft zu werden. Da es dabei um Abläufe von Zuständen geht, reichen Invarianten mit aussagenlogischen Ausdrücken dafür nicht immer aus. Um solche Sequenzen von mehreren Zuständen (Pfade) beschreiben zu können, wird in dieser Arbeit LTL (Linear Temporal Logic) verwendet. Im Gegensatz zu Invarianten werden hier aussagenlogische Ausdrücke mit Angaben zur Zustandsabfolge ergänzt.

Linear Temporal Logic (LTL)
Linear Temporal Logic (LTL) [. . .] besteht aus Formeln der Form $\boldsymbol{A}f$*, wobei* f *eine Pfadformel ist, in der die einzigen möglichen Zustandsformeln atomare Ausdrücke sind. Genauer ist eine LTL Pfadformel:*

- *Wenn* $p \in AP$*, dann ist* p *eine Pfadformel [, wobei* AP *die Menge aller Zustandsbeschreibungen ist].*
- *Wenn* f *und* g *Pfadformeln sind, dann sind* $\neg f$*,* $f \wedge g$*,* $f \vee g$*,* $\boldsymbol{X}f$*,* $\boldsymbol{F}f$*,* $\boldsymbol{G}f$*,* $f\,\boldsymbol{U}g$ *und* $f\,\boldsymbol{R}g$ *Pfadformeln.*

aus [CGP01]

Im Folgenden werden die in dieser Arbeit verwendeten LTL-Operatoren kurz vorgestellt. Tiefergehende Erläuterungen finden sich in [CGP01].
M bezeichnet den Produktautomaten, der durch das SAML-Modell beschrieben wird, π ist ein Pfad im Automaten M. π^i ist ein Teilpfad von π beginnend an Zustand i (s_i). f bezeichnet eine Pfadformel. Pfadformeln können Zustandsformeln oder mit einem LTL-Operator verbundene Pfadformeln sein.

Globaly – G

$$M, \pi \models \mathbf{G}f \quad \Leftrightarrow \quad \forall i \geq 0 \text{ gilt: } M, \pi^i \models f$$

Der Operator Globaly (**G**) besagt, dass die Bedingung f für alle folgenden Zustände auf allen möglichen Ausführungspfaden des Systems gelten muss.

Finaly – F

$$M, \pi \models \mathbf{F} f \quad \Leftrightarrow \quad \exists k \geq 0, \text{ so dass gilt: } M, \pi^k \models f$$

Der Operator Finaly (**F**) besagt, dass die Bedingung f auf jedem sich anschließenden Ausführungspfaden des Systems in einem der Zustände gelten muss.

2.3.1 Verwendete Algorithmen zur Verifikation und Validierung

Zur Verifikation wurde eine Auswahl an Model-Checkern genutzt, dabei kamen entsprechend den Anfordernungen unterschiedliche Algorithmen zum Einsatz. Dies lassen sich in zwei grundlegende Typen einordnen

Bounded Model-Checking

Bounded Model-Checking verfolgt eine vorwärts gerichtete Suche nach Zuständen oder Pfaden, die der Spezifikation widersprechen. Dabei terminiert der Algorithmus nur bei inkorrekten Modellen. Demzufolge kann mit Bounded Model-Checking nur gezeigt werden, dass ein Modell einer Spezifikation widerspricht, es kann jedoch nicht gezeigt werden, dass ein Modell eine gegebene Spezifikation erfüllt. Bounded Model-Checking kann Spezifikationen in Form von Invarianten und LTL-Formeln überprüfen. Tiefgreifendere Erläuterungen finden sich in [BCCZ99].

IC3 und k-live

Dem Algorithmus IC3 (Incremental Construction of Inductive Clauses for Indubitable Correctness) liegt die Idee zugrunde, induktiv eine Invariante aufzustellen, die die erreichbaren Zustände beschreibt. Dadurch kann der Algorithmus eine vollständig bewiesene Aussage über die Korrektheit des Modells bezüglich der gegebenen Spezifikation treffen. Der ursprüngliche IC3 kann nur Spezifikationen in Form von Invarianten verarbeiten. Durch die Erweiterung k-live ist zusätzlich die Überprüfung von LTL-Formeln möglich. Weiterführende Erläuterungen zum IC3 finden sich in [Bra11].

3. Modellierung der Basiskomponenten

Für die Modellierung des Systems müssen die physischen Komponenten, wie Gleise, Weichen, Signale und Züge, sowie die Logik der Stellwerkstechnik abgebildet werden. Das Stellwerk nimmt dabei eine überwachende und koordinierende Position gegenüber den physischen Elementen ein, von hier aus werden z. B. Weichen angewiesen ihre Lage zu verändern oder Signale gestellt. Außerdem hat das Stellwerk den Überblick über belegte und freie Gleisabschnitte. Daraus ergibt sich, dass für alle physischen Elemente (Gleisabschnitte, Weichen und Signale) passende Kontrolleinheiten im Stellwerk vorhanden sein müssen. Die Logik des Stellwerkes wird in Form von Fahrstraßen abgebildet. Bei der hier vorgestellten Modellierung werden immer eine physische Komponente und die dazugehörige Kontrolleinheit im Stellwerk zusammen betrachtet.
Um individuelle Stellwerke modellieren zu können, soll ein Baukastensystem beschrieben werden. Dazu wird je eine Basiskomponente für Weichen, Gleisabschnitte, Signale und Fahrstraßen angelegt, die das generelle Verhalten dieser Komponente wiedergeben. Wird ein konkretes Stellwerk modelliert, so müssen lediglich die Bedingungen zum Auslösen der Funktionalitäten an die örtlichen Gegebenheiten angepasst werden. Die Basiskomponenten wurden in SAML als Templates modelliert. In Kapitel 5 wird beschrieben, wie sich diese Komponenten im Rahmen einer Fallstudie zu einem Bahnhofsstellwerk zusammenfügen lassen.
Da der Fokus dieser Arbeit auf der Stellwerkslogik liegt, sollen nur Komponenten abgebildet werden, die direkten Einfluss auf Schalthandlungen des Stellwerkes haben. So beeinflussen Züge das Stellwerk nur indirekt, indem sie Gleiskontakte auslösen, woraus sich ableitet, ob ein Gleisabschnitt belegt oder frei ist. In diesem Modellierungsansatz soll dieses Verhalten auf die Gleisabschnitte reduziert werden. Dies geschieht, indem Gleisabschnitte auf Basis des Zustandes benachbarter Gleisabschnitte belegt und wieder als frei gemeldet werden. Ein Zug zeigt sich damit, wie im realen System, durch eine Kette belegter Gleisabschnitte. Dieser Ansatz bringt einen weiteren Vorteil, denn bei der Modellierung von Zügen wäre man gezwungen, sich auf eine bestimmte Anzahl festzulegen. Ebenso wurde der Fahrdienstleiter

(bzw. Stellwärter) in dem Modell vernachlässigt, da er im normalen Betrieb keinen Einfluss auf die Sicherheit der Stellwerkstechnik hat. Seine Aufgabe besteht darin, mit Blick auf die betrieblichen Abläufe, die richtige Fahrstraße für den richtigen Zug zu stellen, sowie die Zugfolge zu koordinieren.

3.1 Gleisabschnitt (Track)

```
1  enum state := [clear, reserved, occupiedBy1, occupiedBy2];
2
3  template TrackTemplate
4      state : trackStates init clear;
5
6      formula reserveRequest;
7      formula occupiedContract_0;
8      formula occupiedContract_1;
9      formula occupiedContract_2;
10     formula clearContract;
11
12     isClear & reserveRequest
13         -> state' = trackStates.reserved;
14     isClear & !reserveRequest & occupiedContract_1
15         -> choice: state' = trackStates.occupiedBy1
16         + choice: state' = state;
17     isClear & !reserveRequest & !occupiedContract_1 & occupiedContract_2
18         -> choice: state' = trackStates.occupiedBy2
19         + choice: state' = state;
20     isClear & !reserveRequest & !occupiedContract_1 & !occupiedContract_2 &
           occupiedContract_0
21         -> choice: state' = trackStates.occupiedBy1
22         + choice: state' = trackStates.occupiedBy2;
23     isClear & !reserveRequest & !occupiedContract_1 & !occupiedContract_2 &
           !occupiedContract_0
24         -> state' = state;
25
26     isReserved & occupiedContract_1
27         -> choice: state' = trackStates.occupiedBy1
28         + choice: state' = state;
29     isReserved & !occupiedContract_1 & occupiedContract_2
30         -> choice: state' = trackStates.occupiedBy2
31         + choice: state' = state;
32     isReserved & !occupiedContract_1 & !occupiedContract_2
33         -> state' = state;
34
35     state = trackStates.occupiedBy1 & clearContract
36         -> choice: state' = trackStates.clear
37         + choice: state' = trackStates.occupiedBy1;
38     state = trackStates.occupiedBy2 & clearContract
39         -> choice: state' = trackStates.clear
40         + choice: state' = trackStates.occupiedBy2;
41     isOccupied & !clearContract
42         -> state' = state;
43   endtemplate
```

Quelltext 3.1: Track Template

Eine Fahrstraße setzt sich aus mehreren, aufeinanderfolgenden Gleisabschnitten zusammen. In diesem Modell sollen Gleisabschnitte ihren Zustand (belegt oder frei) kennen, wissen, ob sie Teil einer gestellten Fahrstraße (reserviert) sind, sowie Zugnummer (Zugidentifikator) und Fahrtrichtung des Zuges speichern. Dafür kann jeder Gleisabschnitt „Frei“ (clear), „Reserviert“ (reserved) oder „Belegt“ (occupied) sein (vgl, Quelltext 3.1 Zeile 1). Bei der Belegung wird zwischen zwei unterschiedlichen Zugidentifikatoren unterschieden. Diese sind nötig, um bei der Verifikation zwischen

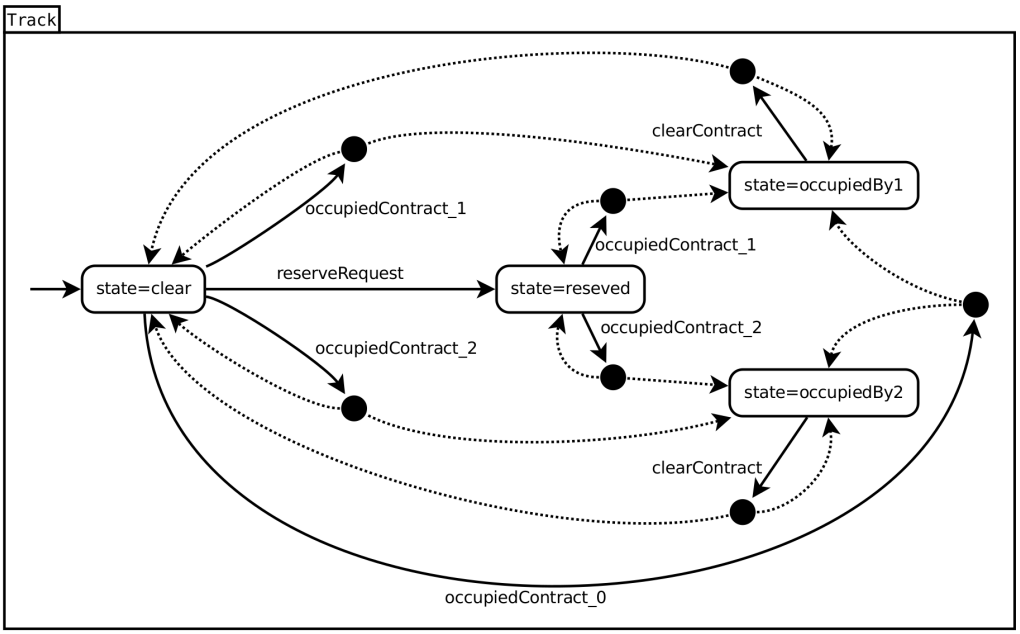


Abbildung 3.1: Automatenmodell der Gleisabschnitte

einem langen Zug (gleiche Zugidentifikatoren) und Kollisionen (unterschiedliche Zugidentifikatoren) unterscheiden zu können. Näheres dazu findet sich in Kapitel 4. Die Gleisabschnitte werden als „Frei“ initialisiert.
Ein Gleisabschnitt kann durch Fahrstraßen reserviert werden. Er geht dabei in den Zustand „Reserviert“ über, wenn er noch nicht belegt ist (vgl. Übergang im Automatenmodell Abbildung 3.1 und Quelltext 3.1 Zeilen 12f). In der Formel *reserveRequest* wird dafür eine Disjunktion aller hier entlangführenden Fahrstraßen eingetragen. Die Belegung der Gleise soll in Abhängigkeit vom Zustand der benachbarten Gleisabschnitte erfolgen. Der Zugidentifikator (Zugnummer) soll dabei erhalten bleiben. Dazu werden entsprechende Formeln (*occupiedContract_1* und *occupiedContract_2* entsprechend der Zugnummer 1 oder 2) angelegt. Diese Formeln sollen im Anschluss dann so überschrieben werden, dass sie „wahr“ werden, wenn sich auf einem benachbarten Gleisabschnitt ein Zug mit der entsprechenden Zugnummer befindet, der im nächsten Schritt diesen Gleisabschnitt befahren kann.
Da der Initialzustand des Modells nur leere, also freie, Gleise vorsieht, muss es die Möglichkeit geben, Züge auf den Gleisen zu platzieren. Dazu werden freie Gleisabschnitte, die Startpunkt einer gestellten Fahrstraße sind, direkt auf „Belegt“ gesetzt (vgl. Quelltext 3.1 Zeilen 20-22). In diesem Schritt wird zudem der nicht eindeutige Zugidentifikator (die Zugnummer) mit einem Indeterminismus zugewiesen. Ausgelöst wird dieses Verhalten durch die generische Formel *occupiedContract_0*. Sie muss bei der Modellierung eines konkreten Stellwerks mit einer Disjunktion der Fahrstraßen überschrieben werden, für die hier ein Zug platziert werden soll. Gleisabschnitte können aus jedem Zustand heraus belegt werden (vgl. Abbildung 3.1). Das bedeutet, dass ein Gleis nicht reserviert sein muss, um belegt werden zu können. Damit wird sichergestellt, dass Fehler in der Stellwerkslogik in einen erkennbaren sicherheitskritischen Zustand führen. Die Freimeldung des Gleisabschnittes kann aus den „Belegt“-Zuständen erfolgen, sobald der in Fahrtrichtung nächste Abschnitt belegt

ist und der vorhergehende Abschnitt frei ist. Das stellt sicher, dass Züge nicht vom Gleisplan verschwinden können, aber auch nicht zerteilt werden. Im konkreten Modell muss die Formel *clearContract* überschrieben werden, um das Verhalten an die örtlichen Gegebenheiten anzupassen.
Ein Zug ist dabei nicht gezwungen, in jedem Schritt einen neuen Gleisabschnitt zu befahren oder einen Alten zu räumen (vgl. nichtdeterministische Übergänge in Abbildung 3.1 und im Quelltext 3.1). Dieses Verhalten macht es möglich, unterschiedlich lange Züge abzubilden.

