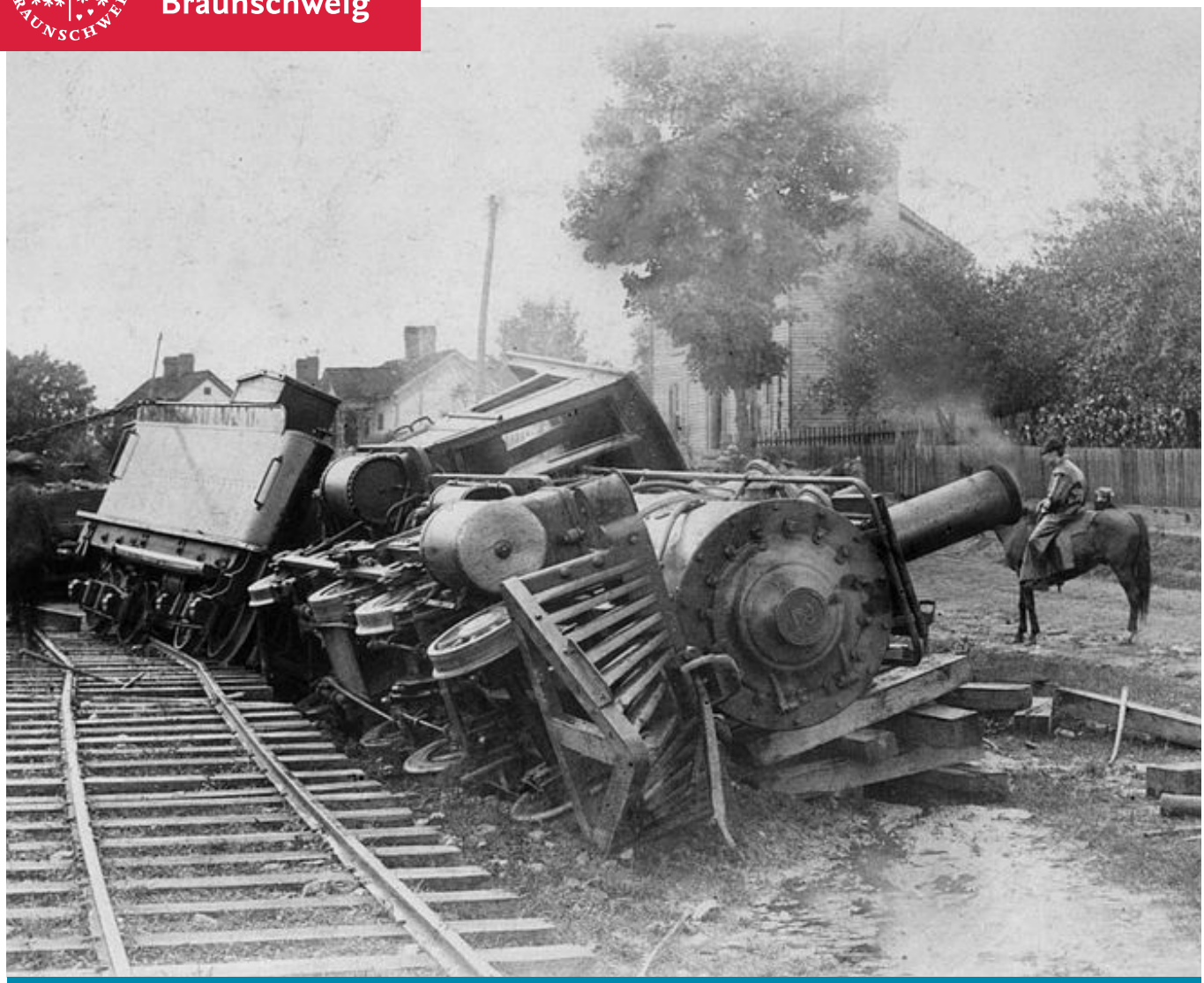
Technische Universität Braunschweig

Institut für Theoretische Informatik



Bahnübergangsfallstudie

Verifikationsbericht

Henning Günther, Ramin Hedayati & Axel Zechner

13. März 2012

Henning Günther <guenther@iti.cs.tu-bs.de>
Institut für Theoretische Informatik
Mühlenpfordtstr. 22-23
38106 Braunschweig

Ramin Hedayati <Ramin.Hedayati@ics-ag.de>
Axel Zechner <Axel.Zechner@ics-ag.de>
ICS AG
Alt-Moabit 73
10555 Berlin

Dieser Bericht entstand im Rahmen des Projektes VerSyKo (http://tu-braunschweig.de/iti/research/versyko). Das Titelbild stammt vom Benutzer «Jordangotcha» von Wikimedia Commons.

Inhaltsverzeichnis

1 Einleitung

In diesem Dokument werden die Ergebnisse der Fallstudie „Bahnübergang“ festgehalten, die in Zusammenarbeit mit dem Institut für Theoretische Informatik und der ICS AG entstanden sind. Der Zweck der Fallstudie ist die Evaluation der im VerSyKo-Projekt erstellten Werkzeuge zur formalen Verfikation. Die verwendeten Modelle sollen hierbei eine für Industrie-Projekte realistische Komplexität vorweisen. Zu diesem Zweck wurde nach den folgenden Schritten vorgegangen:

Anforderungsanalyse In diesem Schritt wurde ein Dokument erstellt, dass die Anforderungen an die zu erstellenden Modelle beschreibt[SZH11]. Aus diesem lassen sich sowohl das Design der Komponenten wie auch globale Verifikationsziele extrahieren.

Design des Systems Aus dem Anforderungsdokument wurden danach SCADE-Modelle für jede im Dokument beschriebene Komponente erstellt. Außerdem wurden erste Tests des Systems mit einfachen Simulationsläufen durchgeführt.

Bestimmung von Verifikationszielen Aus dem Anforderungsdokument wurden nun exemplarische Anforderungen entnommen und in die GTL Sprache übersetzt.

Entwurf von Kontrakten Wie im Konzeptpapier beschrieben, wurden in diesem Schritt Kontrakte für jede im System verwendete Komponente erstellt[Ver11].

Verifikation Das „GTL“-Werkzeug wurde verwendet um die Verifikation des Gesamtsystems durchzuführen. Hierbei wurden verschiedene Modi verwendet.

2 Gesamtsystem

In diesem Kapitel wird das aus den im folgenden Kapitel beschriebenen Komponenten zusammen gesetzte Bahnübergangssystem erklärt.

2.1. Zusammensetzung

Abbildung 2.1 zeigt das Zusammenspiel aller Komponenten im Gesamtsystem. Die Eingänge der Komponenten sind auf ihrer linken Seite, die Ausgänge auf der rechten. Aus Übersichtsgrüden sind hier nur die verbundenen Ein- und Ausgaben dargestellt.

