Otto-von-Guericke-Universität Magdeburg

Fakultät für Informatik

Bachelorarbeit

Erweiterung einer Parametrierungssprache für Verkehrssimulationen unter Einbeziehung von LSA-Steuerungen

Autor:

Nils Stach

29. 03. 2016

Betreuer:

Prof. Dr.-Ing. habil. Graham Horton

Institut für Simulation und Graphik Otto-von-Guericke Universität

Stach, Nils:
Erweiterung einer Parametrierungssprache für Verkehrssimulationen unter Einbeziehung von LSA-Steuerungen
Bachelorarbeit, Otto-von-Guericke-Universität Magdeburg, 2016.

Einführung

TrafficSim ist eine Deklarationssprache zur Konfigurierung von Verkehrssimulationen. Durch sie können Kreuzungen beschrieben und Simulationen konfiguriert werden. Ziel ist die Generierung von Testdaten.
Im Zuge dieser Arbeit soll die Deklarationssprache um Lichtsignalanlagen, im Weiteren durch LSA abgekürzt, erweitert werden. Für diese sollen Steuerungsprogramme erzeugt werden. Dadurch sollen neue Simulationsszenarien erzeugt werden können, um weitere Testdaten zu generieren.
Zuerst wird die Deklarationssprache analysiert. Die Vorgehensweise bei der Beschreibung einer Simulation soll erkannt werden. Dafür muss der Umfang der Sprache beschrieben werden. Daher wird eine Auflistung der Sprachelemente angefertigt. Ziel ist die Umsetzung an das bestehende Eingabemuster anzupassen.
Als nächstes werden bestehende Signalprogramme und ihre rechtlichen Vorgaben analysiert. Ziel ist bestehende Regelungen bei der Erstellung von Lichtsignalanlagenprogrammen herauszuarbeiten. Diese müssen bei der Umsetzung berücksichtigt werden. Hierbei wird eine Unterscheidung zwischen festen und beeinflussbaren Signalprogrammen vorgenommen.
Anschließend werden Konzepte für die Eingabe von Signalprogrammen erarbeitet und beschrieben. Diese werden auf ihre Vorteile und Nachteile untersucht. Ziel ist eine einfache, verständliche Eingabe zu schaffen, welche sich an der bisherigen Eingabe orientiert. Auch die für die Umsetzung in SUMO notwendigen Elemente sollen berücksichtigt werden.
Im nächsten Kapitel wird diese Umsetzung beschrieben. Implementierte Sprachelemente und Änderungen bei der Simulationserzeugung werden beschrieben. Die Umsetzung erfolgt für die Simulationssoftware SUMO. Besonderheiten die sich durch SUMO ergeben werden aufgezeigt.
Abschließend soll die Umsetzung durch Experimente validiert werden. Dabei muss die lose Koppelung nachgewiesen werden. Auch die Erzeugung korrekter Signalprogramme soll gezeigt werden. Ziel ist die Erzeugung eines realen Signalprogramms. Das Erreichen dieses Ziels wird durch ein Experiment nachgewiesen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Quelltextverzeichnis

1. Einführung

1.1 Motivation

Dem Straßenverkehr fällt in der heutigen Zeit eine große Bedeutung für den Transport von Gütern und Personen zu. So wurden in Deutschland im Jahr 2014 über drei Milliarden Tonnen an Gütern mithilfe von Lastkraftwagen bewegt. Schienen- und Schiffsverkehr kommen im selben Zeitraum zusammen auf einen Wert von etwas unter 900.000 Tonnen [Bunb]. Ein Grund für diesen Unterschied ist, dass die meisten Waren nur über Straßen an ihr endgültiges Ziel gebracht werden können. Da keine Wasserwege oder Schienen bis zu diesen führen, erfolgt ein Umladen auf Lastkraftwagen, welche die endgültige Verteilung vornehmen.
Jedoch bewegen sich nicht nur Lastkraftwagen auf den Straßen. Insgesamt sind in Deutschland zum 01.01.2015 mehr als 53,7 Millionen Fahrzeuge zugelassen [Buna]. Den größten Anteil daran machen die Personenkraftwagen, mit über 44,4 Millionen Zulassungen, aus. Das ist eine Steigerung um 748.800 Fahrzeuge im Vergleich zum Vorjahr [Buna]. Beim Straßennetz ist mit 230.100 Kilometern am 01.01.2015 ein leichter Rückgang zu 2014 zu verzeichnen[Bund]. Aufgrund von rund 2,4 Millionen Unfälle, bei denen 3.377 Menschen starben, ist die Sicherheit im Straßenverkehr ein großes Thema [Bunc]. Mit steigender Anzahl an Fahrzeugen steigt auch die Gefahr eines Unfalls. Daher werden Systeme entwickelt um diese zu vermeiden. Somit steigt auch die Bedeutung von verteilten IT-Systemen in Fahrzeugen.
Ein Teil solcher Systeme bedient die Car-2-X Kommunikation. In dieser werden Daten in Echtzeit zwischen den Fahrzeugen und ihrer Umgebung ausgetauscht. Aktuell befinden sich Systeme, die ohne den Menschen einen Zug oder ein Auto steuern können, in der Entwicklung [Gra09]. Die so entwickelten Anwendungen müssen mit der steigenden Anzahl an Daten umgehen können und diese fehlerfrei verarbeiten. Da der Straßenverkehr ein sicherheitskritischer Bereich ist, dürfen keine Ausfälle und Fehler auftreten. Jedoch sind verteilte Systeme meist sehr komplex und besitzen mehrere einzelne Komponenten, die miteinander agieren. Das Testen dieser ist in der Realität sehr aufwendig und teuer, da Prototypen hergestellt und ein Testszenario erschaffen werden muss.
Durch Simulation kann dieser Aufwand reduziert werden, weswegen häufig auf diese zurückgegriffen wird. Für das Erstellen einer Simulation existieren viele verschiedene Simulationsprogramme. Diese unterscheiden sich in Umfang, Komplexität, Genauigkeit und der Art der Eingabe. Daher wird ein Grundwissen über Simulationen und über das verwendete Programm benötigt. Das Einarbeiten und Erstellen einer Simulationen benötigt daher Zeit. Auch werden häufig nicht alle Features des Simulationsprogrammes verwendet.
Mithilfe der vom ifak [iMe] entwickelten Deklarationssprache TrafficSim soll es möglich sein, schnell eine Simulation zur Testdatenerzeugung zu erstellen [Bee15]. Durch einfache, leicht verständliche Eingaben soll es dem Nutzer erlaubt werden eine Kon-

figuration der getesteten Szenarien vorzunehmen. Ziel ist die Erzeugung von relevanten Testdaten zu diesen. Die entwickelte Sprache heißt TrafficSim und kann bisher einzelne Kreuzungen erzeugen und die Anzahl der ankommenden Fahrzeuge einstellen [Bee15].
Da an Kreuzungen ein hohes Gefahrenpotential durch Abbieger und Fußgänger existiert, werden diese zum Teil durch Lichtsignalanlagen gesteuert. Mithilfe von verschiedenen Programmen zur Steuerung dieser, ergeben sich neue Szenarien für die zu testende Anwendung. Daher und um eine bessere Anpassung an die Realität zu ermöglichen, soll die Sprache um Lichtsignalanlagen erweitert werden.

1.2 Ziel der Arbeit

Das Ziel dieser Arbeit ist die Erweiterung der bestehenden Deklarationssprache TrafficSim. Dabei ist die Integration von Lichtsignalanlagen das Hauptziel. Dem Nutzer soll die Konfiguration der Steuerprogramme der LSA ermöglicht werden. Er soll eigene Programme erstellen und ausführen können. Dafür müssen mehrere Optionen der Eingabe geschaffen werden,die leicht verständlich sind und sich an die bisherigen Eingaben anpassen. Auch ohne großen Aufwand für den Nutzer sollen die Steuerprogramme beeinflusst werden können. Verschiedene Aspekte der Lichtsignalanlagen sollen konfigurierbar sein. Ziel ist außerdem durch die Erweiterung neue Testszenarien zu schaffen, um somit mehr Testaten zu erzeugen. Auch eine genauere Anpassung an die Realität soll ermöglicht werden. Dabei muss die Erweiterung bestehende Festlegungen, wie das Prinzip der Losen Koppelung, beibehalten. Ein beschriebenes LSA-Programm soll daher auf alle Kreuzungen mit gleicher Anzahl an Armen anwendbar sein. Reale Steuerprogramme sollen sich nachbilden lassen.

1.3 Aufgabenstellung

Der erste Schritt zur Erweiterung der Deklarationssprache ist die Analyse dieser. Dabei soll der aktuelle Umfang ermittelt werden. Die Eigenschaften der bisherigen Umsetzung sind zu überprüfen. Dabei wird auch die Form der Eingabe untersucht. Bedingungen, die bei der Erweiterung einzuhalten sind, müssen erkannt werden.
Der zweite Arbeitsschritt ist die Analyse gängiger Steuerprogramme für Lichtsignalanlagen. Deren Eigenschaften und Randbedingungen sind zu untersuchen. Dabei sollen Festzeitprogramme und verkehrsabhängige Programme betrachtet werden. Regeln die für erstellte LSA-Programme gelten, sollen in diesem Schritt herausgearbeitet werden. Anschließend wird die Vorgehensweise bei der Spezifizierung von LSA Steuerprogrammen analysiert.
Aus den so gewonnenen Informationen soll ein Konzept zur Deklaration von LSA-Steuerprogrammen entwickelt werden. Dieses muss sich in die vorhandene Deklarationssprache einpassen. Die Form soll sich an der bisherigen Eingabe orientieren und dieses Konzept anschließend prototypisch für SUMO umsetzen. Auch dafür soll vorher eine Vorgehensweise erarbeitet werden.
Nach der Umsetzung soll diese evaluiert und dokumentiert werden. Dafür sind realitätsnahe Experimente durchzuführen. Auch das Einhalten der gefundenen Bedingungen ist zu zeigen.

2. Grundlagen

Dieses Kapitel behandelt die Grundlagen, welche für das Verständnis der weiteren Arbeit notwendig sind. Es dient dem Einführen in Bereiche, die für spätere Erklärungen benötigt werden. Beinhaltet sind die Bereiche Verkehr, Simulation und C2X.