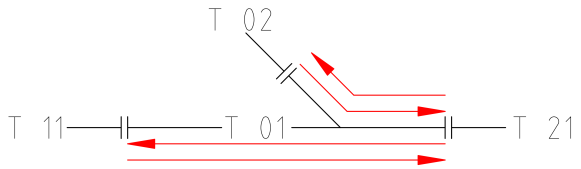


Abbildung 3.2: Ein Gleisabschnitt mit einer Weiche: In Rot sind die Zuständigkeiten der aus dem Template abgeleiteten Komponenten dargestellt.

Zur Veranschaulichung soll die Anwendung des vorgestellten Templates an Gleis T 01 des kleinen Beispielbahnhofes gezeigt werden. Um die Fahrtrichtung der Züge zu speichern, wird für jede Fahrbeziehung eine eigene Komponente abgeleitet (rote Pfeile in Abbildung 3.2). Die Reservierungsanfrage (*reserveRequest*) soll ausgelöst werden, wenn ein dort entlangführende Fahrstraße die Erlaubnis der Stellwerkslogik bekommt. Für z. B. die Komponente T 01 (T 21 → T 11) ist das nur die Fahrstraße C → T 11. Da auf T 01 keine Fahrstraße beginnt, gilt für alle vier Repräsentanten: *occupiedcomntract_0 := false*. Damit Züge die Gleisabschnitte befahren können, also weitergereicht werden, müssen die Formeln *occupiedContract_1* und *occupiedContract_2* angepasst werden. Der Gleisabschnitt wird z. B. von T 21 nach T 02 befahren, wenn Gleis T 21 in Richtung T 01 belegt ist und sich die Weiche auf T 01 in abzweigender Lage befindet. Für dieselbe Fahrtrichtung, jedoch hin zu T 11, ändert sich in der Bedingung die Weichenlage. Der Unterschied zwischen *occupiedContract_1* und *occupiedContract_2* liegt lediglich in der Zugnummer, mit der T 21 belegt ist. So bleibt der Identifikator des Zuges erhalten.

Abbildung 3.3: Ein Gleisabschnitt zum Richtungswechsel: In Rot ist die aus dem Template abgeleitete Komponente dargestellt, sie ist für beide Fahrtrichtungen zuständig.

Bei Gleisen, auf denen Züge ihre Fahrtrichtung ändern können (in diesem Beispiel T 12; Abbildung 3.3), soll die Fahrtrichtung nicht gespeichert werden. Demzufolge existiert dort nur eine Komponente, die für beide Fahrtrichtungen gilt.
Ein Zug zeigt sich durch eine Kette von belegten Gleisabschnitten, dabei kann immer das letzte Element zu „Frei“ werden. Für die Komponente von T 01 (Abbildung 3.2) die die Beziehung T 21 → T 02 abbildet, heißt das also, dass T 01 frei werden kann, wenn T 21 frei ist und T 02 aus Richtung T 01 belegt ist. Mit dieser Bedingung wird die Formel *clearContract* überschrieben.

3.2 Signal

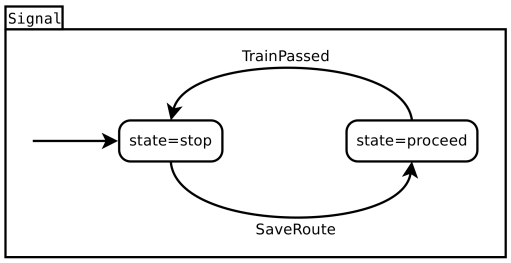


Abbildung 3.4: Automatenmodell der Signale

Signale können sich in den Zuständen „Halt“ (stop oder rot) und „Fahrt“ (proceed oder grün) befinden. Zustände, die die Weiterfahrt mit verminderter Geschwindigkeit erlauben, wurden vernachlässigt.
Initialisiert werden alle Signale im Zustand „Halt“. Sie schalten auf „Fahrt“, wenn eine sichere Weiterfahrt möglich ist. Diese Bedingung wird im konkreten Fall mit der Formel *SaveRoute* ausgedrückt. Nachdem der Zug das Signal passiert hat, fällt es sofort wieder auf „Halt“ zurück. Um es an die örtlichen Gegebenheiten anzupassen, muss dafür die Formel *TrainPassed* überschrieben werden.
Das Verhalten der Signale ist im Quelltext 3.2 sowie in dem Automatenmodell (Abbildung 3.4) abgebildet. Beispielhaft soll hier die Umsetzung am Signal C knapp beschrieben werden. Dabei wird in die Formel *SaveRoute* die Fahrstraße C → T 21 eingetragen. Das Signal gilt als passiert, wenn T 02 belegt ist. Dies wird in *TrainPassed* eingetragen.

```
1  enum signalStates := [stop, proceed];
2
3    template SignalTemplate
4      state : signalStates init stop;
5
6      formula SaveRoute;
7      formula TrainPassed;
8      formula proceed := state = signalStates.proceed;
9
10     state = signalStates.stop & SaveRoute -> state' = signalStates.proceed;
11     state = signalStates.proceed & TrainPassed -> state' = signalStates.stop;
12     (state = signalStates.proceed & !TrainPassed)
13       | (state = signalStates.stop & !SaveRoute)
14       -> state' = state;
15   endtemplate
```

Quelltext 3.2: Signal Template

3.3 Weiche (Switch)

Eine Weiche befindet sich in einem von zwei Zuständen bezüglich ihrer Lage. Das sind „Normal“ und „Abzweigend“ (reverse). Dazu kann eine Weiche verriegelt (locked) oder unverriegelt (unlocked) sein. So ergeben sich vier Zustände pro Weiche.
Weichen werden in normaler Lage und unverriegelt initialisiert. Jede der zwei Weichenlagen wird durch eine Schaltbedingung ausgelöst. Damit bei einer konkreten

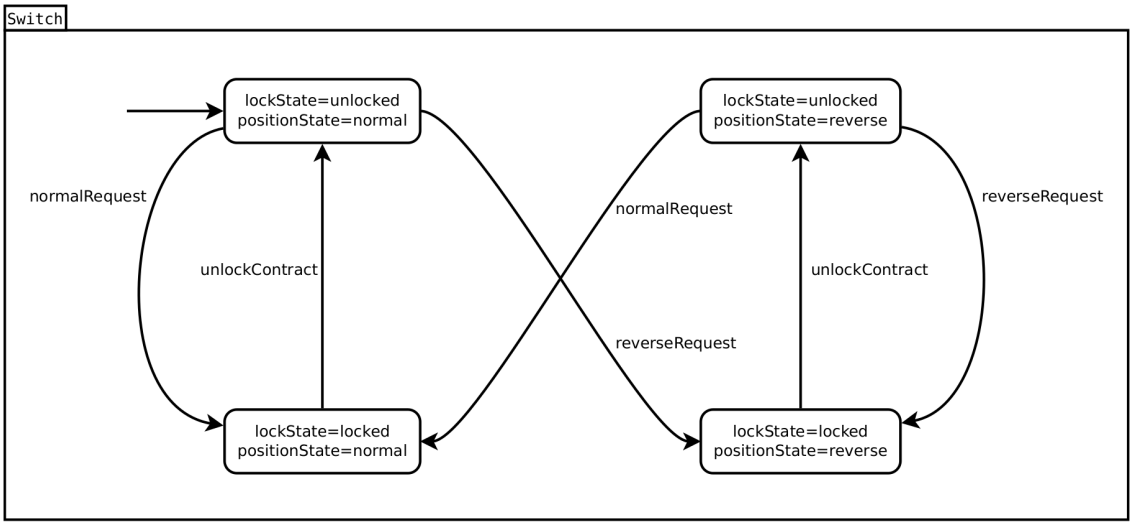


Abbildung 3.5: Automatenmodell der Weichen

Umsetzung an dieser Stelle die entsprechenden Fahrstraßen eingefügt werden können, werden dafür die Formeln *normalRequest* und *reverseRequest* angelegt, die dann überschrieben werden (vgl. Quelltext 3.3 Zeilen 8f). Ist eine dieser Schaltbedingungen ausgelöst, sowie die Weiche unverriegelt, verändert die Weiche ihre Lage entsprechend und verriegelt (vgl. Quelltext 3.3 Zeilen 12-16 und das Automatenmodell in Abbildung 3.5). Das Entriegeln erfolgt, wenn die entsprechende Bedingung (*unlockContract*) ausgelöst wird.
Im kleinen Beispielbahnhof soll z. B. die Weiche P 01 in normale Lage gebracht werden, wenn die Fahrstraßen B → T 21 oder A → T 12 gestellt werden. Das wird in

```
1  enum lockState := [unlocked, locked];     // states of locking
2  enum positionState := [normal, reverse];  // states of position
3
4  template SwitchTemplate
5      lock : lockStates init lockStates.unlocked;
6      position : positionStates init positionStates.normal;
7
8      formula normalRequest;
9      formula reverseRequest;
10     formula unlockContract;
11
12     lock = lockStates.unlocked & normalRequest & !reverseRequest
13         -> position' = positionStates.normal & lock' = lockStates.locked;
14
15     lock = lockStates.unlocked & reverseRequest & !normalRequest
16         -> position' = positionStates.reverse & lock' = lockStates.locked;
17
18     (lock = lockStates.locked & (!unlockContract | normalRequest | reverseRequest))
19       | (lock = lockStates.unlocked & (
20          (normalRequest & reverseRequest)
21         | !(normalRequest | reverseRequest)
22       ))
23         -> position' = position & lock' = lock;
24
25     lock = lockStates.locked & unlockContract & !normalRequest & !reverseRequest
26         -> position' = position & lock' = lockStates.unlocked;
27   endtemplate
```

Quelltext 3.3: Switch Template

der Formel *normalRequest* abgebildet. Die Weiche soll entriegeln, wenn der Zug das Gleis T 01 verlassen hat, wenn T 01 also wieder im Zustand frei ist. Dieses Verhalten wird in die Formel *clearContract* geschrieben.