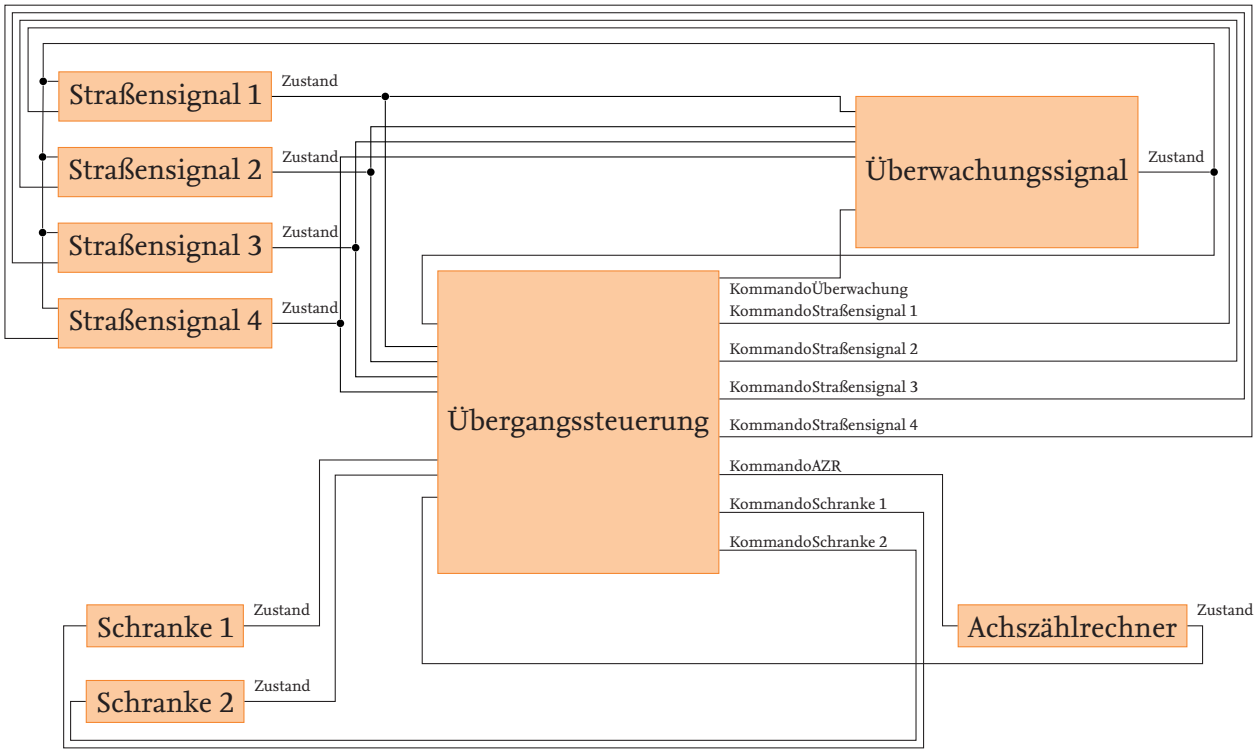


Abbildung 2.1.: Kommunikationspfade der Komponenten

3 Komponenten

Hier werden alle Komponenten, aus denen das Gesamtsystem zusammen gesetzt ist dargestellt und die Anforderungen an ihr Verhalten erläutert. Diese Anforderungen werden in der GTL-Spezifikation in Form von Kontrakten realisiert.

3.1. Übergangssteuerung

Da die Übergangssteuerung die zentrale Kommunikationseinheit des Systems ist, müssen die meisten Anforderungen an ihr Verhalten gestellt werden.

3.1.1. Befehlssende-Verhalten

Die Übergangssteuerung sendet immer erst den Befehl zur Freigabe des Abschnitts für Züge an das Überwachungssignal wenn zuvor

1. Die Straßensignale den Befehl bekamen, den Abschnitt für Autos zu sperren.
2. Die Straßensignale alle zurück meldeten, dass sie nun Rot anzeigen.

Genauso wird der Befehl zur Freigabe des Abschnitts für Autos erst dann an die Straßensignale gesendet, wenn zuvor

1. Das Überwachungssignal den Befehl zur Sperrung des Abschnitts für Züge erhalten hat.
2. Das Überwachungssignal bestätigt hat, dass es nun Rot anzeigt.

Diese Eigenschaft ist informell als Automat in Abbildung 3.1 dargestellt. In GTL wird diese Eigenschaft formuliert als:

```
1   automaton {
2     init final state Open {
3       KommandoUeberwachung!='UeberwachungSichern;
4       transition Open;
5       transition[KommandoStrassenSignal[0]='StrasseSichern] Closing;
6     }
7     final state Closing {
8       KommandoUeberwachung!='UeberwachungSichern;
9       transition[ZustandStrassenSignal=['StrasseRotAnGelbAus,
10                                        'StrasseRotAnGelbAus,
11                                        'StrasseRotAnGelbAus,
12                                        'StrasseRotAnGelbAus]] Closed;
13      transition[ZustandStrassenSignal!=['StrasseRotAnGelbAus,
14                                         'StrasseRotAnGelbAus,
15                                         'StrasseRotAnGelbAus,
```

```
16                                                    'StrasseRotAnGelbAus]] Closing;
17      }
18      final state Closed {
19        KommandoStrassenSignal[0]!='StrasseEntsichern;
20        transition[KommandoUeberwachung='UeberwachungEntsichern] Opening;
21        transition[KommandoUeberwachung!='UeberwachungEntsichern] Closed;
22      }
23      final state Opening {
24        KommandoStrassenSignal[0]!='StrasseEntsichern;
25        transition Opening;
26        transition[ZustandUeberwachung='UeberwachungAus] Open;
27      }
28    };
```

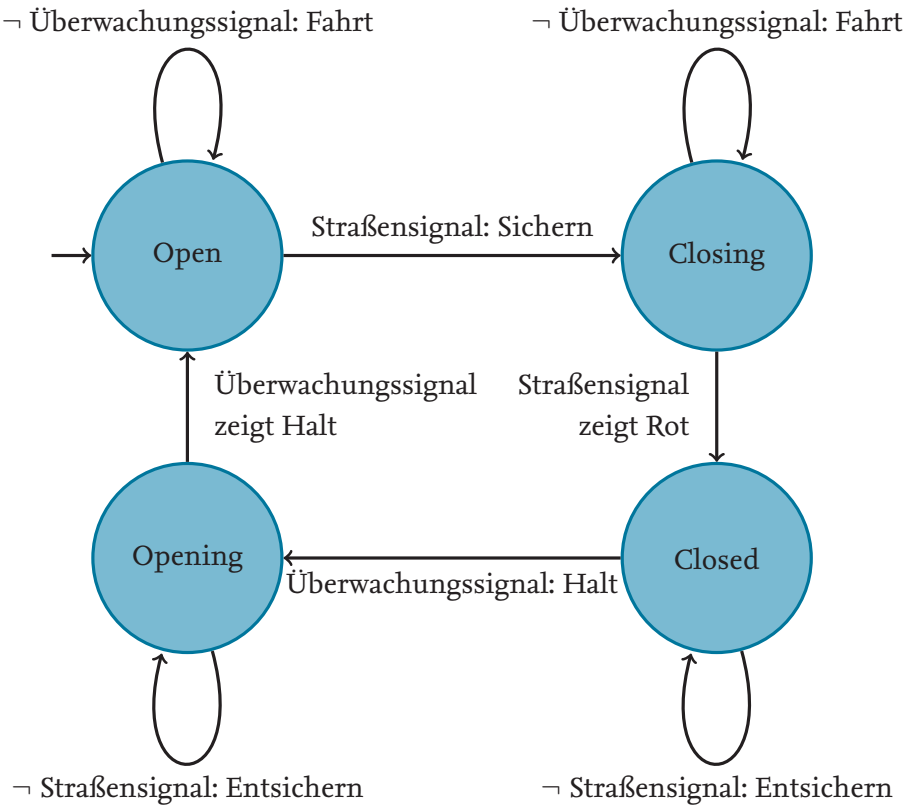


Abbildung 3.1.: Automat für Befehlssende-Verhalten

3.1.2. Störungsverhalten

Es muss sichergestellt werden, dass der Störungsbefehl nur dann an die Straßensignale versendet wird, wenn eine Komponente im System eine Störung meldet. Außerdem muss sichergestellt sein, dass eine Entstörung der Komponente wieder zum Start dieses Verhalten zurück führt. Dieses Verhalten wird in Abbildung 3.2 gezeigt. Die um einen Takt verzögerte Reaktion der Komponente ist durch eine „weak“-Transition im SCADE-Modell bedingt.