2.1 Verkehr

Verkehr wird vom Brockhaus als „Gesamtheit aller Ortsveränderungen von Person und Gütern und Nachrichten als grundlegende Voraussetzung für soziale Teilhabe sowie für arbeitsteiliges Wirtschaften und wirtschaftl. Spezialisierung.“ [Bro06] definiert.
Für den Verkehr kann eine Unterteilung in Straßen-, Schienen-, Wasser- und Luftverkehr vorgenommen werden [Bro06]. Diese unterscheiden sich durch verschiedene Bedingungen, welche für die Fortbewegung gelten. Speziell der Untergrund ist hierbei namensgebend. Für die vier Bereiche existieren eigene Fortbewegungsmittel. So sind im Schienenverkehr Züge und Straßenbahnen zusammengefasst, da diese sich nur auf bereitgestellten Schienen bewegen können. Der Luftverkehr beinhaltet Flugzeuge, Helikopter und Drohnen. Er zeichnet sich dadurch aus, dass er die kürzesten Verbindungen ermöglicht, da nur wenige Hindernisse in der Höhe existieren. Mit dem Schiffsverkehr ist es möglich Flüsse, Seen und Ozeane zu befahren. Dafür werden Schiffe benötigt, aber auch U-Boote zählen in diesen Bereich. Am vielfältigsten ist der Straßenverkehr. Er beinhaltet die PKWs, LKWs, Motorräder, Fahrradfahrer und Fußgänger. Diese benutzen bevorzugt angelegte Straßen und Wege, jedoch können sie auch von diesen abweichen, solange sie einen festen Untergrund haben. [Bro06]
Bei der Ortsveränderung existieren ein Startpunkt und ein Ziel. Zwischen diesen beiden verläuft eine Strecke, welche zurückgelegt wird. Es ist möglich, dass mehrere Strecken existieren. Jedoch erfolgt die Bewegung nur entlang einer von ihnen. Bei der Auswahl der Strecke spielen Aspekte wie Länge und Dauer eine Rolle. Durch Hindernisse kann sich die Strecke ändern.
Eine weitere Unterscheidung des Verkehrs erfolgt anhand des Ziels der Bewegung. Es wird eingeteilt in Berufs-, Einkaufs- und Freizeitverkehr. Der Berufsverkehr bezeichnet die An- und Abreise zum Arbeitsplatz. Meistens führt dies zu einem erhöhten Verkehrsaufkommen über einen gewissen Zeitraum. Welcher sich bei der Abreise in entgegengesetzter Richtung wiederholt. Der Einkaufs- und Freizeitverkehr zeichnen sich durch mehr Unregelmäßigkeit aus, da z.B. der Zeitpunkt des Einkaufens variieren kann. Die größte Unregelmäßigkeit besitzt der Freizeitverkehr, da in diesem fast alle weiteren Fahrten zusammengefasst sind. [Bro06]
Als öffentlicher Verkehr wird die Fortbewegung mit für die Allgemeinheit bereitgestellten Fortbewegungsmitteln bezeichnet. Hierzu zählen z.B. Züge und Busse. Sie verkehren auf der immer gleichen Strecke zu festgelegten Zeiten. Die Benutzung ist

mit Kosten verbunden, steht aber jedem zur Verfügung. Somit wird auch Personen ohne eigene Fahrzeuge ermöglicht, eine Strecke zurückzulegen. Personen mit eigenem Fahrzeug zählen in den Individualverkehr, wenn sie dieses nutzen.

2.2 Lichtsignalanlage

Eine Lichtsignalanlage, im Weiteren als LSA abgekürzt, ist eine „durch grüne, gelbe und rote Lichter [...] der Regelung des Verkehrs dienende techn. Einrichtung. Die Farbfolgen und ihre Bedeutung sind in der Straßenverkehrsordnung, die Wechsel- und Dauerlichtzeichen unterscheidet, festgeschrieben." [Bro06]. Der Wechsel der Farbfolge wird durch einzelne Phasen angegeben. „Eine Phase ist derjenige Teil eines Signalprogramms, während dessen ein bestimmter Grundzustand der Signalisierung unverändert bleibt. Die Freigabezeiten für die freigegebenen Verkehrsströme können zu verschiedenen Zeitpunkten beginnen oder enden." [fSuVAV10].
Lichtsignalanlagen lassen sich in zwei Typen einteilen. Diese Einteilung richtet sich nach der Anzahl ihrer Signale. So existiert eine Variante mit zwei Signalen, welche vorrangig für Fußgänger eingesetzt wird. Sie signalisiert entweder Rot oder Grün. Für Fahrzeuge findet häufiger eine LSA mit vier Signalen Verwendung. Die beiden zusätzlichen Signale sind Gelb und Gelb-Rot. Durch zusätzliches Blinken kann auf Gefahr hingewiesen werden. Für Fahrradfahrer ist die Signalisierung gemeinsam mit den Fußgängern oder mit den Kraftfahrzeugen zulässig. Hauptkriterium dabei ist die Lage der Fahrradspur. Existiert ein gemeinsamer Geh- und Radweg so wird die LSA der Fußgänger genutzt. Bei einer Zufahrt der Fahrradfahrer mit den Kraftfahrzeugen gelten deren Signale. Es kann jedoch auch eine gesonderte Signalisierung erfolgen. Diese wird eingesetzt wenn z.B. aufgrund einer längeren benötigten Freigabezeit eine eigene Phase erstellt werden muss. Der Abbiegevorgang kann durch grüne Pfeile gesondert erlaubt werden. Ist der grüne Pfeil fest angebracht muss der Fahrer halten und schauen ob durch das Abbiegen niemand gefährdet wird. Bei elektrischen grünen Pfeilen ist das Abbiegen gesichert und ein vorheriges Anhalten ist nicht notwendig. [fSuVAV10]
Lichtsignalanlagen regeln die Vorfahrt an Kreuzungen, dafür werden die jeweiligen Spuren mit eigenen Signalen beschaltet. Somit ist zu jedem Zeitpunkt bestimmbar welche Fahrzeuge die Kreuzung befahren dürfen bzw. vor dieser warten müssen. Um einen Signalplan zu erstellen wird zwischen verträglichen, bedingt verträglichen und nichtverträglichen Verkehrsströmen unterschieden. Letztgenannte dürfen sich nicht gleichzeitig auf der Kreuzung befinden. Ein Beispiel dafür sind sich kreuzende Verkehrsströme. Der Abbiegevorgang zählt meistens zu den bedingt verträglichen Richtungen. Daher liegt auch hier ein besonderes Augenmerk auf der Sicherheit. [fSuVAV10]
Eine weitere Aufgabe einer LSA ist die Sicherung beim Wechsel auf die gegenüberliegende Straßenseite durch Fußgänger. Neben den Kreuzungen kann dieses auch an weiteren Stellen der Straße notwendig sein. Häufig befinden sich solche Lichtsignalanlagen an Positionen an denen vermehrt Fußgänger die Straße überqueren wollen. Beispiele dafür sind im Bereich von Schulen und weiteren öffentlichen Einrichtungen zu finden. An diesen Stellen ist der Schutz der Personen besonders wichtig. Daher wird durch die LSA den Fahrzeugen signalisiert anzuhalten, bevor den Fußgängern die Freigabe erteilt wird. Dadurch wird eine Phase geschaffen, in welcher der Verkehr

auf der Straße nicht in den Bereich fahren darf, in dem sich die Personen aufhalten. An Kreuzungen sind jedoch abbiegende Spuren erlaubt, welchen meistens ein zusätzliches Signal gegeben wird, wenn Personen die Straße überqueren. Wichtig ist dabei die Länge der Phase, diese muss so gewählt sein, dass mindestens dreiviertel der Straße überquert sind. Dadurch soll gewährleistet werden, dass alle Personen es rechtzeitig auf die andere Seite schaffen. [fSuVAV10]
Neben der Vermeidung von Unfällen und dem Schutz der Fußgänger haben LSA auch eine Bedeutung bei der Steuerung des Verkehres. Durch aufeinander abgestimmte Signalisierung kann ein besserer Fluss des Verkehrs erreicht werden. Fahrzeuge erreichen die nächste LSA so, dass diese ihnen ein Weiterfahren ohne Anhalten ermöglicht. Dadurch wird stetiges Abbremsen und Anfahren vermieden, was auch den Emissionsausstoß senkt. Die dafür notwendigen LSA-Programme, welche die einzelnen Phasen, deren Reihenfolge sowie Angaben für die Phasenwechsel beinhalten, lassen sich in Festzeitprogramme und verkehrsabhängige Programme unterteilen. Festzeitprogramme folgen strikt den in ihnen festgelegtem Muster. Abweichungen treten nicht auf. Verkehrsabhängige Programme hingegen reagieren auf ihre Umgebung und den Verkehr [fSuVAV10]. Dafür werden über Taster oder z.B. Induktionsschleifen in der Fahrbahn Signale ausgesandt und die entsprechende Phase angefordert. Die LSA ändert daraufhin ihren Ablauf und schiebt die angeforderte Phase ein. Mit dieser Methode werden häufig LSA für Fußgänger ausgestattet. Ziel ist es dabei, die Sicherheit für diese zu erhöhen, ohne den Verkehr stark zu beeinträchtigen. Die LSA hat dabei den Grundzustand den Fahrzeugen Grün und den Fußgängern Rot zu signalisieren. Wird nun durch einen Passanten der Taster betätigt, wechselt das Signal für diesen auf Grün und die Fahrzeuge müssen anhalten. [For]

2.3 Simulationen

Eine Simulation ist eine „modelhafte Darstellung oder Nachbildung bestimmter Aspekte eines vorhanden oder zu entwickelnden kybernet. Systems oder Prozesses, insbesondere auch seines Zeitverhaltens.“ [Bro06]. Dabei können „Untersuchungen oder Manipulationen die am eigentl. System zu gefährlich, zu teuer oder unmöglich sind.“ [Bro06] durchgeführt werden. Das Ziel ist die „Nachbildung eines Systems mit seinen dynamischen Prozessen in einem experimentierfähigen Model, um zu Erkenntnissen zu gelangen, die auf die Wirklichkeit übertragbar sind“ [36313]. Das bedeutet, dass bei einer Simulation Eingabedaten in Ausgabedaten umgewandelt werden. Dafür wird die Realität möglichst exakt abgebildet. Ein Beispiel dafür sind Verkehrssimulationen. Je genauer das Verhalten der Verkehrsteilnehmer nachgestellt wird, desto genauer sind die Ergebnisse. Dieses Verhalten wird durch das erstellte Modell erzeugt. Auch die Qualität der Eingabedaten hat Einfluss auf die Simulation. Bei einer Verkehrssimulation werden Angaben zum Verkehrsnetz und zur Ankunft der Teilnehmer benötigt. Eine weitere Eingabe kann die Steuerung von Lichtsignalanlagen sein. Sind Fehler in den Eingaben werden diese durch das Model umgesetzt. Die entstandenen Ausgaben beruhen dann auf falschen Annahmen und geben nicht das erwartete Szenario wieder. Als Ausgabedaten bei Verkehrssimulationen können die Staulänge, der Durchlass der Straße aber auch Angaben zu Unfällen usw. erzeugt werden.
Mit Simulationen lassen sich aber auch andere Bereiche, als der Verkehr, simulieren. So werden Simulationen auch bei der Entwicklung von Baustoffen, Chemikalien

und anderen Gegenständen eingesetzt, da sich kleine Änderungen und Anpassungen schnell umsetzen lassen. Auch können Wege ausprobiert werden die in der Realität bisher nicht möglich sind. Somit können leicht für unterschiedliche Szenarien Testdaten erzeugt werden. Eine Änderung einzelner Parameter des Models stellt dabei schon ein neues Szenario dar. Ein Beispiel dafür ist eine Änderung des Verhaltens von Verkehrsteilnehmern. Dadurch verändert sich das Verhalten der gesamten Simulation, was zu neuen Daten führt. Auch Änderungen in den Eingabedaten führen zu neuen Ausgabedaten. So kann ein gesteigertes Verkehrsaufkommen, durch Änderung der Ankunft der Fahrzeuge, simuliert werden.