3.4 Fahrstraße (Route)

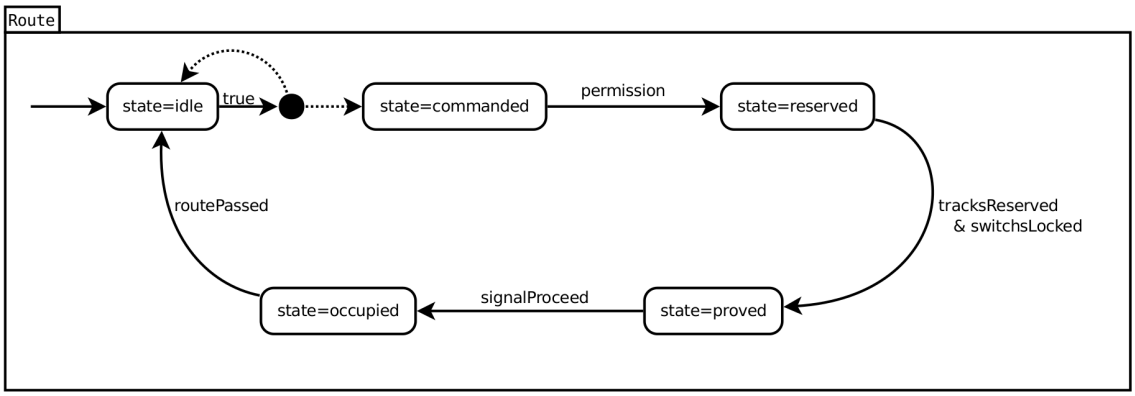


Abbildung 3.6: Automatenmodell der Fahrstraßen

Fahrstraßen überwachen als eine Art übergeordnete Verbindungseinheit den sicheren Betrieb. Dabei wird zwischen Fahrstraßen für Zugfahrten und für Rangierfahrten unterschieden. Für Fahrstraßen, die Zugfahrten absichern sollen, gelten höhere Sicherheitsanforderungen als für Rangierfahrstraßen. Rangierfahrten sollen in dieser Arbeit vernachlässigt werden. Im Modell durchlaufen Fahrstraßen fünf Zustände. Eine Fahrstraße wird als „Frei“ (idle) initialisiert.
Im realen Ablauf kann sie durch einen Stellwärter angewiesen werden. Im Modell ist dies umgesetzt, indem eine freie Fahrstraße in jedem Schritt zu „Angewiesen“

```
1  enum routeStates := [idle, commanded, reserved, proved, occupied];
2
3    template RouteTemplate
4      state : routeStates init idle;
5
6      formula tracksFree;
7      formula permission;
8      formula tracksReserved;
9      formula SwitchsLocked;
10     formula signalProceed;
11     formula routePassed;
12
13     state = routeStates.idle -> choice: state' = state
14                               + choice: state' = routeStates.commanded;
15
16     state = routeStates.commanded & permission -> state' = routeStates.reserved;
17     state = routeStates.commanded & !permission -> state' = state;
18     state = routeStates.reserved & tracksReserved & SwitchsLocked
19       -> state' = routeStates.proved;
20     state = routeStates.reserved & !(tracksReserved & SwitchsLocked) -> state' = state;
21     state = routeStates.proved & signalProceed -> state' = routeStates.occupied;
22     state = routeStates.proved & !signalProceed -> state' = state;
23     state = routeStates.occupied & routePassed -> state' = routeStates.idle;
24     state = routeStates.occupied & !routePassed -> state' = state;
25   endtemplate
```

Quelltext 3.4: Route Template

(commanded) wechseln kann oder im Zustand „Frei“ verbleibt (vgl. Quelltext 3.4 Zeilen 13f und den nichtdeterministischen Übergang in Abbildung 3.6).
Eine angewiesene Fahrstraße kann in den Zustand „Reserviert“ (reserved) wechseln, wenn kein Fahrstraßenelement dem entgegensteht, dazu bekommt sie die Erlaubnis der Stellwerkslogik. Ist dies nicht möglich, verbleibt die Fahrstraße im Zustand „Angewiesen“, dies entspricht einem Fahrstraßenspeicher.
Bevor die Fahrstraße in den Zustand „Überprüft“ (proved) übergeht, wird abgefragt, ob alle Gleisabschnitte reserviert (*tracksReserved*), sowie alle Weichen korrekt gestellt und verriegelt sind (*switchesLocked*).
Sobald der Zug die Fahrstraße befahren kann, also das Startsignal „Fahrt“ zeigt, gilt sie als „Belegt“ (occupied) und wird erst wieder freigegeben, wenn sie vollständig passiert wurde (*signalProceed* und *routePassed*).

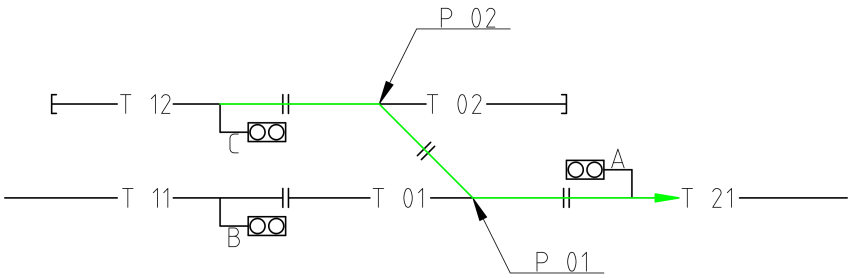


Abbildung 3.7: Beispiel einer Fahrstraße: In grün ist die Fahrstraße C → T 21 zu sehen.

Für die Fahrstraße von Signal C nach Gleis T 21 bedeutet das, dass die Gleise T 02, T 01 und T 21 frei sein müssen, damit die Fahrstraße in den Zustand Reserviert übergehen kann. Daraufhin müssen die Weichen P 01 und P 02 in abzweigende Lage gestellt werden. Sind die Gleise T 02, T 01 und T 21 reserviert und die Weichen P 01 und P 02 in abzweigender Lage verriegelt, dann kann die Fahrstraße in den Zustand „Überprüft“ übergehen, worauf das Signal C auf „Fahrt“ schaltet. Wenn der Zug das Gleis T 01 geräumt hat, geht die Fahrstraße wieder in ihren Ausgangszustand zurück.

3.5 Stellwerkslogik (InterlockingPermission)

```
1  component InterlockingPermission
2    formula A11 := (Route.A_11.state = Route.routeStates.commanded
3              & Route.A_11.tracksFree);
4    formula A12 := !(Route.A_11.state = Route.routeStates.commanded
5              & Route.A_11.tracksFree)
6            & (Route.A_12.state = Route.routeStates.commanded
7              & Route.A_12.tracksFree);
8    [...]
9  endcomponent
```

Quelltext 3.5: Ausschnitt aus der InterlockingPermission

Durch die Modellierung paralleler Automaten kann es vorkommen, dass zwei konkurrierende Fahrstraßen gleichzeitig schalten wollen und sich so bei ihrer Ausführung gegenseitig behindern. Um dies zu verhindern, wurde eine übergeordnete Komponente erstellt, die für den Fall, dass mehrere Fahrstraßen schalten können, nur einer

einzigen Fahrstraße die Erlaubnis dazu gibt. Dies wurde über die Vergabe von Prioritäten realisiert. In Quelltext 3.5 ist ein Auszug aus der Realisierung der Prioritätenvergabe abgebildet.

4. Spezifikationen

Die zentrale Fragestellung ist, ob die Stellwerkslogik es dem System ermöglicht, in einen sicherheitskritischen Zustand zu gelangen. Um dies zu überprüfen, wurden verschiedene Spezifikationen aufgestellt, anhand derer das Modell mithilfe von Verfahren des Model-Checkings verifiziert werden kann.
Die Erkenntnisse aus dem Verifikationsprozess lassen sich jedoch nur vom Modell in die Realität übertragen, wenn sich das Modell hinreichend ähnlich wie das reale System verhält. Dazu wurden Verhaltensrichtlinien spezifiziert, an denen das Modell validiert wird.

4.1 Sicherheitsspezifikationen

Die im Folgenden gezeigten Sicherheitsspezifikationen sollen zeigen, dass das Stellwerksmodell nicht in sicherheitskritische Zustände gelangen kann.
Ein sicherer Zugbetrieb wird an einem Satz allgemeingültiger Regeln festgemacht. So müssen Kollisionen und Entgleisungen ausgeschlossen werden. Darüber hinaus sind zur Vermeidung von seitlichen Kollisionen durch unbeabsichtigt rollende Wagen für Reisezüge mit Geschwindigkeiten über 160 km/h in der Eisenbahn-Bau- und Betriebsordnung [Bun16] Flankenschutzvorkehrungen durch Schutzweichen vorgeschrieben.
Die Verifikation des Modells erfolgt über Invarianten, die im Einzelnen in den folgenden Abschnitten beschrieben sind.

4.1.1 Kollisionsfreiheit

Zwei Züge kollidieren miteinander, wenn sie sich zeitgleich an ein und demselben Ort befinden. Da das Stellwerksmodell keine exakte Ortsbestimmung zulässt, sondern nur abbildet, welche Gleisabschnitte besetzt sind, ist die Belegung eines Gleisabschnittes durch zwei Züge als Zusammenstoß anzusehen.
Frontale Kollisionen sowie die seitliche Kollision von Zügen können dadurch festgestellt werden, dass mehrere Segmente, die demselben Gleisabschnitt zugeordnet

```
formula isCollision :=
  (
    (if T_01_n_lr.isOccupied then 1 else 0)
    + (if T_01_n_rl.isOccupied then 1 else 0)
    + (if T_01_r_lr.isOccupied then 1 else 0)
    + (if T_01_r_rl.isOccupied then 1 else 0)
  ) > 1;
```

Quelltext 4.1: Abfrage von frontalen sowie seitlichen Zusammenstößen am Beispiel des Gleisabschnittes T 01

```
formula isCollision :=
  (occupiedContract_1 & state = state.occupiedBy2)
  | (occupiedContract_2 & state = state.occupiedBy1);
```

Quelltext 4.2: Abfrage von Auffahrunfällen für einen Gleisabschnitt

sind, durch einen Zug belegt sind. Dazu werden alle Repräsentanten eines Gleisabschnittes, die im Zustand „Belegt“ sind, gezählt (vgl. Quelltext 4.1).
Jede Gleiskomponente kann selbstständig erkennen, wenn sie belegt ist und ein weiterer Zug auf den Gleisabschnitt fährt. Da die Gleiskomponenten nicht zählen können, muss die Gefahr unmittelbar bevor es zur Kollision kommt erkannt werden. Dazu werden, wie in Kapitel 3 beschrieben, zwei verschiedene Zugnummern vergeben. Im Zusammenhang mit dem im Model-Checking aufgebauten Zustandsgraphen des Produktautomaten wird so jede mögliche Kollision vier Mal auftreten. Bei zwei Fällen kollidieren Züge mit gleicher Zugnummer. Dies kann nicht erkannt werden, da es keinen Unterschied gibt, wenn ein langer Zug sich über die Gleise bewegt, oder zwei kurze Züge mit derselben Zugnummer aufeinander auffahren. In den anderen beiden Fällen kollidieren aber Züge mit unterschiedlichen Zugnummern. Diese Zusammenstöße werden erkannt, indem ein Gleisabschnitt, der durch einen Zug belegt ist, durch einen Zug mit anderer Zugnummer belegt werden soll (vgl. Quelltext 4.2).

4.1.2 Entgleisungen

Da das Modell keine Geschwindigkeiten beachtet, werden Entgleisungen durch überhöhte Geschwindigkeit ausgeschlossen. Zur Entgleisung kann es aber sehr wohl kommen, wenn Weichen falsch gestellt sind und durch einen Zug aufgefahren werden oder wenn Weichen unter einem Zug umgelegt werden.

```
formula isDerailed :=
  (
    (T_01_n_lr.isOccupied | T_01_n_rl.isOccupied)
          // train on T 01 from T 21 to T 11 or the other way round
    & !P_01.lockedNormal
          // and P 01 is not locked in the normal position
  ) | (
    (T_01_r_lr.isOccupied | T_01_r_rl.isOccupied)
          // train on T 01 from T 21 to T 02 or the other way round
    & !P_01.lockedReverse
          // and P 01 is not locked in the reverse position
  );
```

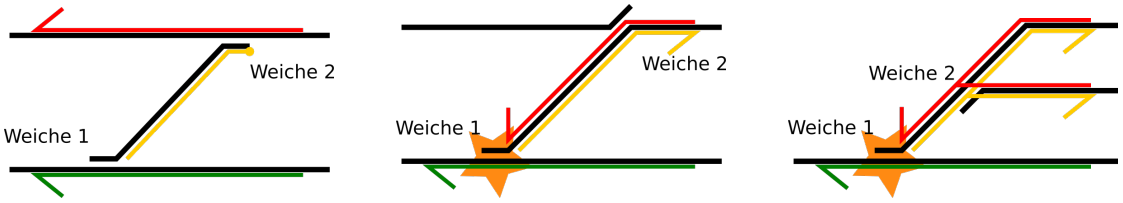
Quelltext 4.3: Detektion von Engleisungen am Beispiel von P 01 auf Gleis T 01

Dazu wird überprüft, ob die Weichenlage mit dem aktuellen befahrenen Gleisabschnitt übereinstimmt. Wird eine Weiche zum Beispiel in abzweigender Richtung

befahren und befindet sich nicht in abzweigender Lage oder ist nicht verriegelt, so wird dies als Entgleisung erkannt (vgl. Quelltext 4.3).

4.1.3 Flankenschutz

Um den Flankenschutz sicherzustellen, sind verschiedene Methoden möglich, in dieser Arbeit soll der Flankenschutz durch Schutzweichen verifiziert werden. Im Beispielbahnhof bedeutet das, dass für die Fahrstraßen A → T 11 und B → T 21 zusätzlich die Weiche P 02 in normale Lage gebracht werden muss, damit keine Wagen von T 12 in die Flanke der durchfahrenden Züge rollen können. Stattdessen rollen sie gegen den Prellbock am geraden Ausgang von P 02.



(a) Flankenschutz sichergestellt: Der gelbe Pfad endet auf Weiche 2, die „abweisend“ gestellt ist. Ein Wagen (roter Pfad) kann nicht in die Flanke des Zuges (grüner Pfad) rollen.

(b) Flankenschutz nicht sichergestellt: Der gelbe Pfad führt in ein Bahnhofsgleis. Verfolgt man diesen Pfad zurück, erhält man den Weg zu einer Flankenkollision (roter Pfad).