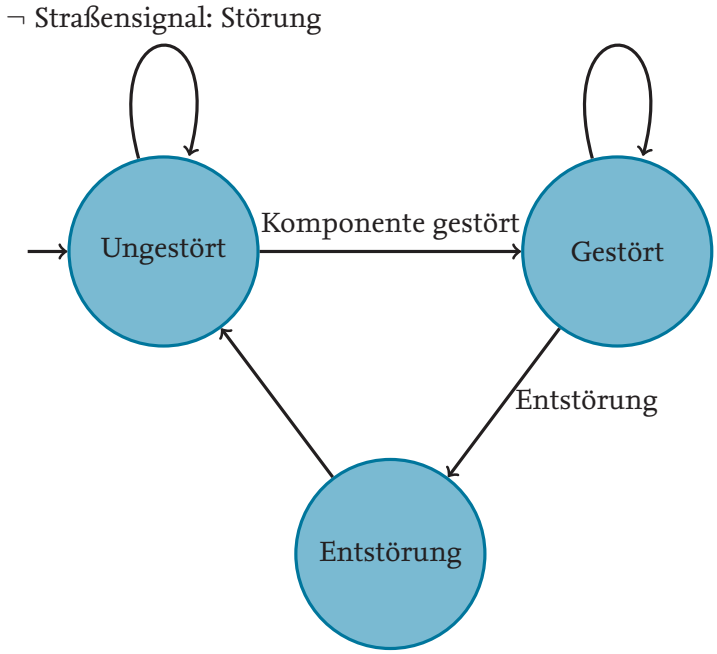


Abbildung 3.2.: Automat für Störungsverhalten

In GTL wird dieses Verhalten durch den folgenden Automaten dargestellt:

```
1  automaton {
2    init final state Ungestoert {
3      KommandoStrassenSignal[0]!='StrasseStoerung;
4      KommandoStrassenSignal[1]!='StrasseStoerung;
5      KommandoStrassenSignal[2]!='StrasseStoerung;
6      KommandoStrassenSignal[3]!='StrasseStoerung;
7      transition Ungestoert;
8      transition[ZustandAZR='AZRGestoert or
9                 ZustandStrassenSignal[0]='StrasseGestoert or
10                ZustandStrassenSignal[1]='StrasseGestoert or
11                ZustandStrassenSignal[2]='StrasseGestoert or
12                ZustandStrassenSignal[3]='StrasseGestoert or
13                ZustandUeberwachung='UeberwachungGestoert] Gestoert;
14   }
15   final state Gestoert {
16     transition[EntstoerungVomServicepersonal] Entstoerung;
17     transition Gestoert;
18   }
19   final state Entstoerung {
20     transition Ungestoert;
21   }
22 };
```

3.1.3. Initialisierungsverhalten

Die Überwachungssteuerung darf den Übergang erst dann für Autos sperren, wenn zuvor die Ampeln initialisiert wurden. Hierfür wird mit dem Kontrakt das Senden des „Sichern“ Befehls verhindert, bis den Ampeln der Initialisierungsbefehl gesendet wurde. Abbildung 3.3 zeigt den einfachen

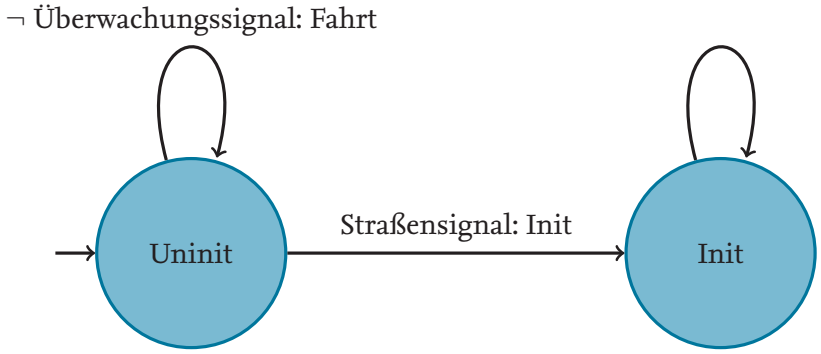


Abbildung 3.3.: Automat für Initialisierungsverhalten

Automaten, der dieses Verhalten realisiert. Die entsprechende Realisierung in GTL lautet wie folgt:

```
1   automaton {
2     init final state Uninit {
3       KommandoUeberwachung!='UeberwachungSichern;
4       transition[KommandoStrassenSignal[0]='StrasseInit] Init;
5       transition[KommandoStrassenSignal[0]!='StrasseInit] Uninit;
6     }
7     final state Init {
8       transition Init;
9     }
10  };
```

3.2. Straßensignal

Da das Straßensignal eine wesentliche Rolle in der Verifikation des Gesamtsystems spielt, ist es erforderlich, diese Komponente sehr genau zu modellieren.

3.2.1. Ausgabekonsistenz

Die Komponente sendet ihren internen Zustand sowohl an die Übergangssteuerung wie auch an das Überwachungssignal. Um das gewünschte Systemverhalten zu verifizieren, muss sichergestellt sein, dass immer der gleiche Zustand an beide Komponenten gesendet wird. Diese Eigenschaft lässt sich leicht mithilfe einer LTL-Formel ausdrücken:

```
1   always (Zustand0 = Zustand1);
```

3.2.2. Schaltverhalten

Das Schaltverhalten wird soweit modelliert, dass das Ampel-typische Verhalten zum Vorschein kommt: Auf das Signal „Sichern“ zeigt die Ampel gelb, eine undefinierte Zeit danach rot und auf

den Befehl „Entsichern“ und einer undefinierten Zeitspanne wieder grün. Das genaue Verhalten wird in Abbildung 3.4 gezeigt. Die Invarianten in jedem Zustand sind in Tabelle 3.2 aufgeführt. In