2.4 SUMO

SUMO (Simulation Of Urban Mobility) ist ein Simulationsprogramm, welches zurzeit vom Deutschen Zentrum für Luft- und Raumfahrt am Institut für Verkehrssystemtechnik in Berlin und Braunschweig entwickelt wird [DLRa]. Es ermöglicht die Durchführung von Verkehrssimulationen. Für die grafische Betrachtung des erzeugten Modells stellt es eine simple Oberfläche zur Verfügung. Auf dieser werden das Straßennetz sowie die verschiedenen Fahrzeugtypen abgebildet. Die Verwendung der Oberfläche ist optional, da auch eine Konsolenvariante existiert. Diese ermöglicht einen schnelleren Durchlauf der Simulation.
SUMO verwendet für die Eingabedaten das XML-Format [DLRb]. Alle benötigten Dateien müssen in diesem Format vorliegen. Für den Aufbau eines Verkehrsnetzes werden Angaben zu Knoten und Kanten genutzt. Die Knoten sind Anfangs- und Endpunkte einer Kante. Sie stellen Kreuzungen sowie Ein- und Ausgänge des Systems dar. Die Kanten sind die Straßen, welche die Knoten miteinander verbinden. Es ist möglich verschiedenste Eigenschaften, wie die Anzahl der Spuren und die erlaubte Geschwindigkeit, für die Straßen einzustellen. Auch können die erlaubten Fahrtrichtungen am Ende der Straße über die Verbindungen kontrolliert werden. Wird dazu keine Angabe gemacht erstellt SUMO diese von selbst. Jedoch sind die Kanten immer Einbahnstraßen. Für die Gegenspur muss eine eigene Kante angelegt werden. Das mitgelieferte Werkzeug NETCONVERT erzeugt aus diesen Dateien dann das zusammenhängende Verkehrsnetz. Ab Version 0.25.0 existiert die Möglichkeit der Verwendung eines Editors um das Netz zu erstellen. Bei diesem werden die Knoten gesetzt und untereinander verbunden. Für eine Verkehrssimulation werden auch Verkehrsteilnehmer benötigt. Bei SUMO können deren Eigenschaften angepasst werden. Dabei sind die Größe, die Geschwindigkeit usw. für den Fahrzeugtyp auswählbar. Auch das Verhalten der Fahrer lässt sich über Parameter beeinflussen. Beispiele hierfür sind die Reaktionszeit und die Risikobereitschaft. SUMO bietet bereits eine Vielzahl von Standartklassen für Verkehrsteilnehmer an. [DLRb]
Um Fahrzeuge in der Simulation zu erzeugen wird eine Angabe zu ihrer Ankunft benötigt. Dafür werden die Anzahl der ankommenden Fahrzeuge und ihr Typ angegeben. JTRROUTER erzeugt dann für jeden Verkehrsteilnehmer eine Route von seinem Startpunkt zu einem möglichen Endpunkt. Es wird jede Route erstellt die durch Verbindungen möglich ist. Jedoch kann auch hierauf Einfluss genommen werden, in dem vorher feste Routen definiert und den einzelnen Ankunftsströmen zugewiesen werden. Für die Strecke wird die Reihenfolge der zu befahrenden Straßen angegeben. [DLRb]

Nach Start der Simulation erlaubt SUMO, über ein Client-Server Modell, einen Eingriff in diese. Verwendete wird dabei das Traffic Control Interface (TraCI). Dieses benötigt ein zuvor erstelltes Programm, welches in Python, Java oder C++ geschrieben sein kann. Wobei C++ in Version 0.25.0 nicht vollständig unterstützt wird. Durch diesen Eingriff können Daten ausgelesen und verändert werden, während die Simulation läuft. [DLRb]
Kommt es zu langen Wartezeiten für einzelne Fahrzeuge werden diese von SUMO entlang ihrer Strecke teleportiert und an einer freien Stelle wieder eingefügt. Dieses Verhalten lässt sich abstellen. Nicht abstellbar hingegen ist, dass bei Unfällen die beteiligten Fahrzeuge ebenfalls teleportiert werden. [DLRb]

2.5 C2X

Mit Car-2-X wird die Kommunikation von Fahrzeugen mit ihrer Umgebung oder untereinander bezeichnet. Sie lässt sich daher auch in Car-2-Car (C2C) und Car-2-Infrastructure (C2I) Kommunikation aufteilen [SL]. Dabei werden Nachrichten zwischen den einzelnen Teilnehmern gesendet und empfangen. Diese Nachrichten müssen festgelegten Protokollen folgen. Als solche Protokolle existieren z.B. „die Cooperative Awareness Message (CAM), Decentralized Environmental Notification Message (DENM), Signal Phase and Time (SPaT) und die Topology Specification (TOPO)“ [SL]. In diesen sind bestimmte Information zu verschiedenen Bereichen des Verkehrs hinterlegt. Die TOPO-Nachricht enthält z.B. Angaben zu Kreuzungen. Hiermit sind vor allem topologische Gegebenheiten, wie der Verlauf der Straßen und die Anzahl der Spuren auf diesen, gemeint [Bee15]. Ziel ist die Erhöhung der Sicherheit im Straßenverkehr sowie eine Steigerung des Komforts für den Fahrer. [Gra09]
Mithilfe der C2C Kommunikation können z.B. Daten über die aktuelle Position, die Geschwindigkeit und den Zustand des Fahrzeugs an andere Verkehrsteilnehmer gesendet werden [Dre13]. Durch weitere auf C2C aufbauende Anwendungen lassen sich diese Informationen auswerten. Ein Beispiel dafür ist die Einschätzung der Gefahr bei Überholvorgängen. Bei dem dafür notwendigen Spurwechsel können schneller fahrende Fahrzeuge übersehen werden. Da dem System alle Informationen zu diesen vorliegen kann ein Warnhinweis an den Fahrer gesandt werden. Auch beim schnelleren Fahrzeug kann der Hinweis die Geschwindigkeit zu reduzieren angezeigt werden. Dadurch kann die Sicherheit für beide erhöht werden.
Bei der C2I erfolgt die Kommunikation zwischen Fahrzeug und Verkehrsanlagen. Dazu zählen Lichtsignalanlagen, elektronische Verkehrsschilder und Roadside Units [Dre13]. Eine Roadside Unit wird speziell für die C2I Kommunikation neben einer Straße installiert. Sie sammelt die von den Fahrzeugen gesendeten Daten und kann diese an eine Zentrale weiterleiten. Auch kann sie selber Daten über die allgemeine Verkehrslage senden, z.B Anzahl an Fahrzeugen und ob Hindernisse auf der Strecke sind [Dre13]. Auch Empfehlungen zu einer anderen Route, um den Verkehr zu entlasten sind möglich. LSA können ihre nächste Phase dem Fahrzeug übermitteln, wodurch dieses dem Fahrer melden kann ob er abbremsen sollte oder weiterfahren kann.

Als C2X wird die Kommunikation von Fahrzeugen, durch Protokolle, mit ihrer Umgebung bezeichnet. Ziel ist die Erhöhung der Sicherheit im Verkehr. Mit Verkehr

wird die „Ortsveränderungen von Person und Gütern“ [Bro06] bezeichnet. Unterteilt wird in Schienen-, Wasser-, Luft- und Straßenverkehr. Anwendung findet C2X vorrangig im Straßenverkehr. Um die Auswirkungen von C2X zu testen werden Simulationen durchgeführt. Bei diesen wird ein Modell entwickelt, welches die Realität abbildet. Aus Eingabedaten erzeugt dieses Modell Ausgabedaten. Es existieren verschiedene Simulationen. Eine davon ist SUMO. Mit SUMO können Verkehrssimulationen durchgeführt werden. Aus der Beschreibung des Verkehrs durch XML-Dateien werden Ausgabedaten erzeugt. Aus diesen kann die Veränderung des Verkehrs, bei Verwendung von C2X, gewonnen werden.

3. Deklarationssprache für Verkehrssimulationen

Durch das ifak Magdeburg [iMe] wurde eine Deklarationssprache entwickelt mit der Simulationen konfiguriert werden können. Das Ziel ist die einfache Erstellung von Verkehrssimulationen, um Testdaten zu erzeugen. Diese Sprache heißt TrafficSim und wurde von Sven Beckmann entwickelt. Bisher existiert eine prototypische Umsetzung für die Simulationssoftware SUMO [Bee15]. In diesem Kapitel soll ihre Funktionalität dargestellt werden. Auch werden die einzelnen Schritte des Programms bei der Erstellung einer Simulation erklärt.