(c) Flankenschutz nicht möglich: Unabhängig von der Lage der Weiche 2 gelangt der gelbe Pfad auf beide Bahnhofsgleise. Ein Wagen der rollen würde (roter Pfad) kann aus beiden Gleisen eine Flankenkollision hervorrufen.

Abbildung 4.1: Verifikation des Flankenschutzes: In Schwarz sind die Gleise dargestellt. Dabei deuten durchgezogene Linien die Weichenlage an. In Grün sind Züge dargestellt, deren Flanke geschützt werden muss. Der gelbe Pfad zeigt die Rückpropagation der Flankenkollision an, während der rote Pfad den Weg eines rollenden Wagens zeigt.

Um die Möglichkeit einer Flankenkollision festzustellen, wird bei einer befahrenen Weiche der nicht befahrene Zweig mit dem Attribut „Flankenkollision“ markiert. Darüber hinaus wird jedes Gleis mit diesem Attribut markiert, von dem aus ein Wagen auf ein mit „Flankenkollision“ markiertes Gleis fahren kann.
Dabei werden die in Abbildung 4.1 gezeigten Fälle unterschieden. Trifft das Markierungsverfahren auf eine stumpfe Weiche (vom Herzstück kommend), wird der folgende Gleisabschnitt nur markiert, wenn sie sich in der Lage befindet, in der sie einen Wagen in die Flanke leiten würde (vgl. Abbildung 4.1(a) und Abbildung 4.1(b)). Bei einer spitzen Weiche (zum Herzstück hin) werden beide folgenden Gleise markiert, da ein Wagen, der die Weiche dann stumpf befährt, sie auffahren könnte (vgl. Abbildung 4.1(c)).
Die eigentliche Spezifikation besagt, dass Bahnhofs- und Streckengleise nie mit dem Attribut „Flankenkollision“ markiert werden. Die Umsetzung im SAML-Modell ist in Quelltext 4.4 angedeutet.

```
component T_01_r_lr instanceof TrackTemplate with
  [...]
  flankCollisionPossible :=
    T_01_n_lr.isOccupied | T_01_n_rl.isOccupied;
endcomponent
component T_02_r_lr instanceof TrackTemplate with
  [...]
  flankCollisionPossible :=
    T_01_r_lr.flankCollisionPossible & P_02.position = Switch.positionStates.reverse;
endcomponent
component T_12 instanceof TrackTemplate with
  [...]
  flankCollisionPossible :=
    T_02_r_lr.flankCollisionPossible | T_02_n_lr.flankCollisionPossible;
endcomponent
INVARSPEC (!T12.flankCollisionPossible);
```

Quelltext 4.4: Detektion von Flankenschutzverletzungen: Fährt ein Zug über T 01 in normaler Lage, ist auf dem abzweigenden Abschnitt eine Flankenfahrt von links möglich. Diese Flankenfahrt wird von T 02 weitergereicht, wenn sich die Weiche P 02 in abzweigender Lage befindet. Hat T 02 durch die Stellung von P 02 die Flankenfahrt nicht abgewendet, so ist von T 12 aus eine Flankenfahrt möglich. Dies fragt die Invariante ab.

4.2 Liveness-Spezifikationen

Um zu zeigen, dass das entworfene Modell sich wie ein reales Stellwerk verhält, wurde es anhand verschiedener Spezifikationen validiert. Diese Spezifikationen wurden in LTL entworfen, um die technischen und logischen Abläufe überprüfen zu können.

4.2.1 Nutzbarkeit von Fahrstraßen

Sicherheitskritische Zustände lassen sich auch dadurch ausschließen, dass sich eine oder mehrere Fahrstraßen nicht stellen lassen. In diesem Fall ist das Modell jedoch nicht valide. Um dies zu prüfen, wird spezifiziert, dass eine Fahrstraße, die angewiesen ist und die Erlaubnis der Stellwerkslogik hat, immer bis in den Zustand „Belegt“ schalten muss (vgl. Quelltext 4.5). So wird sichergestellt, dass die Fahrstraßen korrekt funktionieren. Gleichzeitig wird aber sichergestellt, dass es keine Fehlermeldungen gibt, weil eine Fahrstraße blockiert wird, um einen sicherheitskritischen Zustand zu verhindern (false positive bzw. α-Fehler).

```
SPEC G (
  (Route.<Fahrstraße>.state = Route.state.commanded
    & Route.<Fahrstraße>.permisssion)
    => F(Route.<Fahrstraße>.state = Route.state.occupied)
);
```

Quelltext 4.5: Spezifikation zur Validierung der Fahrstraßen

4.2.2 Fahrbewegung der Züge

Ist es einem Zug aufgrund von Modellierungsfehlern nicht möglich weiter zu fahren, kann dies dazu führen, dass sicherheitskritische Zustände nicht erreicht werden können und so unbemerkt bleiben. Um dass auszuschließen, werden die Fahrbewegungen

der Züge validiert. Dazu wird für jeden Gleisabschnitt spezifiziert, dass er, wenn er belegt ist, im darauf folgenden Gleisabschnitt die Bedingung zum Belegen aktiviert (vgl. Quelltext 4.6). Ausgenommen ist der Fall, dass der Zug vor einem „Halt“ zeigenden Signal steht. Es wird also gezeigt, dass bei einem belegten Gleisabschnitt und „Frei“ zeigendem Signal (oder keinem Signal) der entsprechend der Weichenlage nächste Gleisabschnitt belegt werden kann.

```
1 SPEC G (
2   (Track.<Gleisabschnitt>.isOccupied
3     & <eventuell Halt zeigendes Signal>
4     & <Weichenlage>)
5     => (Track.<nächster Gleisabschnitt>.occupiedContract_1
6         | Track.<nächster Gleisabschnitt>.occupiedContract_2)
7 );
```

Quelltext 4.6: Spezifikation zur Validierung der Fahrbewegung

5. Fallstudie Braine l'Alleud

Um die gewählte Modellierungsidee an einem konkreten Anwendungsfall testen zu können, wurde der Bahnhof Braine l'Alleud modelliert. Der Aufbau ist [CS16] entnommen. Die Station Braine l'Alleud befindet sich in Belgien an einer zweigleisigen Hauptstrecke zwischen Brüssel und Charleroi. Keines der Gleise ist auf eine bestimmte Fahrtrichtung beschränkt.
Das zugehörige Stellwerk kontrolliert 12 Signale, 12 Weichen sowie 18 Gleisabschnitte. Unter den Gleisabschnitten sind vier Streckengleise mit Anschluss an Nachbarstellwerke und vier Gleise auf denen Zugfahrten beginnen und enden dürfen. Aus dieser Konfiguration ergeben sich 32 mögliche Fahrstraßen.
Mit dem beschriebenen Aufbau ergibt sich für das Modell des Bahnhofes Braine l'Alleud ein Zustandsgraph mit ca. $2,7 \times 10^{67}$ Knoten.

5.1 Modellierung des Gleisplans

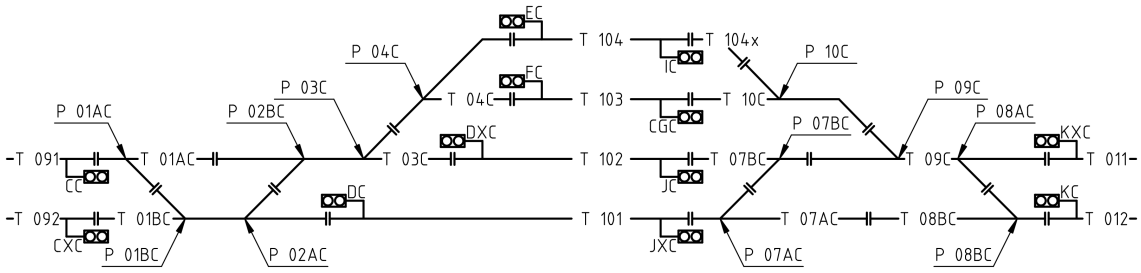


Abbildung 5.1: Gleisplan der Station Braine l'Alleud nach [CS16]

Um den Gleisplan zu modellieren, müssen aus dem in Abschnitt 3.1 beschriebenen Template für jedes Gleiselement separate Komponenten abgeleitet werden. Da die Richtung der Züge gespeichert werden muss (mit Ausnahme der Bahnhofsgleise T 101 - T 104), werden für jedes Gleis zwei Repräsentanten im Modell angelegt. Darüber hinaus werden bei Gleisabschnitten, die Weichen enthalten, die jeweiligen Abzweige separat betrachtet. Ein Gleisabschnitt, der eine Weiche enthält, hat somit

vier Repräsentanten im Modell. Zwei für die „Normale“ und zwei für die „Abzweigende“ Lage der Weiche.
Bei der Modellierung der örtlichen Gegebenheiten müssen die Signale (auch sie sind von dem entsprechenden Template abgeleitete Komponenten) an die passenden Stellen im Gleisplan gesetzt werden. Das Passieren des Zuges (Formel *TrainPassed*) wird daran erkannt, dass der Gleisabschnitt hinter dem Signal belegt ist.
Die Weichen müssen mit den Gleisabschnitten verbunden werden, zu denen sie gehören, um den Moment der Entriegelung bestimmen zu können (*unlockContract*). Eine Weiche soll entriegeln, wenn der dazugehörige Gleisabschnitt frei ist.

5.2 Modellierung der Stellwerkslogik

Nr	Start	Ziel	Gleisabschnitte	Weichen normal	Weichen abzweigend
1	CC	101	T01AC, T01BC, T101	P02AC, P02BC*	P01AC, P01BC
2	CC	103	T01AC, T03C, T04C, T103	P01AC, P01BC*, P02BC, P02AC*	P03C, P04C
3	CC	104	T01AC, T03C, T04C, T104	P01AC, P01BC*, P02BC, P02AC*, P04C	P03C
4	EC	092	T04C, T03C, T01BC, T092	P04C, P01BC, P01AC*	P03C, P02BC, P02AC
5	FC	092	T04C, T03C, T01BC, T092	P01BC, P01AC*	P04C, P02BC, P02AC, P03C
6	KC	102	T08BC, T09C, T07BC, T102	P09C, P07AC*, P07BC	P08BC, P08AC
7	KXC	102	T09C, T07BC, T102	P08AC, P08BC*, P07BC, P07AC*, P09C	
8	IC	011	T104x, T10C, T09C, T011	P08AC, P08BC*	P10C, P09C
9	JC	012	T07BC, T09C, T08BC, T012	P07BC, P07AC*, P09C	P08AC, P08BC
10	JXC	012	T07AC, T08BC, T012	P07AC, P07BC*, P08BC, P08AC*	

Tabelle 5.1: Auszug aus dem Verschlussplan von Braine l'Alleud: Die mit * markierten Weichen dienen dem Flankenschutz.

Um die Schaltlogik eines Stellwerkes festzulegen, werden Verschlusspläne entwickelt. Sie beschreiben, welche Elemente der Gleise zu bestimmten Fahrstraßen gehören und in welchem Zustand sie sich befinden müssen, um eine gefahrlose Zugfahrt zu ermöglichen. Ein Auszug aus dem Verschlussplan des Stellwerkes für Braine l'Alleud ist in Tabelle 5.1 abgebildet.
Damit eine angewiesene Fahrstraße in den Zustand „reserviert“ übergehen kann, wenn sie die Erlaubnis der Stellwerkslogik bekommt, muss die Formel *permission* mit der für die Fahrstraße passende Formel aus der Stellwerkslogik überschrieben werden. Die Stellwerkslogik gibt einer Fahrstraße die „Ausführungserlaubnis“, wenn alle Elemente (Gleisabschnitte und Weichen), die die Fahrstraße belegen möchten, nicht anderweitig verwendet werden (*tracksFree*) und es keine Fahrstraße höherer Priorität gibt, für die diese Eigenschaft auch zu trifft.

Jeder Gleisabschnitt wird entsprechend des Verschlussplanes den Fahrstraßen zugeordnet. Bekommt eine dort entlangführende Fahrstraße die Erlaubnis der Stellwerkslogik, wird der Gleisabschnitt reserviert. Dazu wird die Formel *reserveRequest* entsprechend überschrieben.
Ähnlich werden die Weichen mit den Fahrstraßen verknüpft. Hier werden die Formeln *normalRequest* und *reverseRequest* mit Disjunktionen der hier entlang führenden Fahrstraßen überschrieben. Die Fahrstraßen müssen sich dabei im Reservierungsmodus befinden.
Damit die Fahrstraße in den Status „Überprüft“ wechseln kann, müssen alle Gleisabschnitte reserviert sein (*tracksReserved*) und alle Weichen korrekt liegen sowie verriegelt sein (*switchesLocked*).
Die Signale sollen auf „Grün“ schalten, wenn eine von dort ausgehende Fahrstraße gefahrlos befahren werden kann. Dazu muss die Formel *SaveRoute* mit einer Disjunktion der von hier beginnenden Fahrstraßen überschrieben werden. Dabei müssen sich die Fahrstraßen im Zustand „Überprüft“ befinden.