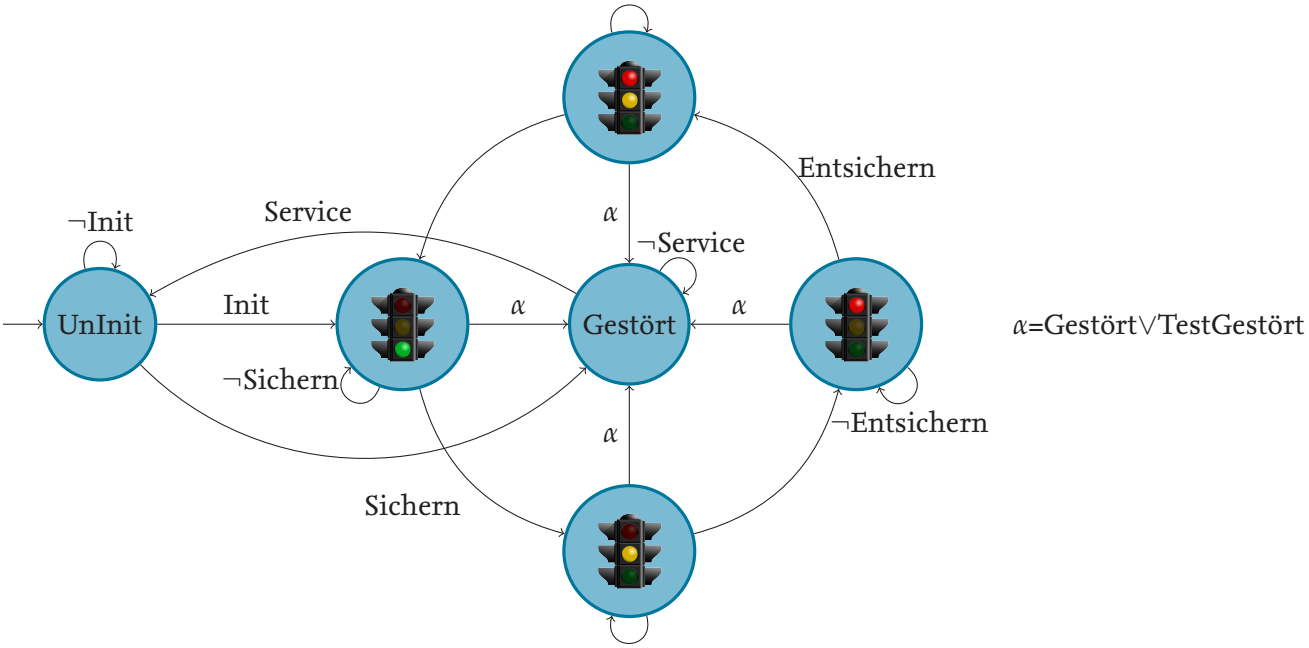


Abbildung 3.4.: Automat für Straßensignal-Verhalten

GTL ist dieser Automat wie folgt kodiert:

```
1   automaton {
2     init final state Uninit {
3       UmgebungKfz = 'KfzRotAusGelbAus;
4       Zustand1 = 'StrasseTestOB or Zustand1 = 'StrasseKeinSignal;
5       transition[Kommando='StrasseInit] Gruen;
6       transition[Kommando!='StrasseInit] Uninit;
7       transition[Gestoert] Gestoert;
8       transition[GestoertTest] Gestoert;
9       transition[Kommando='StrasseStoerung] Gestoert;
10    }
11    final state WaitGruen {
12      UmgebungKfz = 'KfzRotAnGelbAn;
13      Zustand1 = 'StrasseTestOB or
14        Zustand1 = 'StrasseKeinSignal or
15        Zustand1 = 'StrasseRotAnGelbAn;
16      transition WaitGruen;
17      transition Gruen;
18      transition[Gestoert] Gestoert;
19      transition[GestoertTest] Gestoert;
20      transition[Kommando='StrasseStoerung] Gestoert;
```

Zustand	Invarianten
Uninit	UmgebungKfz = KfzRotAusGelbAus Zustand1 = StrasseTestOB ∨ Zustand1 = StrasseKeinSignal
Gestört	-
 (Grün)	UmgebungKfz = KfzRotAusGelbAus Zustand1 = StrasseTestOB ∨ Zustand1 = StrasseKeinSignal ∨ Zustand1 = StrasseRotAusGelbAus
 (WaitRot)	UmgebungKfz = KfzRotAusGelbAn Zustand1 = StrasseTestOB ∨ Zustand1 = StrasseKeinSignal ∨ Zustand1 = StrasseRotAusGelbAn
 (Rot)	UmgebungKfz = KfzRotAnGelbAus Zustand1 = StrasseTestOB ∨ Zustand1 = StrasseKeinSignal ∨ Zustand1 = StrasseRotAnGelbAus
 (WaitGrün)	UmgebungKfz = KfzRotAnGelbAn Zustand1 = StrasseTestOB ∨ Zustand1 = StrasseKeinSignal ∨ Zustand1 = StrasseRotAnGelbAn

Tabelle 3.2.: Zustandsinvarianten des Straßensignals

```
21      }
22      final state Gruen {
23        UmgebungKfz = 'KfzRotAusGelbAus;
24        Zustand1 = 'StrasseTestOB or
25          Zustand1 = 'StrasseKeinSignal or
26          Zustand1 = 'StrasseRotAusGelbAus;
27        transition[Kommando='StrasseSichern] WaitRot;
28        transition[Kommando!='StrasseSichern] Gruen;
29        transition[Gestoert] Gestoert;
30        transition[GestoertTest] Gestoert;
31        transition[Kommando='StrasseStoerung] Gestoert;
32      }
33      final state WaitRot {
34        UmgebungKfz = 'KfzRotAusGelbAn;
35        Zustand1 = 'StrasseTestOB or
36          Zustand1 = 'StrasseKeinSignal or
37          Zustand1 = 'StrasseRotAusGelbAn;
```

```
38        transition WaitRot;
39        transition Rot;
40        transition[Gestoert] Gestoert;
41        transition[GestoertTest] Gestoert;
42        transition[Kommando='StrasseStoerung] Gestoert;
43      }
44      final state Rot {
45        UmgebungKfz = 'KfzRotAnGelbAus;
46        Zustand1 = 'StrasseTestOB or
47          Zustand1 = 'StrasseKeinSignal or
48          Zustand1 = 'StrasseRotAnGelbAus;
49        transition[Kommando='StrasseEntsichern] WaitGruen;
50        transition/*[Kommando!='StrasseEntsichern]*/ Rot;
51        transition[Gestoert] Gestoert;
52        transition[GestoertTest] Gestoert;
53        transition[Kommando='StrasseStoerung] Gestoert;
54      }
55      final state Gestoert {
56        transition Gestoert;
57      }
58    };
```

3.3. Überwachungssignal

3.3.1. Schaltverhalten

Im Schaltverhalten der Komponente ist vor allem wichtig, dass das Fahrtsignal nur im Zustand „Fahrt“ gezeigt werden darf, ansonsten muss immer das Haltesignal gezeigt werden. Ansonsten muss nur sichergestellt sein, dass die Zustandsanzeige korrekt ist, also nur dann „Gestört“ gemeldet werden kann, wenn die Komponente im entsprechenden Zustand ist. Das Schaltverhalten ist in Abbildung 3.5 gezeigt, die Invarianten für die Zustände in Tabelle 3.4.

Zustand	Invarianten
Init	UmgebungTfZug = TfZugHaltbegriff Zustand $\neq$ UeberwachungGestoert
Halt	UmgebungTfZug = TfZugHaltbegriff Zustand $\neq$ UeberwachungGestoert
Fahrt	UmgebungTfZug = TfZugFahrtbegriff Zustand = UeberwachungAn $\vee$ Kommando=UeberwachungTest
Gestört	UmgebungTfZug = TfZugHaltbegriff