3.1 Funktionalität und Eigenschaften der Deklarationssprache

Um die Konfiguration einer Simulation durch den Nutzer zu ermöglichen, müssen verschiedene Eingaben geschaffen werden. Mithilfe dieser kann in TrafficSim eine Kreuzung definiert und beschrieben werden. Auch das Einstellen der Ankunftsrate der Fahrzeuge in verschiedenen Intervallen ist möglich. Im Folgenden werden die dafür erstellten Sprachelemente aufgeführt und ihre Attribute erklärt. Jede Beschreibung mit TrafficSim beginnt mit *TrafficSimulation* [Bee15]. Es ist das umschließende Element in dem die weiteren Eingaben erfolgen. Es unterteilt sich in zwei Bereiche, zum einen die Beschreibung der Kreuzung und zum anderen Beschreibungen zum Ablauf der Simulation. Für die Definition einer Kreuzung wird das Element *Intersection* verwendet [Bee15]. Dieses besitzt folgende Attribute und Unterelemente:

- *accident*: Wahrscheinlichkeit für Unfälle. Für die Umsetzung mit SUMO ohne Auswirkungen, da Unfälle durch Sumo nicht umgesetzt werden.
- *hasTrafficLight*: Auswahl ob die Kreuzung durch eine LSA gesteuert wird. Eingaben sind „true“ oder „false“.
- *name*: Name der Kreuzung.
- *arms*: Liste mit den Armen der Kreuzung. Dafür wird das Element *Arm* verwendet.
- *VehicleDistribution*: Verteilung der Fahrzeugtypen für die Ankünfte. Angabe mehrerer ist möglich. Ist ein eigenes Element.

Innerhalb der *arms* werden die verschiedenen Fahrspuren bestimmt. Hier wird festgelegt, welche Fahrzeuge auf welcher Spur erlaubt sind. Ein *Arm* besitzt folgende zwei Eigenschaften [Bee15]:

- *name*: Der Name des *Arms*. Dieser muss eindeutig sein, um die einzelnen *Arme* zu unterscheiden.

- *lanes*: Angabe der Typen der verschiedenen Fahrspuren von rechts nach links. Trennung erfolgt durch das Zeichen „|“. Erlaubte Eingaben sind Pedestrian, Bicycle, CarIn und CarOut. Durch Pedestrian wird auf dieser Seite der Straße ein Fußweg angelegt. Bicycle erzeugt eine eigene Fahrradspur. Durch CarIn werden Spuren für die Fahrzeuge definiert. Beim Wechsel von CarIn zu CarOut wird die Fahrtrichtung gewechselt. Mit CarOut werden wegführende Spuren erzeugt. Für diese Richtung muss, falls gewünscht, die erneut die Angabe Bicycle und Pedestrian erfolgen.

In der *VehicleDistribution* kann die Verteilung der Fahrzeugtypen angegeben werden. Auf diese kann in den Simulationsschritten zurückgegriffen werden. Erlaubt sind folgende Eingaben [Bee15]:

- *name*: Name der Verteilung, dient dem späteren Aufrufen dieser und muss daher eindeutig sein.

- *bicycle*: Wahrscheinlichkeit, dass ein Fahrrad erzeugt wird, muss zwischen null und eins sein.

- *bus*: Wahrscheinlichkeit, dass ein Bus erzeugt wird. Muss zwischen null und eins sein.

- *moped*: Wahrscheinlichkeit, dass ein Moped erzeugt wird. Muss zwischen null und eins sein.

- *motorcycle*: Wahrscheinlichkeit, dass ein Motorrad erzeugt wird. Muss zwischen null und eins sein.

- *car*: Wahrscheinlichkeit, dass ein Personenkraftwagen erzeugt wird. Muss zwischen null und eins sein.

- *emergency*: Wahrscheinlichkeit, dass ein Krankenwagen erzeugt wird. Muss zwischen null und eins sein.

- *truck*: Wahrscheinlichkeit, dass ein Lastkraftwagen erzeugt wird. Muss zwischen null und eins sein.

Für die Definition von Simulationsschritten wird das Element *Step* verwendet. Dieses besitzt folgende drei Eigenschaften [Bee15]:

- *start*: Startzeitpunkt des Intervalls.

- *end*: Endzeitpunkt des Intervalls.

- *movements*: Liste mit den Ankunftszahlen für dieses Intervall. Benutzt das Element *VehicleMovement*.

Mit dem *VehicleMovement* werden Kreuzungsarm, Verteilung und Anzahl an Fahrzeugen verknüpft. Dadurch lassen sich die in der Simulation befindlichen Fahrzeuge regulieren. Dafür besitzt das Element folgende Attribute [Bee15]:

- *startArm*: *Arm* für den diese Angaben zutreffen. Der Name eines *Arms* muss angegeben werden.
- *vehiclePerHour*: Anzahl der erzeugten Fahrzeuge pro Stunde für diesen *Arm*.
- *vehicleDistribution*: Angabe einer bestimmten Verteilung der Fahrzeugtypen. Name einer *vehicleDistribution* kann angegeben werden. Bei keiner Angabe werden Standardwerte genutzt.

Mithilfe dieser Angaben kann eine Simulation konfiguriert werden. Die Form der Eingabe orientiert sich an QML und kann in Abbildung 3.1 und Abbildung 3.2 auf der nächsten Seite gesehen werden. Es kann nur eine Kreuzung beschrieben und somit erzeugt werden. Für diese lassen sich extra Fahrradspuren und Gehwege erzeugen. Auch die Anzahl an Fahrspuren für Fahrzeuge kann festgelegt werden. Dieses gilt für die zuführende und die wegführende Richtung. Es kann festgelegt werden, ob die Kreuzung durch eine LSA kontrolliert wird. Ein Einstellen dieser ist nicht möglich. Durch das Festlegen von Verteilungen und der Anzahl an erzeugten Fahrzeugen kann der Verkehr gesteuert werden. Das ist für jeden *Arm* einzeln möglich.

```
TrafficSimulation
{
    Intersection
    {
        name:"intersection 47"
        accident: 0.1
        hasTrafficLight: true
            VehicleDistribution
            {
                name:"customDistribution"
                car:0.8
                motorcycle: 0.2
            }
        arms: [
            Arm
            {
                name: "1"
                lanes: "Pedestrian|Bicycle|CarIn|CarIn|CarIn|CarOut|CarOut|Bicycle|Pedestrian"
            },
            Arm
            {
                name: "2"
                lanes: "Pedestrian|Bicycle|CarIn|CarIn|CarIn|CarOut|CarOut|Bicycle|Pedestrian"
            },
            Arm
            {
                name: "3"
                lanes: "Pedestrian|Bicycle|CarIn|CarIn|CarIn|CarOut|CarOut|Bicycle|Pedestrian"
            },
            Arm
            {
                name: "4"
                lanes:  "Pedestrian|Bicycle|CarIn|CarIn|CarIn|CarOut|CarOut|Bicycle|Pedestrian"
            }
        ]
    }
```

Abbildung 3.1: Beispiel für die Beschreibung einer Kreuzung mit TrafficSim

```
steps: [
    Step{
        start:0
        end:200
        movements: [
            VehicleMovement
            {
                startArm: "1"
                vehiclePerHour: 100
                vehicleDistribution: "customDistribution"
            },
            VehicleMovement
            {
                startArm: "2"
                vehiclePerHour: 30
            },
            VehicleMovement
            {
                startArm: "3"
                vehiclePerHour: 100
            },
            VehicleMovement
            {
                startArm: "4"
                vehiclePerHour: 30
            }
        ]
    }
]
}
```

Abbildung 3.2: Beispiel für die Konfiguration einer Simulation mit TrafficSim

Für die Deklarationssprache gelten außerdem weitere Bedingungen. Eine dieser ist das Prinzip der losen Koppelung. Dieses besagt, dass eine Beschreibung, welche für eine Kreuzung verwendet werden kann, auch auf alle Kreuzungen mit gleicher Anzahl an Armen anwendbar sein muss. Wird eine Beschreibung erzeugt, welche eine Kreuzung mit vier Armen darstellt, so kann diese auf alle realen Kreuzungen mit vier Armen angewandt werden. Durch Verwenden der Car-2-X Nachricht TOPO wird eine Anpassung an das reale Aussehen der Kreuzung erreicht. Eine so erzeugte Kreuzung unterscheidet sich nur in der Form der Straßen und der Anzahl ihrer Spuren, welche aus der TOPO-Nachricht übernommen werden. Die weiteren Angaben sind bei allen Kreuzungen mit gleicher Beschreibung identisch. [Bee15]

```
1 0 StreetA
2 1 StreetB
3 2 StreetC
4 3 StreetD
```

Abbildung 3.3: Beispiel für eine bei der Simulationserstellung genutzte Konfigurationsdatei

Über eine Konfigurationsdatei, zu sehen in Abbildung 3.3, kann die Zuordnung der realen Straße auf den beschriebenen *Arm* erfolgen. Durch die angegebene Zahl vor dem Namen wird der dazugehörige *Arm* referenziert. Dabei ist zu beachten, dass die Zahlen in der Konfigurationsdatei um eins niedriger sind. So wird durch die Null der erste definierte *Arm* angesprochen. Die Eins verweist auf den zweiten *Arm*. [Bee15] TrafficSim ist eine Deklarationssprache, welche verschiedene Elemente für die Erzeugung einer Simulation, bereitstellt. Beispiele für diese Elemente sind die *Intersection*,

die *Steps* und die *Arme*. Mit diesen kann eine Kreuzung durch den Nutzer beschrieben werden. Einen Auszug aus der formalen Beschreibung der Deklarationssprache TrafficSim ist in Quelltext 3.1 dargestellt [Bee15]. Auch der Simulationsablauf kann beeinflusst werden. Die Beschreibung muss auf alle Kreuzungen mit gleicher Anzahl an Armen anwendbar sein.