5.3 Experimente

Um das Verhalten und die Ergebnisse der verschiedenen Model-Checking-Verfahren vergleichen zu können, wurden verschiedene Testläufe gestartet. Dazu wurde der Bahnhof Braine l'Alleud modelliert.
Derzeit existieren keine Model-Checker, die SAML als Eingabesprache akzeptieren. So sind Übersetzung in andere System- bzw. Modellbeschreibungssprachen nötig.
Der erste Übersetzungsschritt erfolgt dabei aus VECS herraus in das SMV-Format [CMCHG96]. Die Übersetzung von SAML nach SMV ist bewiesen korrekt [GO10]. SMV baut dabei, ähnlich wie SAML, auf Zustandsvariablen und der Beschreibung von Bedingungen für Zustandsübergänge auf.
Die meisten der verwendeten Model-Checker nutzen als Eingabe-Format jedoch AIGs (And-Inverter Graphs) [BCC+07]. Für diesen Übersetzungsschritt konnte das AIGER-Paket[1] genutzt werden. Das AIG-Format beschränkt sich auf die Darstellung der boolschen Verknüfungen UND und NICHT. Damzufolge können AIGs auch nur boolsche Variablen verarbeiten. Dazu müssen alle Zustandsvariablen des SMV-Modells in eine Kombination aus boolschen Variablen überführt werden. Dies geschieht über die binäre Kodierung durchnummerierter Zustände. Die angegebenen boolschen Operatoren überführen dann die boolschen Variablen aus dem vorhergehenden Zustand in den Nächsten.
Alle Tests liefen auf einem Rechner mit Intel i7 Prozessor bei einer Taktrate von 3,2 GHz.
Um zu zeigen, dass sicherheitskritische Fehler beim Model-Checking gefunden werden, wurden künstlich Fehler in die Stellwerkslogik eingefügt. Sie lassen sich in folgende Kategorien einordnen:

- Eine Fahrstraße reserviert sich einen benötigten Gleisabschnitt nicht.
- Eine Fahrstraße fordert eine falsche Weichenlage an.
- Eine Fahrstraße fordert eine Weiche gar nicht an.

[1] http://fmv.jku.at/aiger/

Nr	Start	Ziel	Gleisab-schnitte	Weichen normal	Weichen abzweigend	Anmerkung
1	CC	101	T01AC, T01BC, T101	P02AC, P02BC*	P01AC, P01BC	kein Fehler
2	CC	103	T01AC, T03C, T04C, T103	P01AC, P01BC*, P02BC, P02AC*	P03C, P04C	T04 nicht angefordert
3	CC	104	T01AC, T03C, T04C, T104	P01AC, P01BC*, P02BC, P02AC*, P04C	P03C	T104 nicht angefordert
4	EC	092	T04C, T03C, T01BC, T092	P04C, P01BC, P01AC*	P03C, P02BC, P02AC	T01BC nicht angefordert
5	FC	092	T04C, T03C, T01BC, T092	P01BC, P01AC*, P03C	P04C, P02BC, P02AC	P03C falsch gestellt
6	KC	102	T08BC, T09C, T07BC, T102	P09C, P07AC*	P07BC, P08BC, P08AC	P07BC falsch gestellt
7	KXC	102	T09C, T07BC, T102	P08AC, P08BC*, P07BC, P07AC*	P09C	P09C falsch gestellt
8	IC	011	T104x, T10C, T09C, T011	P08AC, P08BC*	P10C, P09C	P10C nicht gestellt
9	JC	012	T07BC, T09C, T08BC, T012	P07BC, P07AC*, P09C	P08AC, P08BC	P09C nicht gestellt
10	JXC	012	T07AC, T08BC, T012	P07AC, P07BC*, P08BC, P08AC*		Flanken-schutz P08AC nicht gestellt

Tabelle 5.2: Auszug aus dem Verschlussplan von Braine l'Alleud mit Fehlern: Fehlerhafte Einträge sind in Rot dargestellt, Fehlende in Blau geschrieben.

In Tabelle 5.2 sind die konkreten Fehler im Verschlussplan dargestellt, dabei wurden dieselben Fahrstraßen wie in Tabelle 5.1 verwendet.

5.3.1 IIMC

IIMC[2] wird an der University of Colorado entwickelt. Die Software startet parallel mehrere Algorithmen zur Problemlösung und gibt das schnellste Ergebnis zurück. Für die Software IIMC musste die NuXMV-Modellbeschreibung in ein AIG-Format übersetzt werden. Entsprechende Software findet sich im AIGER-Paket.
Für den Fall, dass Bedingungen verletzt wurden, war eine Rückkonvertierung des Gegenbeispiels nötig. Im Fall der LTL-Spezifikationen zur Validierung endet der IIMC in der originalen Implementierung in einem Segmentation Fault, wenn er bei einem Verstoß ein Gegenbeispiel mit Schleife ausgeben wollte. Im Rahmen der Arbeit wurde er so modifiziert, dass eine Ausgabe von Schleifen möglich ist.
Die Ergebnisse der Testläufe sind in Tabelle A.3 zu finden. Die Invarianten der Sicherheitsspezifikationen lassen sich in ihrer Gesamtheit in 4 min 20 sek verifizieren. Die einzelnen Spezifikationen nehmen zum Teil deutlich mehr oder weniger Zeit in Anspruch. Am aufwendigsten zeigt sich die Verifikation des Flankenschutzes. Abgesehen von dieser Anwendung verbrauchen die Verifikationen mit der parallelen Ausführung mehrerer Algorithmen weniger als 500 MB RAM (weniger als 90 MB

[2] https://github.com/mgudemann/iimc

entfallen auf den IC3). Für den Fall, dass das Modell gemäß der getesteten Spezifikation fehlerfrei ist, gelangt der IC3 als Erster zu einem Ergebnis. Die eingefügten Fehler wurden durch den IIMC zumeist in weniger als 5 sek gefunden. Für diese Aufgabe zeichnet sich der BMC als schnellste Lösung aus. Lediglich bei Verletzungen des Flankenschutzes sind längere Rechenzeiten nötig (3 min 44 sek). Hier kommt der IC3 als Erster zu einem Ergebnis. Der genutzte Arbeitsspeicher blieb bei der Verifikation der fehlerhaften Modellen mit Ausnahme der Flankenschutzverletzung insgesamt unter 500 MB (weniger als 70 MB entfallen auf IC3 bzw. BMC).
Die Überprüfung der Livenessspezifikation musste nach ca. $2\frac{1}{2}$ Wochen abgebrochen werden. Bis dahin verbrauchte IIMC für diese Spezifikation 9,6 GB RAM.

5.3.2 NuXMV

NuXMV[3] wird von der Fondazione Bruno Kessler entwickelt. Das Eingabeformat lässt sich aus VECS heraus sehr einfach generieren.
Die Ergebnisse der Testläufe mit NuXMV sind in Tabelle A.4 zusammengefasst. Die Verifikation der Sicherheitsspezifikationen wurde mit IC3 durchgeführt. Dabei zeigt sich auch hier, dass die kombinierte Verifikation aus allen Sicherheitsspezifikationen aus Sicht der Rechenzeit deutliche Vorteile gegenüber der separaten Verifikation hat. Die Gesamtheit der Sicherheitsspezifikationen war in 32 min 34 sek verifiziert. Dabei beanspruchte NuXMV weniger als 200 MB Arbeitsspeicher. Die reine Verifikation des Flankenschutzes musste nach $2\frac{1}{2}$ Wochen abgebrochen werden. Zur Verifikation der in LTL geschriebenen Livenessspezifikationen wurden k-liveness und BMC mit einer Begrenzung auf 20 Zuständen angewendet. Mit BMC konnte innerhalb von ca. 9,5 h gezeigt werden, dass innerhalb der ersten 20 Zustände des Systems sich das Modell entsprechend den Spezifikationen verhält. Die Verifikation mit k-liveness wurde nach $2\frac{1}{2}$ Wochen erfolglos abgebrochen. Die fehlerhaften Modelle wurden mit BMC und IC3 überprüft. Dabei zeigte sich der NuXMV-BMC in diesem Zusammenhang als ungeeignet. Er bricht mit der Meldung „Induction Fails“ ab. Der IC3 konnte alle Fehler innerhalb von 1min finden. Lediglich die Verletzung des Flankenschutzes beanspruchte 7 min 12 sek. Auch bei den fehlerhaften Modellen genügten 200 MB Arbeitsspeicher.

5.3.3 AIGBMC

Das AIGER-Paket, aus dem der AIGBMC stammt, wird an der Johannes Kepler Universität Linz unter Prof. Dr. Armin Biere entwickelt. Ähnlich wie für den IIMC mussten auch für AIGBMC entsprechende AIG-Modellbeschreibungen erzeugt werden. Der Bounded Model-Checking Algorithmus aus dem AIGER-Packet verifiziert Sequenzen mit bis zu 10 aufeinanderfolgenden Zuständen.
Die Ergebnisse der Testläufe mit AIGBMC sind in Tabelle A.1 zusammengefasst. Wie zu erwarten ist, kann ein Bounded Model-Checking nicht die Korrektheit des Modells zeigen. Aber auch komplexere Fehler mit langen Pfaden für Gegenbeispiele konnte er nicht finden. Für die einfacheren Fehler findet er jedoch meist innerhab von 1 min ein Gegenbeispiel. Der beanspruchte Arbeitsspeicher liegt dabei zum Teil deutlich unter 100 MB.

[3] https://es-static.fbk.eu/tools/nuxmv/

5.3.4 IC3 in der Referenzimplementierung

Der IC3 in der Referenzimplementierung[4] stammt von Aron R. Bradly. Auch für den IC3ref mussten die Modelle in das AIG-Format konvertiert werden.
Die Ergebnisse der Testläufe mit IC3 in der Referenzimplementierung von Aaron R. Bradley (IC3ref) sind in Tabelle A.2 zusammengefasst. Beim Test der Verfügbarkeitsspezifikationen führt der IC3ref in einen Segmentation Fault, da er lediglich Invarienten überprüfen kann. Für alle anderen Probleme kommt er zu einem korrekten Ergebnis, kann jedoch bei einer Verletzung kein Gegenbeispiel liefern. Die Verifikation der korrekten Modelle erfolgt in weniger als 30 min. Auch hier zeigt sich wieder, dass es sinnvoll ist, die Invarianten gemeinsam zu prüfen. Mit dieser Methode ist das Bahnhofsstellwerk in 9 min 24 sek verifiziert. Der Arbeitsspeicherverbrauch bleibt zumeist unter 150 MB, lediglich die Verifikation des Flankenschutzes benötigt mehr. Die eingebauten Fehler werden durch den IC3ref in unter 2 min gefunden. Den Fehler beim Flankenschutz zu finden braucht jedoch auch hier länger (13 min 24 sek). Auch bei der Betrachtung des Arbeitsspeicher-Verbrauches hebt sich der fehlerhafte Flankenschutz ab, ansonsten benötigt der IC3ref weniger als 50 MB RAM um die Fehler zu finden.

5.4 Evaluierung

Im Rahmen der Arbeit konnten Pattern zur Modellierung von Komponenten der Schienenverkehrstechnik mit dem Ziel der Verifikation von Stellwerkslogiken entwickelt werden. Ähnlich einem Baukastensystem wurde daraus in einer Fallstudie das Stellwerk des Bahnhofes Braine l'Alleud mit den entsprechenden Gleisanlagen und Signalen modelliert sowie verifiziert. Abstriche mussten im Bereich des Flankenschutzes gemacht werden. Hier stellte sich im Laufe der Arbeit heraus, dass der Gleisplan nicht auf allen Durchfahrtsgleisen einen Flankenschutz mit Schutzweichen zulässt. Daher konnte der Flankenschutz nur für die Fahrstraßen CXC → T 101, DC → T 092, JXC → T 012 und KC → T 101 gezeigt werden.
Mithilfe der aufgestellten Invarianten konnten versuchsweise in die Stellwerkslogik eingefügte Fehler zuverlässig erkannt werden. Die Validierung des Modells gestaltete sich deutlich schwieriger. Aufgrund der Spezifikation von Pfadeigenschaften (Pfadformeln) in LTL benötigt die Validierung des Modells über zwei Wochen. Mit einem Bounded Model-Checking konnte gezeigt werden, dass in allen Pfaden, die ab dem Startzustand maximal 20 Zustände lang sind, kein Verhalten auftritt, das gegen die Livness-Spezifikationen verstößt. Dieses Ergebnis darf nicht als Validierung des Modells verstanden werden, legt jedoch die Vermutung nahe, dass das Modell das Verhalten des originalen Systems abbildet. Ein Teil der Liveness-Spezifikationen konnten zudem in Invarianten überführt werden. Dadurch war es möglich, in ca $8\frac{1}{2}$ min zumindest zu zeigen, dass kein Zug im Gleisfeld des Bahnhofes stehen bleibt, weil eine Verbindung zwischen zwei Gleisen vergessen wurde zu modellieren.
Bei der Verifikation mit den Model-Checkern IIMC, NuXMV, AIGBMC und IC3ref zeigt sich, dass sich Bounded Model-Checking (BMC) deutlich besser eignet, um Fehler zu finden, wohingegen IC3 die Korrektheit bezüglich der Spezifikation zeigen kann. Dies ist eine logische Folge aus den Ansätzen der beiden Algorithmen.