Tabelle 3.4.: Zustandsinvarianten des Überwachungssignals

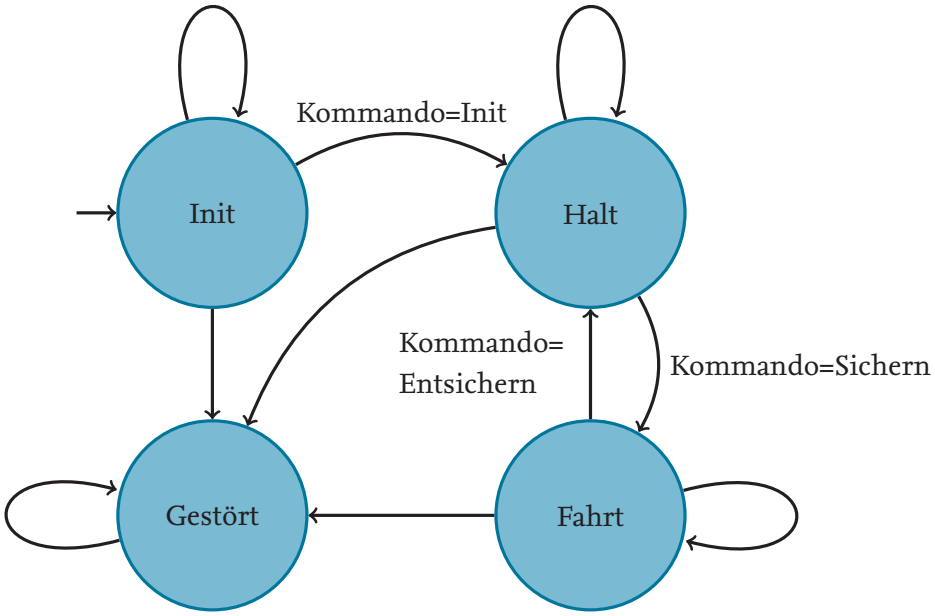


Abbildung 3.5.: Überwachungssignal Schaltverhalten

```
1   automaton {
2     init final state Init {
3       UmgebungTfZug = 'TfZugHaltbegriff;
4       Zustand != 'UeberwachungGestoert;
5       transition Init;
6       transition[Kommando='UeberwachungInit] Halt;
7       transition Gestoert;
8     }
9     final state Halt {
10      UmgebungTfZug = 'TfZugHaltbegriff;
11      Zustand != 'UeberwachungGestoert;
12      transition[Kommando='UeberwachungSichern] Fahrt;
13      transition Halt;
14      transition Gestoert;
15    }
16    final state Fahrt {
17      UmgebungTfZug = 'TfZugFahrtbegriff;
18      Zustand = 'UeberwachungAn or Kommando = 'UeberwachungTest;
19      transition[Kommando='UeberwachungEntsichern] Halt;
20      transition Fahrt;
21      transition Gestoert;
22    }
23    final state Gestoert {
24      UmgebungTfZug = 'TfZugHaltbegriff;
```

```
25        transition Gestoert;
26      }
27    };
```

3.4. Achszählrechner

Der Achszählrechner spielt keine große Rolle in der Verifikation des Gesamtsystems. Es muss lediglich verifiziert werden, dass er nur dann den Status „Gestört“ meldet, wenn tatsächlich eine Störung anliegt.

3.4.1. Störungsverhalten

Hierfür wird das Verhalten der Komponente als Automat dargestellt, der zunächst auf die Initialisierung wartet und danach im „Aktiv“-Zustand bleibt. In den „Gestört“-Zustand wird nur übergegangen, wenn das „Gestört“- oder „TestGestört“-Signal anliegt. Der genaue Automat ist in Abbildung 3.6 zu sehen.

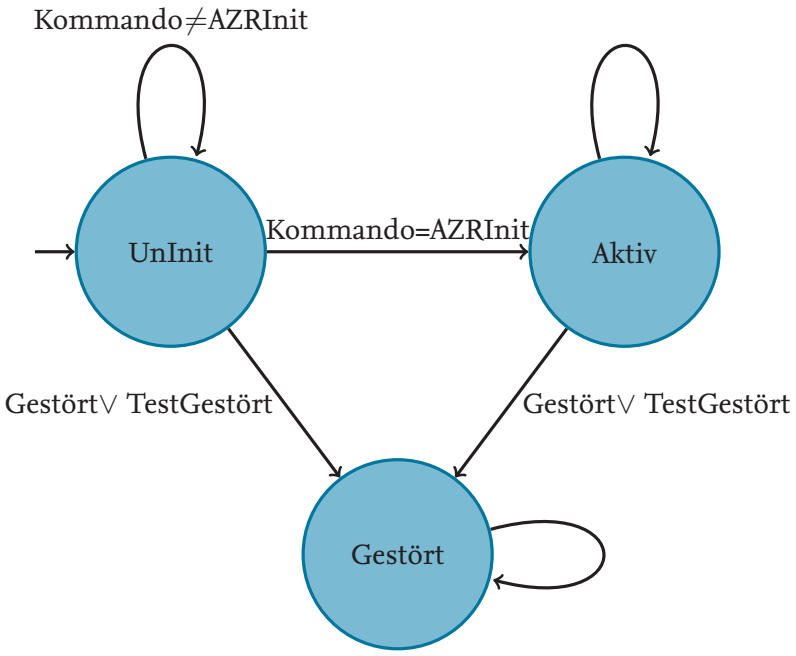


Abbildung 3.6.: Achszählrechner Störungsverhalten

```
1   automaton {
2     init final state Uninit {
3       Zustand = 'AZRKeinSignal or Zustand = 'AZRTestOB;
4       transition[Gestoert] Gestoert;
5       transition[TestGestoert] Gestoert;
6       transition[Kommando!='AZRInit] Uninit;
7       transition[Kommando='AZRInit] Aktiv;
8     }
9     final state Gestoert {
```

```
10        //Zustand = 'AZRGestoert;
11        transition Gestoert;
12      }
13      final state Aktiv {
14        Zustand = 'AZRTestOB or
15          Zustand = 'AZRFrei or
16          Zustand = 'AZRBelegt;
17        transition[Gestoert] Gestoert;
18        transition[TestGestoert] Gestoert;
19        transition Aktiv;
20      }
21    };
```

3.5. Schranke

Da auch die Schranke keine bedeutende Rolle im globalen Verifikationsziel spielt, muss auch sie nicht sonderlich detailiert modelliert werden. Allerdings ist ihr Störungsverhalten nicht von ihrem normalen Verhalten zu separieren. Daher müssen auch Teile ihres normalen Verhaltens im Kontrakt abgebildet werden.

3.5.1. Basisverhalten

Das Basisverhalten der Schranke stellt den normalen Schrankenzyklus dar: Neben dem uninitialisierten Zustand und dem Fehlerzustand „Gestört“ kann die Schranke entweder geschlossen, geöffnet oder in einem unbekannten Zustand sein. Wichtig ist, dass die Schranke im geöffneten Zustand nur durch das „Gestört“ Signal in den entsprechenden Zustand übergehen kann.

```
1   automaton {
2     init final state Uninit {
3       SchrankeZustand = 'SchrankeUnbekannt;
4       UmgebungVerkehr = 'VerkehrFrei;
5       transition Uninit;
6       transition[SchrankeKommando='SchrankeInit] ZustandUnbekannt;
7     }
8     final state ZustandUnbekannt {
9       SchrankeZustand = 'SchrankeUnbekannt;
10      UmgebungVerkehr = 'VerkehrBlockiert;
11      transition[not Gestoert] ZustandUnbekannt;
12      transition[not Gestoert] Geoeffnet;
13      transition[not Gestoert] Geschlossen;
14      transition Gestoert;
15      // This is required because the transition to Gestoert is weak
16      transition[ServiceSchnittstelleSchranke] Uninit;
17    }
18    final state Geoeffnet {
19      SchrankeZustand = 'SchrankeOffen;
```

```
20        UmgebungVerkehr = 'VerkehrFrei;
21        transition[not Gestoert and
22                   SchrankeKommando='SchrankeSchliessen] ZustandUnbekannt;
23        transition[not Gestoert and
24                   SchrankeKommando!='SchrankeSchliessen] Geoeffnet;
25        transition[Gestoert] Gestoert;
26      }
27      final state Gestoert {
28        SchrankeZustand = 'SchrankeGestoert;
29        UmgebungVerkehr = 'VerkehrFrei;
30        transition[not ServiceSchnittstelleSchranke] Gestoert;
31        transition[ServiceSchnittstelleSchranke] Uninit;
32      }
33      final state Geschlossen {
34        SchrankeZustand = 'SchrankeGeschlossen;
35        UmgebungVerkehr = 'VerkehrBlockiert;
36        transition[Gestoert] Gestoert;
37        transition[not Gestoert and
38                   SchrankeKommando!='SchrankeOeffnen] Geschlossen;
39        transition[not Gestoert and
40                   SchrankeKommando='SchrankeOeffnen] ZustandUnbekannt;
41      }
42    };
```