Quelltext 3.1: Formale Beschreibung von TrafficSim

```
TrafficSim = 'TrafficSim','{',Intersection, Steps,'}';
Intersection = 'Intersection','{', Name, Accident,
TrafficLight, Arms,'}';
VehicleDistribution = 'VehicleDistribution','{',Name,[Car],
[Motorcycle],[Moped],[Emergency],[Truck],[Bus],[Bicycle],'}';
Arms = 'arms',':','[',Arm,',',Arm,',',Arm,',',Arm,']';
Arm = 'Arm','{',Name,Lanes,'}';
Lanes = ['Pedestrian|'],['Bicycle|'],{'CarIn|'},{'CarOut|'},
['Bicycle|'],['Pedestrian'];
Steps = 'steps',':','[',Step,{,Step},']';
Step = 'Step',':','[',Start, End, VehicleMovements,'}';
CehicleMovements = 'movements:[',VehicleMovement,
{',',VehicleMovement},']';
VehicleMovement = 'VehicleMovement','{',StartArm,
VehiclePerHour, VehicleDistribution,'}';
```

3.2 Ablauf der Simulationserstellung

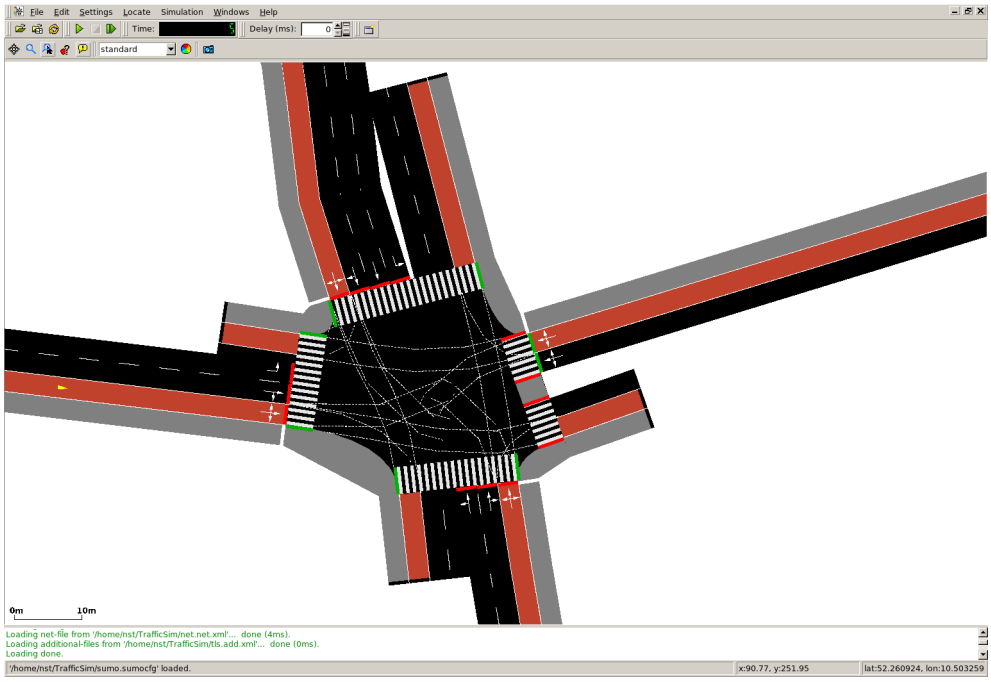

Abbildung 3.4: Beispiel für eine durch TrafficSim erzeugte Kreuzung

Um die in Abbildung 3.4 auf der vorherigen Seite zu erkennende Kreuzung mit TrafficSim zu erzeugen werden drei Eingabedateien benötigt. Das ist zum einen die TOPO-Nachricht aus der Car-2-X Kommunikation. In dieser befinden sich die Angaben zum Verlauf der realen Kreuzung. Die zweite Datei ist die in Abbildung 3.3 auf Seite 12 dargestellte Konfigurationsdatei. Mit dieser werden die Informationen aus der TOPO-Nachricht mit der aus der Kreuzungsbeschreibung verknüpft. In der dritten Datei ist die Beschreibung der Kreuzung durch die Deklarationssprache enthalten. Für die erzeugte Kreuzung wurde die Beschreibung aus Abbildung 3.1 auf Seite 11 und Abbildung 3.2 auf Seite 12 verwendet. [Bee15]
Beim Erzeugen der Kreuzung werden zuerst die einzelnen Dateien nacheinander eingelesen. Begonnen wird mit der TOPO-Nachricht. In dieser befinden sich Informationen zum Verlauf der Straßen und der Anzahl ihrer Fahrspuren. Auch welche Abbiegemöglichkeiten an der Kreuzung vorhanden sind, ist für jede Richtung enthalten. Eine Unterscheidung in hinführende und wegführende Straßen ist möglich. Beim Einlesen der Datei werden Knoten und Kanten erstellt. Durch die Knoten kann der Verlauf der Straßen nachempfunden werden. Jede Kante enthält daher die für sie geltenden Knoten. Auch ob es sich um eine eingehende oder eine wegführende Kante handelt und die Anzahl der Spuren, wird so gespeichert. Durch eine Liste an Verbindungen werden die erlaubten Abbiegemöglichkeiten für die hinführenden Kanten gesichert. Dieses erfolgt von Kante zu Kante und von Spur zu Spur. Die so erstellten Kanten sind Teil des *SumoModells* aus dem später die Simulation erzeugt wird. Ein zweites Modell ist das *TrafficSimModell*, in diesem befinden sich die Angaben aus der Beschreibung. Nach diesem ersten Schritt ist dieses jedoch noch leer. Nur im *SumoModell* befinden sich Angaben zu den Straßen der Simulation. [Bee15]
Als Nächstes wird die Konfigurationsdatei ausgewertet. Hierbei erfolgt eine Kontrolle, ob die Anzahl der Arme übereinstimmt. Ist dieses nicht der Fall, bricht das Programm ab. Beim Auslesen werden zwei verschiedene Wertepaare erstellt. Zum einen wird der Name der Position des Eintrages zugewiesen. Das heißt, dem ersten Straßennamen wird die Null zugewiesen und dem zweiten die Eins. Das zweite Wertepaar verknüpft die eingegeben Zahl und den Namen miteinander. So entstehen zwei Listen mit diesen Paaren. Durch diese Listen erfolgt später die Verknüpfung der beiden Modelle. [Bee15]
Als Letztes wird die mit TrafficSim vorgenommene Beschreibung eingelesen. In dieser befindet sich die Definition der Kreuzung durch den Nutzer sowie Angaben für den Ablauf der Simulation. Die Eingaben werden in Elemente umgewandelt und im *TrafficSimModell* abgespeichert. Dieses besteht aus einem *Intersection* Element und den *Simulationsschritten*. Das *Intersection* Element beinhaltet die Angaben zu den Armen der Kreuzung, wie Anzahl und die beschriebenen Spuren. In den *Simulationsschritten* werden die Angaben zu den definierten Intervallen und die gewünschte Anzahl an Fahrzeugen gesammelt. Nach diesem Schritt sind alle Dateien ausgelesen und ausgewertet. Es existieren zwei Modelle, das *SumoModell* und das *TrafficSimModell*. In Ersterem befinden sich Angaben zu den Straßen der Kreuzung. Das zweite Modell beinhaltet die Eingaben des Nutzers. [Bee15]
Diese beiden Modelle werden im Weiteren miteinander verbunden. Dafür werden die Informationen des *TrafficSimModells* in das *SumoModell* übertragen. Die Zuordnung der Kanten zu den Eingaben erfolgt durch die Listen aus der Konfigurationsdatei. Im *SumoModell* befinden sich die Kanten im Uhrzeigersinn angeordnet, im Norden

beginnend. Mittels der Zahl vor dem Namen wird die Position in der Liste bestimmt und der entsprechende Name den Kanten im *SumoModell* zugeordnet. Über die so zugeordneten Namen und der zweiten Liste aus der Konfigurationsdatei erfolgt die Zuordnung der *Arme* des *TrafficSimModells* zu den Kanten. Die restlichen Angaben werden einfach übertragen. Nach diesem Schritt beinhaltet das *SumoModell* alle Angaben der Eingabe und der TOPO-Nachricht. Es sind nun alle benötigten Informationen vereint. [Bee15]
Da bisher nur eine Umsetzung für SUMO vorliegt gelten die nächsten Schritte nur für diese Version. Bei Verwendung einer anderen Simulationssoftware wird gegebenenfalls ein veränderter Ablauf benötigt. Der nächste Schritt ist das Schreiben der von SUMO verwendeten Dateien. Dabei werden sechs XML-Dateien erzeugt. Als Erstes werden die Knoten geschrieben. Diese sind in nodes.nod.xml zu finden. Es werden die einzelnen Koordinaten zu jedem Knoten aufgelistet. SUMO benötigt diese, da zwischen den Knoten die Straßen verlaufen. Diese werden als Kanten in der edges.edg.xml festgelegt. Angaben zur Anzahl der Spuren und den erlaubten Fahrzeugtypen sind hier bereits enthalten. Durch die connections.con.xml werden die einzelnen Verbindungen an der Kreuzung definiert. Die Ankunftsverteilungen der Fahrzeuge für die einzelnen Arme sind in flows.flow.xml abgegeben. Diese Datei bezieht sich auch auf die vehicle.add.xml, in der die einzelnen Verteilungen der Fahrzeugtypen definiert werden. Durch die equalTurns.turns.xml werden gleiche Abbiegewahrscheinlichkeiten für die verschiedenen Fahrtrichtungen erzeugt. All diese Dateien müssen nach diesem Schritt vorliegen. [Bee15]
Um aus den Dateien ein Netz in SUMO zu erstellen wird ein Prozess von SUMO gestartet. Dieser heißt NETCONVERT. In diesem wird durch Parameter eine Glättung der Kurven vorgenommen, die Fußgängerüberwege erzeugt und die LSA erstellt. Aus den Knoten, Kanten und Verbindungen entsteht ein Netz. Um dieses mit Fahrzeugen zu füllen wird eine zusätzliche Datei benötigt. Durch den Prozess JTRROUTER wird diese erstellt. Als Eingabe dienen das Netz, die vehicle.add.xml, die flows.flow.xml und die equalTurns.turns.xml. Durch den Prozess wird festgelegt wann ein Fahrzeug in die Simulation eintritt und welche Route es verfolgt. Es werden alle durch Verbindungen möglichen Routen erstellt. Nach Abschluss des Prozesses kann die Simulation mit SUMO gestartet werden. [Bee15]
Aus der Kreuzungsbeschreibung und der TOPO-Nachricht werden zwei Modelle der Kreuzung erzeugt. Das eine Modell beinhaltet die Eingaben des Nutzers und das andere die realen Gegebenheiten der Kreuzung. Bei der Simulationserzeugung werden beide Modelle zusammengeführt und die Angaben in eine durch SUMO lesbare Form geschrieben. Mit den so erzeugten XML-Dateien kann die Simulation gestartet werden.

Mithilfe von TrafficSim kann eine Kreuzung beschrieben werden. Dafür wird das Element *Intersection* verwendet. Innerhalb von diesem können die einzelnen *Arme* definiert werden. Durch diese Eingabe wird die Kreuzung in der Simulation an die Vorstellungen des Nutzers angepasst. Da eine Beschreibung auf alle Kreuzungen mit gleicher Anzahl an Armen anwendbar sein muss, wird über eine TOPO-Nachricht die Topologie der realen Kreuzung erzeugt. Mit TrafficSim ist es daher möglich eine Simulation einer realen Kreuzung zu erzeugen.