[4] https://github.com/arbrad/IC3ref

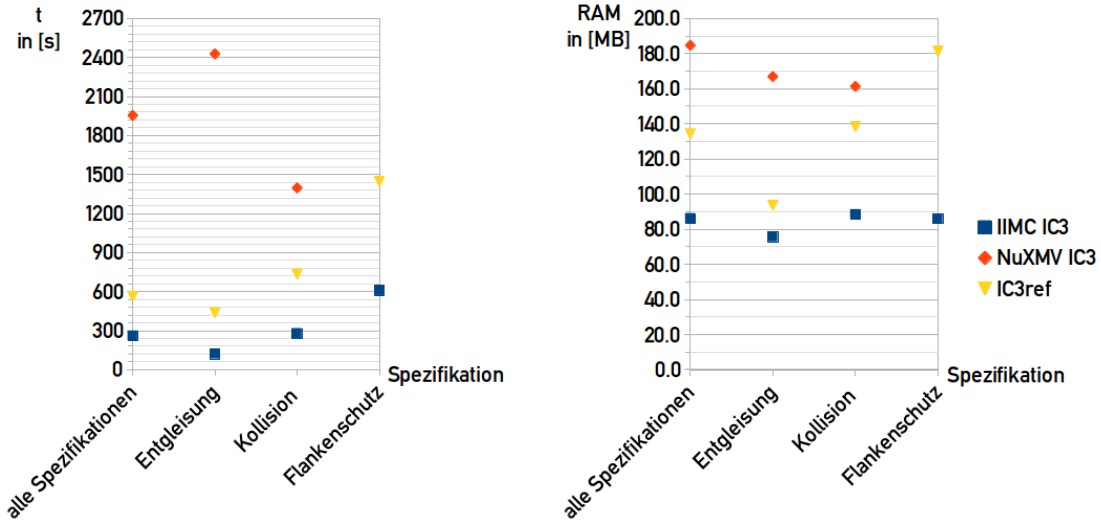


Abbildung 5.2: Laufzeit (links) und Arbeitsspeicher Verbrauch (rechts) von IIMC IC3, NuXMV IC3 und IC3ref bei der Verifikation des korrekten Modells. Verifiziert wurden alle Sicherheitsspezifikationen zusammen, sowie Kollision, Entgleisung und Flankenschutz jeweils einzeln.

So sucht BMC in einem inkrementell wachsenden (zum Teil beschränkten) Umkreis um den Startzustand nach Zuständen, die die Spezifikation verletzen. Bei korrekten Modellen ist es für Bounded Model-Checking jedoch unmöglich, dies zu zeigen. In manchen Modellen mit sehr komplexen Fehlern (Fehler mit sehr langen Pfaden) lassen sich diese auch mit BMC nicht finden (z. B. bei Flankenschutzverletzungen). IC3 hingegen baut eine Beschreibung auf, die mindestens die erreichbaren Zustände einschließt und zeigt, dass sich daraus die Spezifikation ableiten lässt.
Bei der Verifikation der in Invarianten beschriebenen Sicherheitsspezifikationen hat es sich als vorteilhaft gezeigt, alle Invarianten zusammen zu prüfen. Dabei liegen die Rechenzeit und der Speicherverbrauch zum Teil sogar unter den Werten der separaten Verifikationen.
Die in Abschnitt 5.3 entstandenen Ergebnisse zeigen, dass IIMC die zeitlich effizienteste Software unter den vier Getesteten ist, um das Bahnhofsstellwerk zu verifizieren, aber auch um Fehler zu finden. Die parallele Ausführung mehrerer Algorithmen kosten jedoch Arbeitsspeicher und beansprucht mehr CPU-Kerne. In Abbildung 5.2, Abbildung 5.3 und Tabelle A.3 ist der Arbeitsspeicherverbrauch nicht für die gesamte parallele Ausführung, sondern für einen Algorithmus angegeben. Hier zeigt sich, dass der IC3 des IIMC bei korrekten Modellen deutlich speichersparsamer arbeitet als die anderen Algorithmen. Für die parallele Ausführung von mehreren Algorithmen genügt es, sich auf BMC und IC3 zu beschränken.
Der IC3 in der Referenzimplementierung von Aron R. Bradly (IC3ref) liegt bei den Anforderungen an den Arbeitsspeicher im Mittelfeld. Für den Fall einer Spezifikationsverletzung kann er keinen Pfad zu einem Gegenbeispiel generieren und ist so für die Fehlersuche ungeeignet. Außerdem braucht der IC3ref am längsten bei der Fehlersuche, zeigt aber ansonsten ein ähnliches Zeitverhalten, wie die IC3-Implementierung von IIMC.
Die IC3-Implementierung von NuXMV schneidet im Hinblick auf den Speicherver-

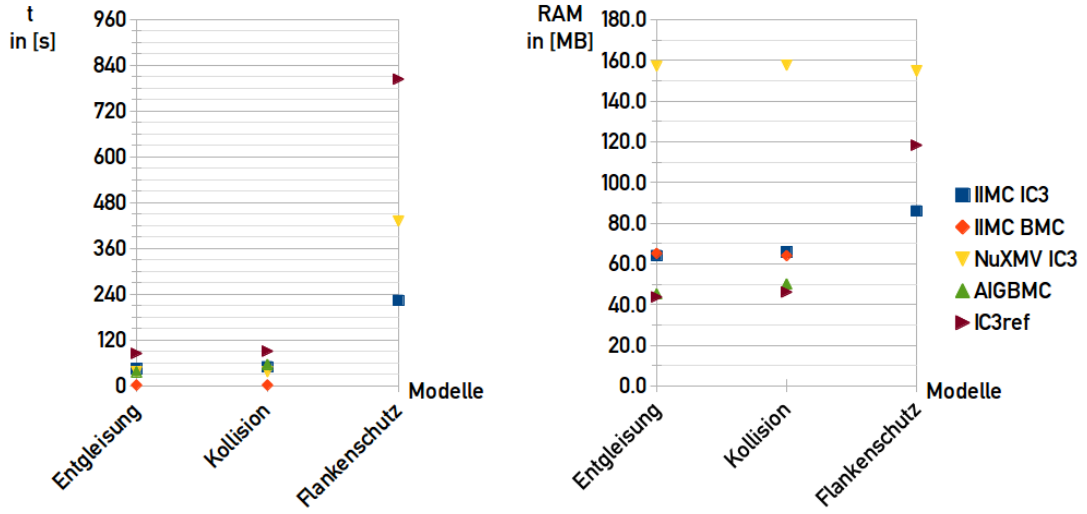


Abbildung 5.3: Laufzeit (links) und Arbeitsspeicher Verbrauch (rechts) von IIMC (IC3 und BMC), NuXMV IC3, AIGBMC und IC3ref bei der Verifikation aller Sicherheitsspezifikationen mit fehlerhaften Modellen. Die BMC-Algorithmen (IIMC BMC und AIGBMC) konnte die Verletzung des Flankenschutzes nicht feststellen. Die Fehler sind aus Tabelle 5.2. Die Entgleisung entsteht aus der Situation in Zeile 2, zur Kollision führt der Fehler aus Zeile 3 und die Flankenschutzverletzung ist in Zeile 10 zu sehen.

brauch am schlechtesten ab. Für die Verifikation des korrekten Modells benötigt es außerdem die längste Rechenzeit von über 30 min. Bei einfacheren Fehlern ist er jedoch nicht schlechter als andere IC3-Implementierungen. Vorteilhaft ist, dass das Eingabeformat von NuXMV sehr nah an VECS/SAML ist. Es liegt lediglich eine Übersetzung dazwischen. Dadurch sind die Ergebnisse, die NuXMV bei fehlerhaften Modellen liefert, sehr aussagekräftig und müssen nicht zeitaufwendig interpretiert werden.
AIGBMC hat sich als gänzlich ungeeignet gezeigt. Auch bei fehlerhaften Modellen kann der Model-Checker aus dem AIGER-Paket nicht immer eine eindeutige Aussage treffen.
Zum Vergleich wurde zusätzlich ein BDD basierter Lösungsansatz zur Verifikation der Sicherheitsspezifikationen mit NuXMV gestartet. Dieser wurde nach über 2 Wochen ergebnislos abgebrochen. Zu diesem Zeitpunkt benötigte NuXMV bereits 32 GB RAM um die Transitionsregeln im Arbeitsspeicher zu halten.
Abschließend ist anzumerken, dass mit dem templatebasierten Modellierungsansatz eine durchaus praktikable Möglichkeit zur Verifikation von Stellwerken erarbeitet wurde. Die Verifikation mit IIMC bietet durch die Parallelisierung von IC3 und BMC eine gute Lösung, um entweder Fehler zu finden oder die Korrektheit zu zeigen. Lediglich die gelieferten Gegenbeispiele fordern noch ein hohes Maß an fachlicher Interpretation.

6. Verwandte Arbeiten

Viele Arbeitsgruppen haben sich bisher mit Themengebieten um die Verifikation und Validierung von computergestützter Stellwerkstechnik auseinandergesetzt. Sie verfolgen dabei unterschiedliche Ansätze und auch Absichten.
In [RMS+12] beschreiben die Autoren eine Methode, computerbasierte Stellwerkstechnik mit Hilfe von CSP||B, einer Kombination aus ereignisorientiertem und zustandsorientiertem Modellierungsansatz, zu modellieren. Anschließend suchen sie nach Deadlocks in dem modellierten System. Abschließend wird überprüft, ob vorher modellierte Fehlerevents ausgelöst werden können. Als Model-Checking-Tool wird ProB verwendet.
Kirsten Winter modelliert im Rahmen von [Win02] Stellwerke in CSP. Zur Verifikation nutz sie den Model-Checker FDR. Dazu modelliert sie explizit Züge, die die Gleise befahren. Dabei wird spezifiziert, dass Züge nicht kollidieren und dass Züge nicht durch eine unverriegelte Weiche entgleisen.
Anne Elisabeth Haxthausen und Jan Peleska beschreiben in [HP15] einen Verifikationsansatz für Stellwerkstechnik mit einem induktivem Bounded Model Checking (k-induction). Da es bei diesem Verfahren zu false-negatives kommen kann, stellen sie zusätzliche Invarianten auf. Mit diesen können die Autoren unerreichbare Zustände im induktiven Teil des Verfahrens ausschließen. Anschließend setzen sie ihr Verfahren im Rahmen einer Fallstudie um. Bei der Modellierung trennen sie zwischen realem Objekt im Bahnhof und virtuellem Objekt im Stellwerkssystem. Dadurch ergibt sich, dass das Stellwerk in der Ausführung zum Teil einen Schritt weiter ist und die realen Komponenten ihren Zustand anpassen müssen.
In [KMS09] stellen die Autoren ein Verfahren vor, in dem sie Stellwerkssoftware verifizieren, die aus Kontaktplänen (Ledder Logic / Ledder Diagram) erstellt wurde. Der Verifikationsansatz beruht dabei auf den, durch den Kontaktplan gegebenen, Zustandsübergängen. Auch sie nutzen Invarianten, um unerreichbare Zustände auszuschließen. Bei der Verifikation beschränken sie sich jedoch auf die Weichenlage und -verriegelung.
Phillip James und Markus Roggenbach generieren ihre Stellwerksmodelle in [JR10] ebenfalls aus Kontaktplänen. Sie berechnen die Abhängikeiten der Sicherheitsspezifikationen von den Variablen der Kontaktpläne mit dem Ziel, Variablen, die keinen