3.6. Verifikationsziele

3.6.1. „Niemals grün für alle“

Dieses Verifikationsziel spezifiziert, dass das Bahnübergangssystem niemals gleichzeitig für Züge und Autos grün zeigt. Da allerdings das Straßensignal bei einer Störung den KFZ freie Fahrt anzeigt, muss das Verifikationsziel auf die folgende Form abgeschwächt werden:

> Unter der Annahme, dass keine Komponente gestört wird, zeigt das Straßensignal immer Rot, wenn das Überwachungssignal Fahrt zeigt.

Dieses Verifikationsziel lässt sich in GTL wie folgt darstellen:

```
1   (always (not strassensignal0.Gestoert and
2            not strassensignal0.GestoertTest and
3            not strassensignal1.Gestoert and
4            not strassensignal1.GestoertTest and
5            not strassensignal2.Gestoert and
6            not strassensignal2.GestoertTest and
7            not strassensignal3.Gestoert and
8            not strassensignal3.GestoertTest and
9            not schranke0.Gestoert and
```

```
10              not schranke1.Gestoert and
11              not achszaehlrechner.Gestoert and
12              not achszaehlrechner.TestGestoert and
13              not ueberwachungssignal.Zustand = 'UeberwachungGestoert
14      )) =>
15      (always (ueberwachungssignal.UmgebungTfZug = 'TfZugFahrtbegriff
16              => strassensignal0.UmgebungKfz = 'KfzRotAnGelbAus));
```

4 Verifikationsergebnisse

4.1. Kontrakte

Die Verifikation der Kontrakte wurde mit dem SCADE Design Verifier (Version 6.2) mit der „Prove“-Strategie durchgeführt. Der verwendete Computer ist ein Vierkern-Prozessor „AMD Phenom II X4 840“ mit 1.5GB Arbeitsspeicher.

Kontraktname	**Abschnitt**	**Ergebnis**	**Laufzeit** (s)
Übergangssteuerung	3.1		
Befehlssendeverhalten	3.1.1	erfolgreich	4.571
Störungsverhalten	3.1.2	erfolgreich	3.136
Initialisierungsverhalten	3.1.3	kein Ergebnis[1]	–
Straßensignal	3.2		
Ausgabekonsistenz	3.2.1	erfolgreich	2.621
Schaltverhalten	3.2.2	erfolgreich	3.104
Überwachungssignal	3.3		
Schaltverhalten	3.3.1	erfolgreich	5.373
Achszählrechner	3.4		
Störungsverhalten	3.4.1	erfolgreich	3.370
Schranke	3.5		
Basisverhalten	3.5.1	erfolgreich	3.026

Tabelle 4.2.: Kontrakt-Verifikations-Ergebnisse

4.2. Globale Verifikation

Die globale Verifikation wurde mit dem Bounded-Model-Checking basierend auf dem Z3-SMT-Solver durchgeführt. Das Gesamtmodell wurde bis zur Tiefe 66 durchsucht, ohne das Fehler gefunden wurden. Die Laufzeit ist hierbei exponentiell abhängig von der Suchtiefe, wie in Abbildung 4.1 zu sehen ist.

Die meisten Fehler beim Zusammenspiel der Kontrakte wurden in den Suchtiefen 20–22 gefunden.

[1] Der SCADE Design Verifier wurde hier 6 Tage lang ohne Ergebnis laufen gelassen. Die Vermutung ist, dass der Kontrakt korrekt ist.

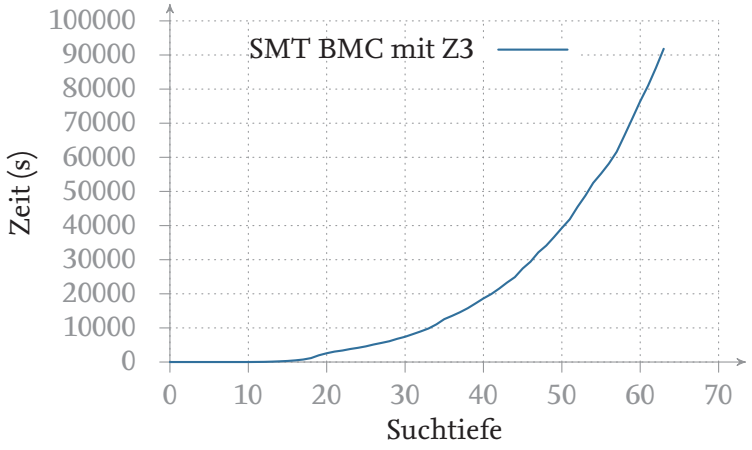


Abbildung 4.1.: Globale Verifikation: Laufzeit vs. Suchtiefe

4.3. Gefundene Fehler

4.3.1. Straßensignal

In der Komponente „Straßensignal“ hatten mehrere Transitionen in den „Gestört“-Zustand nicht die höchste Priorität. Dies führte dazu, dass die Komponente unter bestimmten Umständen nicht korrekt auf das Anliegen des „Gestört“-Signals reagierte. Aufgedeckt durch den Kontrakt 3.2.2.

4.3.2. Schranke

Auch in der Komponente „Schranke“ hatte eine Transition von „Geöffnet“ nach „Gestört“ nicht die höchste Priorität. Aufgedeckt durch den Kontrakt 3.5.1.

4.3.3. Überwachungssignal

Im Zustand „Normalbetrieb“ war es für das Überwachungssignal möglich, den Zügen „Fahrt“ anzuzeigen, obwohl ein Straßensignal den Zustand „Gestört“ meldete.

4.4. „Lessons-learned“

- Die Erstellung von Kontrakten erfordert häufig tieferen Einblick in die interne Funktionsweise der Komponenten.
- Kontrakte müssen auf das globale Verifikationsziel zugeschnitten werden, um gute Ergebnisse zu erzielen.
- Schon das Erstellen und die lokale Verifikation von Kontrakten erhöht die Modellqualität durch das Auffinden von Fehlern in den Komponenten, die nicht unmittelbar durch das globale Verifikationsziel abgedeckt werden.
- Für Kontrakte, die mittels SCADE Design Verifier nicht verifiziert werden, ist eine andere Korrektheitsargumentation notwendig. Dazu werden in als nächste Schritte weitere Modellprüfungsverfahren und Testverfahren untersucht.
- Die Ergebnisse zum BMC zeigen, dass gut geeignet ist, um Fehler im Modell zu finden.