4. Anforderungen der Erweiterung von TrafficSim um Lichtsignalanlagen

Dieses Kapitel beinhaltet die Vorüberlegungen für die Erweiterung der Konfigurationssprache TrafficSim. Dafür wird zuerst auf allgemeine Bedingungen und Anforderungen an eine LSA eingegangen. Auch die zu beachtenden Vorgaben von TrafficSim werden analysiert. Anschließend werden die durch SUMO bereitgestellten Möglichkeiten, aber auch die dadurch entstehenden Einschränkungen, betrachtet. Dieses ist notwendig, da bisher nur eine prototypische Umsetzung für SUMO vorliegt. Weil die Konfigurationssprache schon eine bestehende Form der Eingabe besitzt, wird untersucht wie eine daran angepasste Eingabe für die Steuerung einer LSA aussehen kann. Dabei werden verschieden Ansätze betrachtet und ausgewertet.

4.1 Anforderungen an die Lichtsignalanlagensteuerung

Bei der Erstellung eines Programmes für eine LSA müssen einige Punkte beachtet werden. So ist eine der Hauptaufgaben einer LSA, die Sicherheit der Verkehrsteilnehmer zu gewährleisten. Daraus ergeben sich Regeln für die Signalisierung einer LSA. So dürfen Fußgänger die Straße nur überquerenc, wenn die Fahrzeuge der kreuzenden Fahrtrichtungen angehalten haben. Parallel verlaufende Fahrspuren dürfen trotz Abbieger gleichzeitig aktiv sein. Die Fußgänger, die sich schon auf der Straße befinden, müssen die Möglichkeit haben diese vollständig zu überqueren, bevor das Signal für die Fahrzeuge auf Grün wechselt. Dafür wechselt ihr Signal frühzeitig auf Rot. Ein LSA-Programm muss diesen Übergang beachten. Auch für Fahrzeuge existiert so eine Übergangszeit, um die Kreuzung rechtzeitig zu räumen. Es ist nicht erlaubt, zwei als zueinander nichtverträglich eingestufte Fahrtrichtungen gleichzeitig das Befahren der Kreuzung zu ermöglichen. Daher muss eine der beiden Spuren mit Rot signalisiert werden. Für die Abbiegespuren ist es möglich gesonderte Freigaben zu erteilen. Auch dabei ist auf die Verträglichkeit zu achten. Eine weitere Anforderung an die Umsetzung eines LSA-Steuerprogramms ist die Einhaltung der Signalreihenfolge mit den vorgeschriebenen Zeiten. So ist die Phase in der das Signal Gelb anzeigt drei Sekunden lang, während die Gelb-Rot Dauer bei einer Sekunde liegt. Diese Übergangsphasen müssen im LSA-Programm korrekt enthalten sein. Es darf kein sprunghafter Wechsel von Grün auf Rot oder andersherum erfolgen. Ausnahme dabei sind Anlagen mit nur zwei Signalen, welche bei Fußgängern benutzt werden. [fSuVAV10]
Aber auch aus der Sprache selbst ergeben sich Anforderungen an die Erweiterung. Eine davon ist die lose Koppelung. Dabei sollen die angegebenen TrafficSim-Modelle

auf alle Kreuzungen mit gleicher Armanzahl anwendbar sein. Aus den Angaben für die LSA-Steuerung muss also auch im Fall der Anwendung auf eine andere Intersection-Nachricht ein funktionierendes LSA-Programm erstellt werden [Bee15]. Auch soll die Einstellung mehrere solcher Programme erlaubt sein. Der Nutzer muss Einfluss auf Eigenschaften, wie Dauer einer Phase und Signalisierung, haben. Daher wird eine Eingabeform benötigt in der mehrere Phasen definierbar sind und auch die einzelnen Signale festgelegt werden können. Diese Form soll sich an der bisherigen Eingabe orientieren und möglichst einfach nutzbar sein.

4.2 Möglichkeiten und Einschränkungen der Umsetzung für die Simulationssoftware SUMO

In diesem Abschnitt werden die Möglichkeiten, die von SUMO bereitgestellt werden um eine LSA-Steuerung umzusetzen, untersucht. Auch wird auf auftretende Vorgaben und Standards sowie Einschränkungen eingegangen. SUMO bietet für die Umsetzung einer LSA-Steuerung zwei verschiedene Möglichkeiten. Eine davon ist das feste Signalprogramm, welches aus einer zusätzlichen Datei beim Start ausgelesen wird. Die andere Möglichkeit ist die Beeinflussung zur Laufzeit. Dafür wird ein Programm benötigt, welches das Traffic Control Interface nutzt. Beide Varianten werden im Folgenden genauer behandelt.

4.2.1 Feste Signalprogramme

Das feste Signalprogramm wird beim Start der Simulation geladen und dann wie festgelegt durchlaufen. Dafür muss ein sogenanntes Additional-File erstellt werden und in der SUMO-Konfigurationsdatei angegeben werden. Diese Datei muss dem XML-Standard entsprechen, um von SUMO erkannt zu werden. Die Definition eines Programmes erfolgt unter dem Element *tlLogic* mit folgenden Attributen [DLRb]:

- *id*: SUMO interne Bezeichnung der Kreuzung, für die das LSA-Programm gilt.
- *type*: Angabe des Typs des LSA-Programmes. Erlaubt sind *static* und *actuated*.
- *name*: Name des LSA-Programmes, für die eindeutige Unterscheidung mehrerer Programme.
- *offset*: Dauer um die der Beginn des Programms verzögert wird.

Bei Angabe des *types static* verwendet das Programm die später in den Phasen festgelegten Zeiten, bevor die nächste Phase aktiviert wird. Es können keine Abweichungen vom Programm auftreten. Bei der Eingabe von *actuated* wird das Programm durch den Verkehr in einem kleinen Rahmen beeinflusst. Deshalb wird in den Phasen eine minimale und eine maximale Dauer abgegeben. Durch virtuelle Sensoren, die in einem gewissen Abstand zur LSA platziert werden, wird gemessen, ob ein Fahrzeug diesen Punkt passiert. Ist dieses auf einer Grün beschalteten Straße der Fall so wird die Länge der aktuellen Phase verlängert. Spätestens bei der maximalen Dauer wird geschaltet. Einfluss auf den *type actuated* kann, mithilfe der Position der Sensoren

und der benötigten Zeit zwischen zwei Fahrzeugen, genommen werden. Weitere Einstellungen sind jedoch nicht möglich. [DLRb]
Die für ein Programm benötigten Phasen stellen jeweils ein eigenes Element in *tlLogic* da. Eine *Phase* hat die Attribute [DLRb]:

- *duration*: Angabe der Dauer der Phase, wird immer benötigt und erfolgt in Sekunden.
- *state*: Angabe der Signale als Zeichenkette. Erlaubt sind die Zeichen g für Grün, r für Rot, y für Gelb, u für Gelb-Rot, o für blinkend und O für aus. Zuweisung zu Signalanlage erfolgt im Uhrzeigersinn. Anfangsarm kann variieren. Fußgängerampeln am Ende der Zeichenkette angegeben.
- *minDur*: Angabe der minimalen Dauer der Phase in Sekunden. Wird nur bei *actuated* benötigt.
- *maxDur*: Angabe der maximalen Dauer der Phase in Sekunden. Wird nur bei *actuated* benötigt.

Durch die eindeutige Kennzeichnung eines LSA-Programms durch den Namen können mehrere Programme in einer Datei definiert werden. Beachtet werden muss, dass SUMO das zuletzt erstellte Programm startet und die weiteren ignoriert. Um einen Wechsel der Programme zu erzeugen muss eine „Wochenschaltautomatik“ (*WAUT*) am Ende der Datei hinzugefügt werden. Das Element *WAUT* besitzt die Attribute [DLRb]:

- *id*: Name der Wochenschaltautomatik.
- *refTime*: Zeit bevor die Wechsel anfangen.
- *startProg*: Angabe des Programms mit dem begonnen wird.

Durch das Element *wautSwitch* werden die Wechsel genauer beschrieben. Es besitzt die Attribute [DLRb]:

- *time*: Angabe des Zeitpunktes des Wechsels.
- *to*: Angabe zu welchem Programm gewechselt wird.

Dabei muss auf den korrekten zeitlichen Ablauf geachtet werden. Damit die *WAUT* die richtige Kreuzung beeinflusst wird das Element *wautJunction* mit folgenden Attributen verwendet [DLRb]:

- *wautID*: Name der Wochenschaltautomatik. Entspricht der ID der *WAUT*.
- *junctionID*: SUMO interne Bezeichnung der Kreuzung. Entspricht der ID der *tlLogic*.

- *procedure*: Methode die beim Wechsel der Programme verwendet wird. Mögliche Eingaben sind *GSP*, *JustSwitch* und *Stretch*.
- *synchron*: Angabe ob der Wechsel synchron erfolgen soll. Erlaubt sind die Eingaben *true* und *false*.

Durch die Attribute *procedure* und *synchron* kann Einfluss auf die Art des Umschaltens genommen werden. So existieren die Varianten, direkt beim Erreichen des Zeitpunktes zu schalten oder auf das Ende der Phase zu warten, um mit einer ähnlichen Phase das neue Programm zu beginnen. [DLRb]
Durch die beschriebenen Elemente ermöglicht SUMO die Erzeugung von LSA-Programmen. Über die Attribute dieser Elemente können Einstellung an der LSA vorgenommen werden. Die so erzeugten Programme können während der Simulation nicht beeinflusst werden. Die Erweiterung von TrafficSim muss, bei der Umsetzung für SUMO, LSA-Programme mit diesen Elementen und Attributen erzeugen.