Einfluss auf die aktuell zu prüfende Spezifikation haben, herrauszukürzen.
Über die Integration von Model Checking in bestehende Softwareentwicklungsprozesse bei Ansaldo berichten die Autoren um Alessandro Cimatti in [CGM+98]. Sie beschreiben den strukturellen Aufbau der Stellwerkssoftware und ihre Modellierung mit Promelia als Modellierungssprache für verteilte, kommunizierende Prozesse. Bei der Verifikation mit Spin als Model-Checker beschränken sie sich auf sehr konkrete Spezifikationen, die das Verhalten in bestimmten Situationen beschreiben.
Arne Borälv fasst in [Bor98] die Anwendung formaler Methoden bei ADtranz zusammen. Zur Verifikation wird hier die kommerzielle Software NP-Tools verwendet.
In [ZKA12] beschäftigen sich Nazir Ahmad Zafar, Sher Afzal Khan und Keijiro Araki mit der Erweiterung von Verifikationsansätzen für Stellwerkstechnik bei fixen Blockabständen hin zu Stellwerkstechnik für wandernde Blockabstände. Neben einfachen Stellwerken, ähnlich Braine l'Alleud, verifizieren sie auch Stellwerke, die Kreuzungen und Bahnübergänge kontrollieren.
Quentin Cappart und Pierre Schaus entwickeln einen speziellen Algorithmus zur Verifikation von Kollisionsfreiheit in Stellwerkstechnik [CS16]. Sie testen dazu systematisch jedes Tupel aus zwei Routen. Darüber hinaus verifizieren sie, ob Züge auf den zugewiesenen Routen das korrekte Zielgleis erreichen. Mit Simon Busard hatten sie bereits zuvor in [BCL+15] eine optimierte Form des NuSMV zur Verifikation kleinerer Stellwerke genutzt.
Die Entwicklung spezialisierter Model-Checking-Verfahren für die Verifikation großer Stellwerke empfiehlt auch das Team um Alessio Ferrari in [FMGF10]. Sie hatten zuvor verschiedene Verschlusspläne mit NuSMV und Spin verifiziert, scheiterten jedoch bei Stellwerken mit vielen Gleiselementen an der Größe des Zustandsraumes.
Anne Elisabeth Haxthausen, Hoang Nga Nguyen und Markus Roggenbach haben sich in [HNR16] mit dem Vergleich verschiedener Verifikations- und Modellierungsansätze für Stellwerks-Systeme befasst. Sie betrachten dabei die Ergebnisse der TU Bremen, die im Zusammenhang mit der Verifikation von ETCS Level 2 fähiger Stellwerkstechnik in Dänemark gemacht wurden, sowie die Methoden der Universität Swansea im Bereich des Tool-Supports bei der Optimierung von Schienenverkehrstechnik.
In [BFG+98] stellen die Autoren um Cinzia Bernardeschi ein Tool zur Verifikation von Stellwerken vor. Teil davon ist unter anderem die Möglichkeit den Zustandsraum durch das Ableiten von Teilmodellen zu verkleinern. Sie modellieren dabei mit der Prozessalgebra CSS/MEIJE. Die Spezifikationen werden in ACTL geschrieben.
Bei den gezeigten Arbeiten war die Zielstellung die Optimierung der Laufzeit der Verifikation. Dabei wurden spezielle örtliche Gegebenheiten ausgenutzt, um die Modelle und auch Algorithmen zu optimieren. Dadurch ergibt sich, dass diese Ansätze alle ein sehr hohes Fachwissen im Bereich der formalen Methoden vom Anwender fordern. – Ein Fachwissen, das meist nicht vorhanden ist.
Der in dieser Arbeit vorgestellte Ansatz bietet die Möglichkeit einer (teil-)automatisierten Modellgenerierung und Verifikation. Dadurch wird nicht die optimalste Laufzeit erreicht. Die Hemmschwelle zur Nutzung dieser Herangehensweise ist jedoch deutlich geringer und auch die Erklärung gegenüber den Zertifizierungsstellen ist verständlicher, was zu einem einfacheren Zertifizierungsprozess führt.

7. Zusammenfassung und weiterführende Forschung

Stellwerke sind für einen sicheren Betrieb von Eisenbahnen unerlässlich. Bei Fehlern in der Stellwerkslogik, die das Verhalten des Stellwerkes maßgeblich bestimmt, kann es zu folgenschweren Unfällen kommen, deswegen ist die Verifikation mit formalen Methoden angebracht. Im Gegensatz dazu steht die aktuelle Verfahrensweise, die eine zeitintensive, fehleranfällige, manuelle Prüfung der Stellwerkslogiken vorsieht.
Basis für die formale Verifikation ist ein Modell des Stellwerkes. Für die Modellierung sind jedoch Kenntnisse aus der Domäne der Eisenbahntechnik sowie aus dem Bereich der formalen Methoden nötig. Um diese Hürde für Eisenbahningenieure zu überwinden, wird ein Baukastensystem in der Modellierungssprache SAML entwickelt, dessen Elemente (Basiskomponenten) bereits das gesamte nötige Verhalten implementieren. Die Bausteine müssen nur zu einem Stellwerk zusammengefügt werden. Im Rahmen dieser Arbeit entstehen dabei Basiskomponenten für Gleisabschnitte, Weichen, Signale und Fahrstraßen.
Damit ein Model-Checker entscheiden kann, ob ein Stellwerk sicher funktioniert, werden sicherheitskritische Zustände über Invarianten spezifiziert. Hier wird das Modell auf Kollisionen, Entgleisungen aufgrund von fehlerhafter Weichenansteuerung und Flankenfahrten geprüft. Um von dem Verhalten des Modells auf das reale System schließen zu können, muss das Modell sich hinreichend ähnlich, wie das original System verhalten. Dazu wird das Verhalten des realen Stellwerks mit LTL-Formeln beschrieben. LTL-Formeln sind nötig um Angaben über die Abfolge von Zuständen machen zu können. Im Rahmen dieser Arbeit wird gezeigt, dass die Befahrbarkeit der Gleise dem realen Bahnhof entspricht. Es bleibt also kein Zug im Gleisfeld stehen, weil Gleisenden nicht korrekt zusammengefügt sind. Die Spezifikation der Abfolge zum Stellen einer Fahrstraße scheitert an der Komplexität. Ihre Verifikation musste erfolglos nach $2\frac{1}{2}$ Wochen abgebrochen werden.
Zur Erprobung des Konzeptes wird das Stellwerk des Bahnhofes Braine l'Alleud in Belgien modelliert. Die Verifikation des viergleisigen Bahnhofes mit 32 Fahrstraßen ist in weniger als 5 min möglich. In einem Vergleich verschiedener Model-Checker kann sich IIMC als am besten geeignet positionieren.

In dieser Arbeit wurden grundlegende Basiskomponenten für ein Stellwerk modelliert. Ziel war es, Modellierungsansätze für die Umsetzung solcher Projekte in VECS zu erproben und geeignete Tools für die anschließenden Aufgaben des Model-Checkings zu finden. Im Zusammenhang mit dieser Zielstellung wurden Vereinfachungen angenommen, die in einer weiterführenden Arbeit aufgehoben werden können.
Im Gegensatz zu den Annahmen in dieser Arbeit darf nicht jede Fahrstraße mit Höchstgeschwindigkeit befahren werden. Insbesondere neuere Signalisierungssysteme lassen eine feine Abstufung der für den anschließenden Weichenbereich geltenden herabgesetzten Geschwindigkeit zu. Eine zukünftige Aufgabe ist es, zu verifizieren, dass Signale keine Geschwindigkeiten erlauben, die zu Entgleisungen auf den befahrenen Fahrstraßen führen.
Der gewählte Bahnhof Braine l'Alleud repräsentiert nicht die gesamte Vielfalt an möglichen Konfliktpunkten. In weiterführenden Arbeiten kann die Sicherung von Schienenkreuzungen und Bahnübergängen verifiziert werden. Außerdem wurden Rangierfahrten vernachlässigt, sie können bei weiterführenden Arbeiten Beachtung finden.
In dieser Arbeit wurde der Flankenschutz nur unter Beachtung der Schutzweichen überprüft. Dies stellt jedoch nur eine von vielen Möglichkeiten dar, um Flankenfahrten auszuschließen.
Eine der grundlegenden Annahmen dieser Arbeit ist, dass Züge immer vor „Halt“ zeigenden Signalen anhalten. Die Berichte des Eisenbahn-Bundesamtes der letzten Jahre [Eis15] [Eis16] zeigen jedoch, dass mit 470 überfahrenen Haltsignalen in den Jahren 2014 und 2015 diese Annahme nicht zu halten ist. Damit es trotzdem nicht zu Kollisionen kommt, werden je nach zulässiger Geschwindigkeit und unter Berücksichtigung der örtlichen Gegebenheiten (z. B. Gefälle) hinter einem Signal Durchrutschwege festgelegt. Sie gehören mit zu der Fahrstraße, die an diesem Signal endet, und müssen dementsprechend frei sein, wenn diese Fahrstraße befahren werden soll. Dies zu beachten, ist Teil weiterführender Arbeiten.
Die Rückführung von Gegenbeispielen hin zum eigentlichen Fehler in der Stellwerkslogik ist im aktuellen Zustand der Arbeit nur mit sehr viel Fachwissen und guten Vermutungen möglich. Ziel weiterer Arbeiten soll sein, die durch den Model-Checker generierten Pfade zu den Spezifikationsverletzungen so aufzubereiten, dass für Eisenbahningenieure verständlich ist, welches Fehlverhalten vorliegt und wo diese Fehler entstanden sind.

A. Anhang

Algorithmus	**Verifikationsbedingung**	**Laufzeit**	**RAM**	**Ergebnis**	**Fehler**
BMC	Verfügbarkeit	8 m 1 s	44,6 MB	keine Aussage	Fehlerfrei
BMC	Kollisionsfreiheit	1 m 21 s	55,5 MB	keine Aussage	Fehlerfrei
BMC	Weichenlage und -verriegelung	37 s	44,9 MB	keine Aussage	Fehlerfrei
BMC	Flankenschutz	30 s	35,7 MB	keine Aussage	Fehlerfrei
BMC	alle Sicherheitsspezifikationen	2 m 4 s	69,6 MB	keine Aussage	Fehlerfrei
BMC	alle Sicherheitsspezifikationen	37 s	45,2 MB	Entgleisung auf Gleis T 04C	Fehler siehe Zeile 2
BMC	alle Sicherheitsspezifikationen	56 s	50,1 MB	Kollision auf Gleis T 104	Fehler siehe Zeile 3
BMC	alle Sicherheitsspezifikationen	33 s	45,4 MB	Entgleisung auf Gleis T 01BC	Fehler siehe Zeile 4
BMC	alle Sicherheitsspezifikationen	15 s	29,6 MB	Entgleisung auf Gleis T 03C	Fehler siehe Zeile 5
BMC	alle Sicherheitsspezifikationen	2 m 28 s	74,8 MB	keine Aussage	Fehler siehe Zeile 6
BMC	alle Sicherheitsspezifikationen	14 s	29,7 MB	Entgleisung auf Gleis T 10C	Fehler siehe Zeile 7
BMC	alle Sicherheitsspezifikationen	14 s	31,5 MB	Entgleisung auf Gleis T 10C	Fehler siehe Zeile 8
BMC	alle Sicherheitsspezifikationen	13 s	30,7 MB	Entgleisung auf Gleis T 09C	Fehler siehe Zeile 9
BMC	alle Sicherheitsspezifikationen	2 m 12 s	83,1 MB	keine Aussage	Fehler siehe Zeile 10

Tabelle A.1: Testergebnisse des Bahnhofes Braine l'Alleud mit den Fehlern aus dem Verschlussplan (Tabelle 5.2) bei der Verifikation mit aigbmc. Die Reichweite liget bei 10 Zuständen.

Algorithmus	**Verifikationsbedingung**	**Laufzeit**	**RAM**	**Ergebnis**	**Fehler**
IC3	Verfügbarkeit		3,9 MB	segmentation fault	Fehlerfrei
IC3	Kollisionsfreiheit	12 m 14 s	138,5 MB	verifiziert	Fehlerfrei
IC3	Weichenlage und -verriegelung	7 m 18 s	93,7 MB	verifiziert	Fehlerfrei
IC3	Flankenschutz	24 m 8 s	181,3 MB	verifiziert	Fehlerfrei
IC3	alle Sicherheitsspezifikationen	9 m 24 s	134,5 MB	verifiziert	Fehlerfrei
IC3	alle Sicherheitsspezifikationen	1 m 25 s	43,7 MB	Spezifikationsverletztung	Fehler siehe Zeile 2
IC3	alle Sicherheitsspezifikationen	1 m 31 s	46,1 MB	Spezifikationsverletztung	Fehler siehe Zeile 3
IC3	alle Sicherheitsspezifikationen	1 m 27 s	42,3 MB	Spezifikationsverletztung	Fehler siehe Zeile 4
IC3	alle Sicherheitsspezifikationen	1 m 6 s	41,0 MB	Spezifikationsverletztung	Fehler siehe Zeile 5
IC3	alle Sicherheitsspezifikationen	1 m 56 s	49,4 MB	Spezifikationsverletztung	Fehler siehe Zeile 6
IC3	alle Sicherheitsspezifikationen	1 m 2 s	40,7 MB	Spezifikationsverletztung	Fehler siehe Zeile 7
IC3	alle Sicherheitsspezifikationen	1 m 12 s	41,8 MB	Spezifikationsverletztung	Fehler siehe Zeile 8
IC3	alle Sicherheitsspezifikationen	1 m 3 s	38,7 MB	Spezifikationsverletztung	Fehler siehe Zeile 9
IC3	alle Sicherheitsspezifikationen	13 m 24 s	118,3 MB	Spezifikationsverletztung	Fehler siehe Zeile 10

Tabelle A.2: Testergebnisse des Bahnhofes Braine l'Alleud mit den Fehlern aus dem Verschlussplan (Tabelle 5.2) bei der Verifikation mit IC3 in der Referenzimplementierung von Aaron R. Bradley.