Literaturverzeichnis

[SZH11] Sulzmann, M. ; Zechner, A. ; Hedayati, R.: Anforderungsdokument für die Fallstudie Bahnübergangssicherungsanlage / ICS AG. 2011. – Forschungsbericht

[Ver11] ICS AG, Verified Systems International GmbH, TU Braunschweig: Contract Specification and Domain Specific Modelling Language for GALS Systems, An Approach to System Validation. 2011. – Forschungsbericht. – Available at www.versyko.de

A GTL-Spezifikation

```
1  model[scade] uebergangssteuerung
2    ("../KCG/kcg_xml_filter_out.scade",
3     "BahnUebergangsSteuerung::BahnUebergangsSteuerung") {
4    init KommandoStrassenSignal ['StrasseKeinKommando,
5                                 'StrasseKeinKommando,
6                                 'StrasseKeinKommando,
7                                 'StrasseKeinKommando];
8    automaton {
9      init final state Open {
10       KommandoUeberwachung!='UeberwachungSichern;
11       transition Open;
12       transition[KommandoStrassenSignal[0]='StrasseSichern] Closing;
13     }
14     final state Closing {
15       KommandoUeberwachung!='UeberwachungSichern;
16       transition[ZustandStrassenSignal=['StrasseRotAnGelbAus,
17                                         'StrasseRotAnGelbAus,
18                                         'StrasseRotAnGelbAus,
19                                         'StrasseRotAnGelbAus]] Closed;
20       transition[ZustandStrassenSignal!=['StrasseRotAnGelbAus,
21                                          'StrasseRotAnGelbAus,
22                                          'StrasseRotAnGelbAus,
23                                          'StrasseRotAnGelbAus]] Closing;
24     }
25     final state Closed {
26       KommandoStrassenSignal[0]!='StrasseEntsichern;
27       transition[KommandoUeberwachung='UeberwachungEntsichern] Opening;
28       transition[KommandoUeberwachung!='UeberwachungEntsichern] Closed;
29     }
30     final state Opening {
31       KommandoStrassenSignal[0]!='StrasseEntsichern;
32       transition Opening;
33       transition[ZustandUeberwachung='UeberwachungAus] Open;
34     }
35   };
36   // Stoerungsautomat
```

```
37    automaton {
38      init final state Ungestoert {
39        KommandoStrassenSignal[0]!='StrasseStoerung;
40        KommandoStrassenSignal[1]!='StrasseStoerung;
41        KommandoStrassenSignal[2]!='StrasseStoerung;
42        KommandoStrassenSignal[3]!='StrasseStoerung;
43        transition Ungestoert;
44        transition[ZustandAZR='AZRGestoert or
45                   ZustandStrassenSignal[0]='StrasseGestoert or
46                   ZustandStrassenSignal[1]='StrasseGestoert or
47                   ZustandStrassenSignal[2]='StrasseGestoert or
48                   ZustandStrassenSignal[3]='StrasseGestoert or
49                   ZustandUeberwachung='UeberwachungGestoert] Stoerung;
50      }
51      final state Stoerung {
52        transition[EntstoerungVomServicepersonal] Entstoerung;
53        transition Gestoert;
54      }
55      final state Gestoert {
56        //KommandoStrassenSignal[0]='StrasseStoerung;
57        transition[EntstoerungVomServicepersonal] Entstoerung;
58        transition Gestoert;
59      }
60      final state Entstoerung {
61        transition Ungestoert;
62      }
63    };
64    // Initialisation von Ampel abwarten
65    // Funktioniert nicht :(
66    automaton {
67      init final state Uninit {
68        KommandoUeberwachung!='UeberwachungSichern;
69        transition[KommandoStrassenSignal[0]='StrasseInit] Init;
70        transition[KommandoStrassenSignal[0]!='StrasseInit] Uninit;
71      }
72      final state Init {
73        transition Init;
74      }
75    };
76  }
77
78  instance uebergangssteuerung uebergangssteuerung;
79
```

```
80  model[scade] schranke
81    ("../KCG/kcg_xml_filter_out.scade","Schranke::Schranke") {
82    automaton {
83      init final state Uninit {
84        SchrankeZustand = 'SchrankeUnbekannt;
85        UmgebungVerkehr = 'VerkehrFrei;
86        transition Uninit;
87        transition[SchrankeKommando='SchrankeInit] ZustandUnbekannt;
88      }
89      final state ZustandUnbekannt {
90        SchrankeZustand = 'SchrankeUnbekannt;
91        UmgebungVerkehr = 'VerkehrBlockiert;
92        transition[not Gestoert] ZustandUnbekannt;
93        transition[not Gestoert] Geoeffnet;
94        transition[not Gestoert] Geschlossen;
95        transition Gestoert;
96        // This is required because the transition to Gestoert is weak
97        transition[ServiceSchnittstelleSchranke] Uninit;
98      }
99      final state Geoeffnet {
100       SchrankeZustand = 'SchrankeOffen;
101       UmgebungVerkehr = 'VerkehrFrei;
102       transition[not Gestoert and
103                  SchrankeKommando='SchrankeSchliessen] ZustandUnbekannt;
104       transition[not Gestoert and
105                  SchrankeKommando!='SchrankeSchliessen] Geoeffnet;
106       transition[Gestoert] Gestoert;
107     }
108     final state Gestoert {
109       SchrankeZustand = 'SchrankeGestoert;
110       UmgebungVerkehr = 'VerkehrFrei;
111       transition[not ServiceSchnittstelleSchranke] Gestoert;
112       transition[ServiceSchnittstelleSchranke] Uninit;
113     }
114     final state Geschlossen {
115       SchrankeZustand = 'SchrankeGeschlossen;
116       UmgebungVerkehr = 'VerkehrBlockiert;
117       transition[Gestoert] Gestoert;
118       transition[not Gestoert and
119                  SchrankeKommando!='SchrankeOeffnen] Geschlossen;
120       transition[not Gestoert and
121                  SchrankeKommando='SchrankeOeffnen] ZustandUnbekannt;
122     }
```

```
123   };
124 }
125
126 instance schranke schranke0;
127 instance schranke schranke1;
128
129 model[scade] strassensignal("../KCG/kcg_xml_filter_out.scade","StrassenSignal::Stras
130   init Zustand1 'StrasseKeinSignal;
131   always (Zustand0 = Zustand1);
132   //Behaviour filtering failure traces
133   automaton {
134     init final state Uninit {
135       UmgebungKfz = 'KfzRotAusGelbAus;
136       Zustand1 = 'StrasseTestOB or Zustand1 = 'StrasseKeinSignal;
137       transition[Kommando='StrasseInit] Gruen;
138       transition[Kommando!='StrasseInit] Uninit;
139       transition[Gestoert] Gestoert;
140       transition[GestoertTest] Gestoert;
141       transition[Kommando='StrasseStoerung] Gestoert;
142     }
143     final state WaitGruen {
144       UmgebungKfz = 'KfzRotAnGelbAn;
145       Zustand1 = 'StrasseTestOB or
146         Zustand1 = 'StrasseKeinSignal or
147         Zustand1 = 'StrasseRotAnGelbAn;
148       transition WaitGruen;
149       transition Gruen;
150       transition[Gestoert] Gestoert;
151       transition[GestoertTest] Gestoert;
152       transition[Kommando='StrasseStoerung] Gestoert;
153     }
154     final state Gruen {
155       UmgebungKfz = 'KfzRotAusGelbAus;
156       Zustand1 = 'StrasseTestOB or