4.2.2 Beinflussbare Signalprogramme

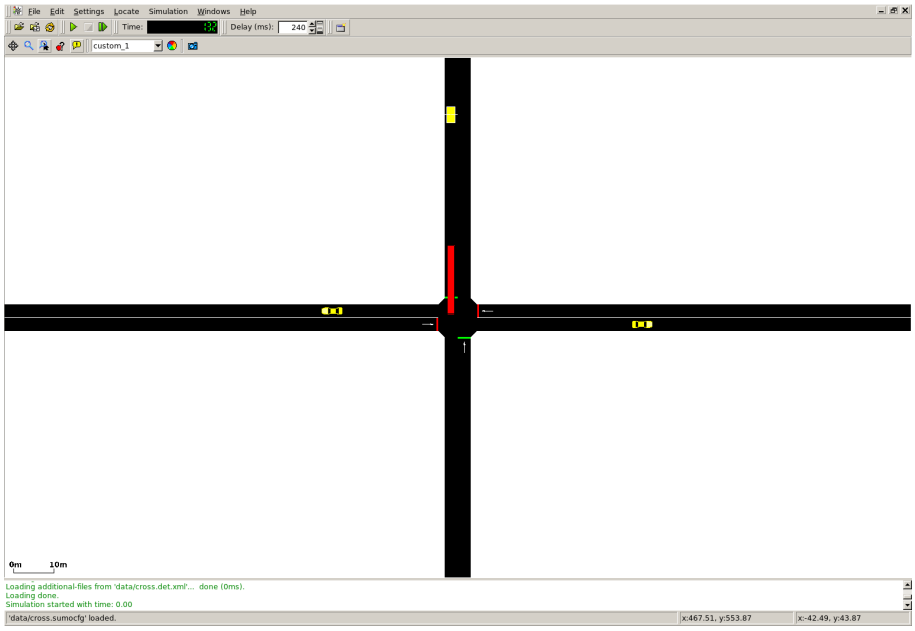

Abbildung 4.1: Kreuzung des Beispiels aus SUMO für TraCI [DLRb]

Das feste Signalprogramm bietet nur wenig Einflussnahme auf die LSA nach dem die Simulation gestartet wurde. Mehr Einfluss kann durch das Traffic Control Interface erreicht werden. Dafür wird ein Programm in Python, Java oder C++, in Version 0.25.0 nicht vollständig integriert, geschrieben [DLRb]. Dieses verwendet eine Client-Server Struktur, um über einen Port mit der Simulation zu kommunizieren. Dabei können Informationen über Verkehrsteilnehmer (Anzahl) aber auch über die Simulation (Zeit) selbst sowie ihrer Straßen (Länge) ausgelesen werden. Als Hilfe dafür dienen Detektoren, von denen SUMO drei beinhaltet. Einer dieser Detektoren ist die Induktionsschleife [DLRb]. Mit ihr können Daten über die Fahrzeuge gewonnen werden, die den Ort der Schleife überquert haben. Jedoch können Informationen nicht nur ausgelesen werden sondern auch verändert. Im SUMO Beispiel TraCi4Traffic

Lights wird die LSA manipuliert, um ausgewählten Fahrzeugklassen Vorfahrt zu gewähren [DLRb]. In dem Beispiel, zu sehen in Abbildung 4.1 auf der vorherigen Seite, unterteilt sich das Netz in eine vertikale und eine horizontale Straße. Beide treffen sich an einer von einer LSA gesteuerten Kreuzung. Im Beispiel fahren auf der horizontalen Straße nur normale PKW und auf der vertikalen die bevorzugten Fahrzeuge, z.B. Krankenwagen. Die LSA ist so eingestellt, dass sie den PKW dauerhaft Grün zeigt. Überquert nun ein Krankenwagen einen Detektor fragt das Programm welchen Zustand die LSA aktuell hat. Ist für den Krankenwagen Rot angezeigt so wird die Phase auf Grün gewechselt. Ist es bereits Grün so wird die Phase nur neu gestartet, damit die Grünphase verlängert wird. Somit kann nach dem Start der Simulation noch Einfluss auf diese genommen werden. [DLRb]

Mit SUMO lassen sich sowohl feste als auch beeinflussbare Signalprogramme simulieren. Für die Beschreibung eines festen LSA-Programms werden verschiedene Elemente und ihre Attribute verwendet. Dauer und Signale einzelner Phasen lassen sich einstellen. Beeinflussbare LSA-Programme können durch das Traffic Control Interface erstellt werden. Dabei kann über Sensoren der Zustand der Simulation bestimmt werden. Eine Veränderung der LSA aufgrund dieser Informationen ist möglich. Bei der Erweiterung von TrafficSim wurden jedoch nur feste Signalprogramme umgesetzt, da kein weiteres Programm erzeugt werden sollte. Dieses wäre jedoch für die Beeinflussung der Simulation zur Laufzeit notwendig gewesen. Die Erweiterung muss auf die beschriebenen Elemente und deren Attribute eingehen, da ansonsten keine Umsetzung für SUMO möglich wäre.

4.3 Möglichkeiten zur Spracherweiterung

Dieser Abschnitt behandelt verschieden Ansätze zur Erweiterung der TrafficSim-Sprache um eine Steuerung der Lichtsignalanlagen. Dafür werden die verschiedenen Varianten erläutert und ihre Vorteile und Nachteile analysiert. Ziel ist es, dass die Eingabe leicht zu verstehen und umzusetzen ist. Außerdem soll die Form der anderen Eingaben nachgebildet werden. Durch nur wenige Angaben soll sich ein LSA-Programm erstellen lassen. Dabei kann trotzdem Einfluss auf den genauen Ablauf der Steuerung genommen werden. Dies beinhaltet die Dauer von Phasen und die Signalisierung für die Fahrzeuge.

4.3.1 Kurze Variante

Durch eine kurze Variante kann nur Einfluss auf die Dauer der Grünphasen genommen werden. Es besteht keine Möglichkeit selber Einfluss auf die Signalisierung zu nehmen. Der Vorteil dieser Möglichkeit ist ihre Einfachheit. Sie dient dem schnellen Anpassen einer LSA-Steuerung, wenn das genaue Programm nicht weiter relevant ist. Durch nur wenige Eingaben kann ein neues Ampelprogramm erzeugt werden. Der Nachteil liegt jedoch darin, dass die genauere Signalisierung nicht verändert werden kann. Es wird immer dasselbe Programm, nur mit unterschiedlicher Länge einzelner Phasen, erzeugt. In diesem LSA-Programm erhält jede Spur eines Armes dasselbe Signal. Des Weiteren stellt das Programm sicher, dass jedes Signal mindestens einmal Grün anzeigt. Die Abbildung 4.2 auf der nächsten Seite zeigt die hier beschriebene kurze Variante. Dabei wird ein neues Element *trafficLight* in die Sprache integriert.

```
TrafficSimulation
{
    Intersection
    {
        name:"intersection 47"
        accident: 0.1
        hasTrafficLight: true
            VehicleDistribution
            { ...}
        arms: [
            Arm
            { ...
            },
            Arm
            { ...
            },
            Arm
            { ...}
        ]
        trafficLights: [
            TrafficLight
            {
                duration: 10
                name: "test4"
                type: "static"
                turns: 10
            }

        ]
    }
```

Abbildung 4.2: Beispiel für die kurze Variante zum Beschreiben eines LSA-Programms

Da die LSA immer eindeutig einer Kreuzung zugeordnet werden muss wird dieses Element als neues Attribut in die *Intersection* aufgenommen [DLRb]. Durch die Eingabe als Liste ist es möglich mehrere Programme zu definieren. *TrafficLight* selbst besitzt folgende Attribute:

- *duration*: Dauer der Phasen in Sekunden, beeinflusst nur die Dauer von Grünphasen, Übergangsphasen werden nicht verändert.
- *name*: Name des LSA-Programms, dieser muss eindeutig gewählt werden, spätere Identifizierung verschiedener Programme erfolgen über diesen.
- *type*: Gibt den für SUMO benötigten Typ des LSA-Programms an, erlaubt sind nur *static* und *actuated*.
- *turns*: Anzahl wie oft ein LSA-Programm durchlaufen wird, bevor ein Programmwechsel stattfindet. Diese Angabe ist optional. Der Standardwert ist Eins.

Die kurze Variante ist eine erste Form der Spracherweiterung von TrafficSim. Sie erlaubt nur einen geringen Einfluss auf das LSA-Programm. Durch die Eingabe eines einzelnen Elements und verschiedener Attribute stellt sie eine schnelle Variante der LSA-Steuerung da.

4.3.2 Ausführliche Variante

Die ausführliche Variante stellt eine Erweiterung der kurzen Variante dar. Sie dient dem genaueren Anpassen der LSA. Dafür werden mehr Eingaben des Nutzers benötigt. Dieser kann somit auch einen größeren Einfluss auf das erstellte Programm nehmen. Er hat Zugriff auf die Gestaltung einzelner Phasen und die Möglichkeit beliebig viele solcher zu erstellen. Der Sinn eines erzeugten Programms hängt jedoch direkt von der Eingabe ab. So können auch Konflikte, z.B. alle Spuren haben Grün, erzeugt werden. Auch die getrennte Signalisierung von Abbiegern ist einstellbar.

```
trafficLights: [
    TrafficLight
    {
        name: "test"
        type: "actuated"
        turns: 10
        phases: [
            Phase
            {
                duration: 17
                minDur: 16
                maxDur: 18
                arms: [
                    Arm
                    {
                        name: "1"
                        state: "red"
                        rightTurn: "green"
                        leftTurn: "green"
                    },
                    Arm
                    { ...
                    },
                    Arm
                    { ...}

                ]
            },
            Phase
            { ...]
```

Abbildung 4.3: Beispiel für die ausführliche Variante zum Beschreiben eines LSA-Programms

In Abbildung 4.3 ist die Erweiterung des Elements *trafficLight* um eine Liste mit Phasen zu sehen. Für *trafficLight* sind, mit Ausnahme der *duration*, dieselben Eingaben wie bei der kurzen Variante gestattet. Die *duration* ist nicht mehr erlaubt, da diese in den Phasen angegeben wird. Die *Phasen* sind ein neues eigenständiges Element, welche folgende Attribute enthalten:

- *duration*: Dauer der einzelnen Phase in Sekunden.
- *minDur*: Minimale Dauer dieser Phase in Sekunden wenn bei *type actuated* ausgewählt wurde, für *static* keine Bedeutung.
- *maxDur*: Maximale Dauer dieser Phase in Sekunden wenn bei *type actuated* ausgewählt wurde, für *static* keine Bedeutung.

- *factor*: Dient der Berechnung der minimalen und maximalen Dauer aus der *duration*. Angabe erfolgt zwischen 0 und 100. Stellt Alternative zu *minDur* und *maxDur* dar.

Diese gelten diesmal nur für die beschriebene *Phase*. Somit benötigt jede *Phase* ihre eigene Angabe dieser Werte. Durch eine Liste von *Armen* wird auf die bereits in der *Intersection* definierten *Arme*, über ihren Namen referenziert. Durch die folgenden zusätzlichen Attribute wird die Signalisierung bestimmt:

- *state*: Angabe des Hauptsignals für diesen *Arm*. Erlaubte Eingaben sind *green* und *red*.
- *rightTurn*: Angabe des Signals für Rechtsabbieger auf diesem *Arm*. Diese Angabe ist optional. Erlaubte Eingaben sind *green* und *red*.
- *leftTurn*: Angabe des Signals für Linksabbieger auf diesem *Arm*. Diese Angabe ist optional. Erlaubte Eingaben sind *green* und *red*.