Algorithmus	**Verifikationsbedingung**	**Laufzeit**	**RAM**	**Ergebnis**	**Fehler**
k-live	Verfügbarkeit	>2,5Wochen	12,7 GB	abgebrochen	Fehlerfrei
IC3	Kollisionsfreiheit	4 m 38 s	88,4 MB	verifiziert	Fehlerfrei
IC3	Weichenlage und -verriegelung	2 m 0 s	75,6 MB	verifiziert	Fehlerfrei
IC3	Flankenschutz	10 m 8 s	85,9 MB	verifiziert	Fehlerfrei
IC3	alle Sicherheitsspezifikationen	4 m 20 s	86,1 MB	verifiziert	Fehlerfrei
IC3	alle Sicherheitsspezifikationen	45 s	64,1 MB	Entgleisung auf Gleis T 04C	Fehler siehe Zeile 2
BMC	alle Sicherheitsspezifikationen	2 s	65,1 MB	Entgleisung auf Gleis T 04C	Fehler siehe Zeile 2
IC3	alle Sicherheitsspezifikationen	50 s	65,8 MB	Kollision auf Gleis T 104	Fehler siehe Zeile 3
BMC	alle Sicherheitsspezifikationen	2 s	64,0 MB	Kollision auf Gleis T 104	Fehler siehe Zeile 3
IC3	alle Sicherheitsspezifikationen	56 s	65,0 MB	Entgleisung auf Gleis T 01BC	Fehler siehe Zeile 4
BMC	alle Sicherheitsspezifikationen	2 s	66,5 MB	Entgleisung auf Gleis T 01BC	Fehler siehe Zeile 4
IC3	alle Sicherheitsspezifikationen	39 s	65,0 MB	Entgleisung auf Gleis T 03C	Fehler siehe Zeile 5
BMC	alle Sicherheitsspezifikationen	< 1 s	60,7 MB	Entgleisung auf Gleis T 03C	Fehler siehe Zeile 5
IC3	alle Sicherheitsspezifikationen	1 m 0 s	68,9 MB	Entgleisung auf Gleis T 07AC	Fehler siehe Zeile 6
BMC	alle Sicherheitsspezifikationen	4 s	67,4 MB	Entgleisung auf Gleis T 07AC und Kollision auf T 101	Fehler siehe Zeile 6
IC3	alle Sicherheitsspezifikationen	35 s	64,1 MB	Entgleisung auf Gleis T 10C	Fehler siehe Zeile 7
BMC	alle Sicherheitsspezifikationen	1 s	59,9 MB	Entgleisung auf Gleis T 10C	Fehler siehe Zeile 7
IC3	alle Sicherheitsspezifikationen	38 s	63,2 MB	Entgleisung auf Gleis T 10C	Fehler siehe Zeile 8
BMC	alle Sicherheitsspezifikationen	< 1 s	65,1 MB	Entgleisung auf Gleis T 10C	Fehler siehe Zeile 8
IC3	alle Sicherheitsspezifikationen	20 s	61,7 MB	Entgleisung auf Gleis T 09C	Fehler siehe Zeile 9
BMC	alle Sicherheitsspezifikationen	< 1 s	60,7 MB	Entgleisung auf Gleis T 09C	Fehler siehe Zeile 9
IC3	alle Sicherheitsspezifikationen	3 m 44 s	86,0 MB	Flankenkollision auf Weiche P 08BC möglich	Fehler siehe Zeile 10
BMC	alle Sicherheitsspezifikationen	50 s	122,4 MB	keine Aussage	Fehler siehe Zeile 10

Tabelle A.3: Testergebnisse des Bahnhofes Braine l'Alleud mit den Fehlern aus dem Verschlussplan (Tabelle 5.2) bei der Verifikation mit IIMC.

Algorithmus	Verifikationsbedingung	Laufzeit	RAM	Ergebnis	Fehler
k-live	Verfügbarkeit	>2,5Wochen	567,1 MB	abgebrochen	Fehlerfrei
BMC *	Verfügbarkeit	9 h 30 m 36 s	1,3 GB	keine Gegenbeispiele der Länge ≤ 20	Fehlerfrei
IC3	Kollisionsfreiheit	23 m 17 s	161,3 MB	verifiziert	Fehlerfrei
IC3	Weichenlage und -verriegelung	40 m 27 s	166,9 MB	verifiziert	Fehlerfrei
IC3	Flankenschutz	>2,5Wochen	2,6 GB	abgebrochen	Fehlerfrei
IC3	alle Sicherheitsspezifikationen	32 m 34 s	184,7 MB	verifiziert	Fehlerfrei
IC3	alle Sicherheitsspezifikationen	37 s	157,2 MB	Entgleisung auf Gleis T 04C	Fehler siehe Zeile 2
BMC	alle Sicherheitsspezifikationen	< 1 s	31,0 MB	keine Aussage („induction fails“)	Fehler siehe Zeile 2
IC3	alle Sicherheitsspezifikationen	37 s	157,6 MB	Kollision auf Gleis T 104	Fehler siehe Zeile 3
BMC	alle Sicherheitsspezifikationen	< 1 s	31,3 MB	keine Aussage („induction fails“)	Fehler siehe Zeile 3
IC3	alle Sicherheitsspezifikationen	36 s	144,7 MB	Entgleisung auf Gleis T 01BC	Fehler siehe Zeile 4
BMC	alle Sicherheitsspezifikationen	< 1 s	32,2 MB	keine Aussage („induction fails“)	Fehler siehe Zeile 4
IC3	alle Sicherheitsspezifikationen	34 s	145,3 MB	Entgleisung auf Gleis T 03C	Fehler siehe Zeile 5
BMC	alle Sicherheitsspezifikationen	< 1 s	31,0 MB	keine Aussage („induction fails“)	Fehler siehe Zeile 5
IC3	alle Sicherheitsspezifikationen	54 s	145,6 MB	Entgleisung auf Gleis T 07AC	Fehler siehe Zeile 6
BMC	alle Sicherheitsspezifikationen	< 1 s	31,2 MB	keine Aussage („induction fails“)	Fehler siehe Zeile 6
IC3	alle Sicherheitsspezifikationen	26 s	146,3 MB	Entgleisung auf Gleis T 10C	Fehler siehe Zeile 7
BMC	alle Sicherheitsspezifikationen	< 1 s	32,4 MB	keine Aussage („induction fails“)	Fehler siehe Zeile 7
IC3	alle Sicherheitsspezifikationen	29 s	146,3 MB	Entgleisung auf Gleis T 10C	Fehler siehe Zeile 8
BMC	alle Sicherheitsspezifikationen	< 1 s	30,7 MB	keine Aussage („induction fails“)	Fehler siehe Zeile 8
IC3	alle Sicherheitsspezifikationen	29 s	157,8 MB	Entgleisung auf Gleis T 09C	Fehler siehe Zeile 9
BMC	alle Sicherheitsspezifikationen	< 1 s	30,7 MB	keine Aussage („induction fails“)	Fehler siehe Zeile 9
IC3	alle Sicherheitsspezifikationen	7 m 12 s	154,7 MB	Flankenkollision auf Weiche P 08BC möglich	Fehler siehe Zeile 10
BMC	alle Sicherheitsspezifikationen	< 1 s	32,4 MB	keine Aussage („induction fails“)	Fehler siehe Zeile 10

Tabelle A.4: Testergebnisse des Bahnhofes Braine l'Alleud mit den Fehlern aus dem Verschlussplan (Tabelle 5.2) bei der Verifikation mit NuXMV.

*Bounded-Model-Checking mit einer Reichweite von 20 Zuständen

Literaturverzeichnis

[BCC+07] Robert Brummayer, Alessandro Cimatti, Koen Claessen, Niklas Een, Marc Herbstritt, Hyondeuk Kim, Toni Jussila, Ken McMillan, Alan Mishchenko, Fabio Somenzi, et al. The AIGER And-Inverter Graph (AIG) Format Version 20070427. 2007. (zitiert auf Seite 31)

[BCCZ99] Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu. Symbolic Model Checking without BDDs. In *TACAS*, volume 1579 of *LNCS*, pages 193–207, 1999. (zitiert auf Seite 12)

[BCL+15] Simon Busard, Quentin Cappart, Christophe Limbrée, Charles Pecheur, and Pierre Schaus. Verification of railway interlocking systems. In Jun Pang, Yang Liu, and Sjouke Mauw, editors, *ESSS*, volume 184 of *EPTCS*, pages 19–31, 2015. (zitiert auf Seite 38)

[BFG+98] Cinzia Bernardeschi, Alessandro Fantechi, Stefania Gnesi, Salvatore Larosa, Giorgio Mongardi, and Dario Romano. A Formal Verification Environment for Railway Signaling System Design. *Formal Methods in System Design*, 12(2):139–161, 1998. (zitiert auf Seite 38)

[Bor98] Arne Borälv. Case Study: Formal Verification of a Computerized Railway Interlocking. *Formal Asp. Comput.*, 10(4):338–360, 1998. (zitiert auf Seite 38)

[Bra11] Aaron R. Bradley. SAT-Based Model Checking without Unrolling. In Ranjit Jhala and David A. Schmidt, editors, *VMCAI*, volume 6538 of *Lecture Notes in Computer Science*, pages 70–87. Springer, 2011. (zitiert auf Seite 12)

[Bun16] Bundesministeriums der Justiz und für Verbraucherschutz. Eisenbahn-Bau- und Betriebsordnung vom 8. Mai 1967 (BGBl. 1967 II S. 1563), die zuletzt durch Artikel 1 der Verordnung vom 10. Oktober 2016 (BGBl. I S. 2242) geändert worden ist. October 2016. (zitiert auf Seite 23)

[CGM+98] Alessandro Cimatti, Fausto Giunchiglia, Giorgio Mongardi, Dario Romano, Fernando Torielli, and Paolo Traverso. Formal Verification of a Railway Interlocking System using Model Checking. *Formal Asp. Comput.*, 10(4):361–380, 1998. (zitiert auf Seite 38)

[CGP01] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model checking*. MIT Press, 2001. (zitiert auf Seite 11)

[CMCHG96] Edmund M. Clarke, Kenneth L. McMillan, Sérgio Vale Aguiar Campos, and Vasiliki Hartonas-Garmhausen. Symbolic Model Checking. In Rajeev Alur and Thomas A. Henzinger, editors, *CAV*, volume 1102 of *Lecture Notes in Computer Science*, pages 419–427. Springer, 1996. (zitiert auf Seite 31)

[CS16] Quentin Cappart and Pierre Schaus. A Dedicated Algorithm for Verification of Interlocking Systems. In Amund Skavhaug, Jérémie Guiochet, and Friedemann Bitsch, editors, *SAFECOMP*, volume 9922 of *Lecture Notes in Computer Science*, pages 76–87. Springer, 2016. (zitiert auf Seite 29 und 38)

[Eis15] Eisenbahn-Bundesamt. *Sicherheitsbericht des Eisenbahn-Bundesamts 2014*. September 2015. (zitiert auf Seite 40)

[Eis16] Eisenbahn-Bundesamt. *Sicherheitsbericht des Eisenbahn-Bundesamts 2015*. September 2016. (zitiert auf Seite 40)

[FMGF10] Alessio Ferrari, Gianluca Magnani, Daniele Grasso, and Alessandro Fantechi. Model Checking Interlocking Control Tables. In Eckehard Schnieder and Géza Tarnai, editors, *FORMS/FORMAT*, pages 107–115. Springer, 2010. (zitiert auf Seite 38)

[GO10] Matthias Güdemann and Frank Ortmeier. A Framework for Qualitative and Quantitative Formal Model-Based Safety Analysis. In *HASE*, pages 132–141. IEEE Computer Society, 2010. (zitiert auf Seite 8, 9 und 31)

[Güd11] Matthias Güdemann. *Qualitative and quantitative formal model-based safety analysis: push the safety button*. PhD thesis, Uni Magdeburg, 2011. (zitiert auf Seite 8)

[HNR16] Anne Elisabeth Haxthausen, Hoang Nga Nguyen, and Markus Roggenbach. Comparing Formal Verification Approaches of Interlocking Systems. In Thierry Lecomte, Ralf Pinger, and Alexander Romanovsky, editors, *RSSRail*, volume 9707 of *Lecture Notes in Computer Science*, pages 160–177. Springer, 2016. (zitiert auf Seite 38)

[HP15] Anne Elisabeth Haxthausen and Jan Peleska. Model Checking and Model-Based Testing in the Railway Domain. In Rolf Drechsler and Ulrich Kühne, editors, *SyDe Summer School*, pages 82–121. Springer, 2015. (zitiert auf Seite 37)

[JR10] Phillip James and Markus Roggenbach. Automatically Verifying Railway Interlockings using SAT-based Model Checking. *ECEASST*, 35, 2010. (zitiert auf Seite 37)

[KMS09] Karim Kanso, Faron Moller, and Anton Setzer. Automated Verification of Signalling Principles in Railway Interlocking Systems. *Electr. Notes Theor. Comput. Sci.*, 250(2):19–31, 2009. (zitiert auf Seite 37)

[Pac11] Jörn Pachel. *Systemtechnik des Schienenverkehrs*, volume 6. Vieweg + Teubner Verlag | Springer Fachmedien Wiesbaden GmbH, 2011. (zitiert auf Seite 6 und 7)

[RMS+12] Markus Roggenbach, Faron Moller, Steve Schneider, Helen Treharne, and Hoang Nga Nguyen. Railway modelling in CSP||B: the double junction case study. *ECEASST*, 53, 2012. (zitiert auf Seite 37)

[Win02] K. Winter. Model Checking Railway Interlocking Systems. In Michael J. Oudshoorn, editor, *ACSC*, volume 4 of *CRPIT*, pages 303–310. Australian Computer Society, 2002. (zitiert auf Seite 37)

[ZKA12] Nazir Ahmad Zafar, Sher Afzal Khan, and Keijiro Araki. Towards the safety properties of moving block railway interlocking system. *International Journal of Innovative Computing, Information and Control*, 8(8):5677–5690, August 2012. (zitiert auf Seite 38)

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Magdeburg, den 31.03.2017