157         Zustand1 = 'StrasseKeinSignal or
158         Zustand1 = 'StrasseRotAusGelbAus;
159       transition[Kommando='StrasseSichern] WaitRot;
160       transition[Kommando!='StrasseSichern] Gruen;
161       transition[Gestoert] Gestoert;
162       transition[GestoertTest] Gestoert;
163       transition[Kommando='StrasseStoerung] Gestoert;
164     }
165     final state WaitRot {
```

```
166         UmgebungKfz = 'KfzRotAusGelbAn;
167         Zustand1 = 'StrasseTestOB or
168           Zustand1 = 'StrasseKeinSignal or
169           Zustand1 = 'StrasseRotAusGelbAn;
170         transition WaitRot;
171         transition Rot;
172         transition[Gestoert] Gestoert;
173         transition[GestoertTest] Gestoert;
174         transition[Kommando='StrasseStoerung] Gestoert;
175       }
176       final state Rot {
177         UmgebungKfz = 'KfzRotAnGelbAus;
178         Zustand1 = 'StrasseTestOB or
179           Zustand1 = 'StrasseKeinSignal or
180           Zustand1 = 'StrasseRotAnGelbAus;
181         transition[Kommando='StrasseEntsichern] WaitGruen;
182         transition/*[Kommando!='StrasseEntsichern]*/ Rot;
183         transition[Gestoert] Gestoert;
184         transition[GestoertTest] Gestoert;
185         transition[Kommando='StrasseStoerung] Gestoert;
186       }
187       final state Gestoert {
188         transition Gestoert;
189       }
190     };
191   }
192
193   instance strassensignal strassensignal0;
194   instance strassensignal strassensignal1;
195   instance strassensignal strassensignal2;
196   instance strassensignal strassensignal3;
197
198   model[scade] ueberwachungssignal
199     ("../KCG/kcg_xml_filter_out.scade",
200      "UeberwachungsSignal::UeberwachungsSignal") {
201     init UmgebungTfZug 'TfZugHaltbegriff;
202     automaton {
203       init final state Init {
204         UmgebungTfZug = 'TfZugHaltbegriff;
205         Zustand != 'UeberwachungGestoert;
206         transition Init;
207         transition[Kommando='UeberwachungInit] Halt;
208         transition Gestoert;
```

```
209       }
210       final state Halt {
211         UmgebungTfZug = 'TfZugHaltbegriff;
212         Zustand != 'UeberwachungGestoert;
213         transition[Kommando='UeberwachungSichern] Fahrt;
214         transition Halt;
215         transition Gestoert;
216       }
217       final state Fahrt {
218         UmgebungTfZug = 'TfZugFahrtbegriff;
219         Zustand = 'UeberwachungAn or Kommando = 'UeberwachungTest;
220         transition[Kommando='UeberwachungEntsichern] Halt;
221         transition Fahrt;
222         transition Gestoert;
223       }
224       final state Gestoert {
225         UmgebungTfZug = 'TfZugHaltbegriff;
226         transition Gestoert;
227       }
228     };
229   }
230
231   instance ueberwachungssignal ueberwachungssignal;
232
233   model[scade] achszaehlrechner
234     ("../KCG/kcg_xml_filter_out.scade",
235      "AchsZaehlRechner::AchsZaehlRechner") {
236     init Zustand 'AZRKeinSignal;
237     automaton {
238       init final state Uninit {
239         Zustand = 'AZRKeinSignal or Zustand = 'AZRTestOB;
240         transition[Gestoert] Gestoert;
241         transition[TestGestoert] Gestoert;
242         transition[Kommando!='AZRInit] Uninit;
243         transition[Kommando='AZRInit] Aktiv;
244       }
245       final state Gestoert {
246         //Zustand = 'AZRGestoert;
247         transition Gestoert;
248       }
249       final state Aktiv {
250         Zustand = 'AZRTestOB or Zustand = 'AZRFrei or Zustand = 'AZRBelegt;
251         transition[Gestoert] Gestoert;
```

```
252         transition[TestGestoert] Gestoert;
253         transition Aktiv;
254       }
255     };
256 }
257
258 instance achszaehlrechner achszaehlrechner;
259
260 connect strassensignal0.Zustand0
261         ueberwachungssignal.ZustandStrassensignal[0];
262 connect strassensignal1.Zustand0
263         ueberwachungssignal.ZustandStrassensignal[1];
264 connect strassensignal2.Zustand0
265         ueberwachungssignal.ZustandStrassensignal[2];
266 connect strassensignal3.Zustand0
267         ueberwachungssignal.ZustandStrassensignal[3];
268
269 connect strassensignal0.Zustand1
270         uebergangssteuerung.ZustandStrassenSignal[0];
271 connect strassensignal1.Zustand1
272         uebergangssteuerung.ZustandStrassenSignal[1];
273 connect strassensignal2.Zustand1
274         uebergangssteuerung.ZustandStrassenSignal[2];
275 connect strassensignal3.Zustand1
276         uebergangssteuerung.ZustandStrassenSignal[3];
277
278 connect schranke0.SchrankeZustand
279         uebergangssteuerung.ZustandSchranke[0];
280 connect schranke1.SchrankeZustand
281         uebergangssteuerung.ZustandSchranke[1];
282
283 connect ueberwachungssignal.Zustand
284         uebergangssteuerung.ZustandUeberwachung;
285 connect ueberwachungssignal.Zustand
286         strassensignal0.ZustandUeberwachungssignal;
287 connect ueberwachungssignal.Zustand
288         strassensignal1.ZustandUeberwachungssignal;
289 connect ueberwachungssignal.Zustand
290         strassensignal2.ZustandUeberwachungssignal;
291 connect ueberwachungssignal.Zustand
292         strassensignal3.ZustandUeberwachungssignal;
293
294 connect achszaehlrechner.Zustand uebergangssteuerung.ZustandAZR;
```

```
295
296 connect uebergangssteuerung.KommandoSchranke[0]
297         schranke0.SchrankeKommando;
298 connect uebergangssteuerung.KommandoSchranke[1]
299         schranke1.SchrankeKommando;
300 connect uebergangssteuerung.KommandoAZR
301         achszaehlrechner.Kommando;
302
303 connect uebergangssteuerung.KommandoStrassenSignal[0]
304         strassensignal0.Kommando;
305 connect uebergangssteuerung.KommandoStrassenSignal[1]
306         strassensignal1.Kommando;
307 connect uebergangssteuerung.KommandoStrassenSignal[2]
308         strassensignal2.Kommando;
309 connect uebergangssteuerung.KommandoStrassenSignal[3]
310         strassensignal3.Kommando;
311
312 connect uebergangssteuerung.KommandoUeberwachung
313         ueberwachungssignal.Kommando;
314
315 verify {
316   (always (not strassensignal0.Gestoert and
317            not strassensignal0.GestoertTest and
318            not strassensignal1.Gestoert and
319            not strassensignal1.GestoertTest and
320            not strassensignal2.Gestoert and
321            not strassensignal2.GestoertTest and
322            not strassensignal3.Gestoert and
323            not strassensignal3.GestoertTest and
324            not schranke0.Gestoert and
325            not schranke1.Gestoert and
326            not achszaehlrechner.Gestoert and
327            not achszaehlrechner.TestGestoert and
328            not ueberwachungssignal.Zustand = 'UeberwachungGestoert
329   )) =>
330   (always (ueberwachungssignal.UmgebungTfZug = 'TfZugFahrtbegriff =>
331            strassensignal0.UmgebungKfz = 'KfzRotAnGelbAus));
332 }
```