Die Angabe von Übergangsphasen ist nicht notwendig, da diese automatisch erzeugt werden. Für jeden Arm der Kreuzung werden Angaben benötigt. Es können beliebig viele solcher *Phasen* innerhalb eines *trafficLights* definiert werden. Auch das Erstellen mehrerer Programme ist weiterhin möglich sowie eine Mischung aus der kurzen und der ausführlichen Variante.
In Quelltext 4.1 sind die Erweiterungen von TrafficSim formal aufgelistet. Erkennbar sind die neuen Elemente *TrafficLight* und *Phase*. Mit diesen können mehrere LSA-Programme beschrieben werden.

Quelltext 4.1: Formale Beschreibung Erweiterung

```
TrafficLights = 'trafficLights',':','[',TrafficLight,
','TrafficLight,']';
TrafficLight = 'TrafficLight','{',Duration,Name,
Type,Turns,Phases,'}';
Phases = 'phases',':','[',Duration,MinDur,
MaxDur,Factor,Arms,']';
Arm = 'Arm','{',Name,Lanes,state,rightTurn,leftTurn'}';
```

4.3.3 Mögliche weitere Varianten

Eine nicht implementierte Überlegung war die Vermeidung des Referenzierens der *Arme*. Dies sollte durch Eingabe der Zustände in den schon definierten *Armen* geschehen. Dabei wären wie für die *lanes* eine hintereinander folgende Auflistung der Zustände, getrennt durch das Trennsymbol |, notwendig gewesen. Dazu wäre unter dem Element *trafficLight* eine gleich aufgebaute Liste mit der Dauer einer jeden Phase zu erstellen nötig gewesen. Diese hätte dieselbe Anzahl an Einträgen wie die Zustände haben müssen, damit eine Zuordnung von Dauer zu Phase erfolgen hätte können. Diese Variante wurde jedoch nicht implementiert, da die Art der Eingabe

nicht intuitiv ist. Außerdem sollte die Eingabe der Zustände und der Dauer nicht getrennt werden. Ebenfalls hätte es durch die benötigten gleichen Längen der Listen, schneller zu Fehlern durch den Nutzer kommen können. Auch wäre die Änderung einzelner Phasen verkompliziert worden, da die jeweilige Position gefunden werden müsste. Ein Vorteil wäre die Reduzierung der benötigten Wörter, um ein Programm zu erstellen. Da die *Arme* bereits definiert werden müssen, wäre eine abermalige Angabe dieses Elements nicht mehr notwendig gewesen.
Eine weitere Überlegung war, die Arme, welche immer dasselbe Signal haben, für die Phase gemeinsam anzugeben. Dafür wäre ein Attribut nötig gewesen, in dem die zusammengehörenden Arme aufgelistet worden wären. Diesen Armen wäre ein Signal zugeordnet worden. Alternative hätte auch in den schon definierten Armen ein Attribut, z.B. als Zahl, für die Verknüpfung sorgen können. Dann wären diese Verbindungen aber für alle Programme und alle Phasen gültig. Da diese Variante aber keine genaueren Einstellungen zulässt und nur schwer zu durchschauen ist, wurde sie nicht implementiert.
Auch das Einfügen von Parametern für den *type actuated* wurde nicht umgesetzt, da nur wenige solcher Parameter existieren. Eine Definition des *trafficLight* Elementes außerhalb der *Intersection* wurde nicht weiter verfolgt, da für die LSA ein starker Bezug zur Kreuzung besteht.

Bei der Erstellung einer LSA-Steuerung gelten verschiedene Bedingungen, die eingehalten werden müssen. Diese ergeben sich aus der Straßenverkehrsordnung und den Aufgaben einer LSA. Hierzu zählen die Sicherheit der Verkehrsteilnehmer zu gewährleisten und der Übergang bei einem Phasenwechsel. Bei der Erweiterung von TrafficSim muss auf diese Vorgaben geachtet werden. Auch für die Eingabe durch den Nutzer musste ein Konzept erarbeitet werden. Dabei wurden zwei Variante erstellt. Die kurze Variante bietet die Möglichkeit schnell Veränderungen an der LSA vorzunehmen. Da ihr Einfluss aber beschränkt ist wurde alternativ eine ausführliche Variante entwickelt. Mit dieser können einzelne Phasen beschrieben und somit komplexe LSA-Steuerungen umgesetzt werden. Beide Varianten wurden erstellt um verschiedene Ansprüche an Genauigkeit und Einfachheit der Eingabe zu bedienen.

5. Praktische Umsetzung

Dieses Kapitel beschäftigt sich mit der Implementierung der Varianten zur Erweiterung der Konfigurationssprache TrafficSim. Es wird der Ablauf des Programms beschrieben und erläutert warum dieser Weg gewählt wurde. Dabei wird auf neue Klassen und Änderungen bereits existierender Klassen eingegangen. Zuerst wird die Erweiterung der Sprache, also die dem Nutzer zur Verfügung stehenden Elemente, betrachtet. Anschließend werden die Änderungen in den Parserfunktionen erläutert. Bevor auf die eigentliche Generierung der LSA-Steuerung eingegangen wird.

5.1 Implementierung der Spracherweiterung

Für die Umsetzung der Eingaben muss die Deklarationssprache um zwei Sprachklassen erweitert werden. Das wird notwendig, da mit dem *TrafficLight* und den *Phasen* zwei neue Objekte hinzukommen sollen. Jedes dieser Objekte wird durch eine Klasse beschrieben. In diesen Klassen sind die Eigenschaften und Funktionen enthalten. *TrafficLight* besitzt als Eingaben den Namen und den Typ des LSA-Programms, diese werden als Text angegeben. Der Name dient der Identifizierung des Programms. Für die Dauer sind gebrochene Zahlen erlaubt. Diese Angabe wird nur für die kurze Variante benötigt. Die Angabe für die Anzahl der Durchläufe erfolgt ganzzahlig. Die letzte Eigenschaft ist eine Liste von *Phasen*. In dieser werden die angegebenen Phasen aufgenommen. Durch die Eingabe einer *Phase* wird auf die ausführliche Variante gewechselt. Die Eingabe mehrere *Phasen* ist erlaubt.
Die zweite neue Klasse ist die *Phase*. In ihr kann eine maximale und minimale Dauer sowie die allgemeine Dauer dieser Phase festgelegt werden. Wie beim *TrafficLight* ist die Verwendung von gebrochenen Zahlen erlaubt. Der Faktor wird ganzzahlig angegeben. Mit ihm sollen Prozente ausgedrückt werden, um die maximale und minimale Dauer einer Phase aus der allgemeinen Dauer zu berechnen. Wichtig für die *Phase* ist die Liste der *Arme*. Diese ermöglicht die Angabe von *Armen* in den einzelnen *Phasen* und greift auf die dafür bereits existierende Klasse zurück. Über den Namen wird der jeweilige *Arm* referenziert. Es müssen alle Arme der Kreuzung referenziert werden.
Für die Verwendung von LSA müssen die Arme erweitert werden. Da die Signalisierung für jeden *Arm* angeben werden soll, erfolgt die Eingabe auch über diese. Es existieren bereits die Eigenschaften name und lanes. Zusätzlich erhält diese Klasse nun state, leftTurn und rightTurn. Diese beinhalten den geltenden Zustand der LSA. Durch leftTurn werden die linksabbiegenden Fahrzeuge und durch rightTurn die rechtsabbiegenden Fahrzeuge gesondert gesteuert. Es ist pro Phase und Arm nur ein Zustand erlaubt.
Durch die Erweiterung der *Intersection* um eine Liste von *TafficLights*, ist die Eingabe mehrerer LSA-Programme umgesetzt.

5.2 Implementierung der Parsefunktionen

Die mithilfe der Spracherweiterung erzeugten Eingaben müssen anschließend geparst werden. Die Daten müssen in das TrafficSim-Modell und das SUMO-Modell übertagen werden. Genutzt wird dafür der TrafficSimParser. Dieser erstellt aus allen Eingaben das TrafficSim-Modell. In diesem wird die Beschreibung der Kreuzung, mit allen Eingaben, gespeichert. Aufgrund der Erweiterung der Sprache muss auch der Parser erweitert werden. Zwei neue Funktionen zum Umwandeln des *TafficLights* und der *Phasen* wurden implementiert. Außerdem wurden beide Modelle um diese Klassen erweitert. Die Klassen besitzen dieselben Eigenschaften wie die Eingabe. Beim Parsen der *TafficLights* wird zwischen der kurzen und der ausführlichen Variante unterschieden und ein entsprechender Wahrheitswert gesetzt. Bei der ausführlichen Variante wird, im Gegensatz zur kurzen Variante, die Dauer auf null gesetzt, da diese in den jeweiligen Phasen angegeben ist. Die weiteren Eingaben werden übernommen.
Für die ausführliche Variante werden anschließend die *Phasen*, mit den jeweiligen Informationen, geparst. Die erzeugten Objekte werden jedoch nicht innerhalb des *TafficLights*, sondern als Teil der *Arme* gespeichert. Diese Implementierung wurde gewählt, da die *Arme* bereits erzeugt sind und somit die jeweiligen Signale direkt dem zugehörigen Arm zugeordnet werden. Diese Zuordnung hätte durch das Erzeugen der *Phasen* innerhalb des *TafficLights* vermieden werden können. Dies hätte jedoch dazu geführt, dass in jeder *Phase* für jeden *Arm* ein neues Objekt hätte erzeugt werden müssen. Des Weiteren soll das *TafficLight* nur die endgültigen Phasen enthalten, diese müssen aber erst aus den Eingaben erzeugt werden. Da alle *Phasen* innerhalb einer Liste stehen, musste der Name des LSA-Programms zugeordnet werden. Über diesen kann die *Phase* wieder dem jeweiligen Programm zugeordnet werden.
Nach diesem Schritt sind alle Eingaben im TrafficSim-Modell vereint. Da aber die XML-Dateien aus dem SUMO-Modell erstellt werden, müssen die Informationen an dieses weitergegeben werden. Dafür werden neue Objekte des SUMO-Modells erzeugt. In diese werden die Informationen übertragen. Für die LSA-Programme müssen dabei keine Daten aus der TOPO-Nachricht berücksichtigt werden. Daher können neue *TafficLight*- und *Phasen*objekte erzeugt und die Informationen des TrafficSim-Modells übernommen werden.
Nach dem Parsen enthält das Sumo-Modell alle Angaben aus der Beschreibung der Kreuzung und der TOPO-Nachricht.

5.3 Generierung der Lichtsignalanlagensteuerung

Nach Beendigung des Parsens enthält das SUMO-Modell alle, bis auf die zu erzeugenden LSA-Programme, relevanten Informationen für die Simulation. Diese werden in Form von XML-Dateien für SUMO lesbar gemacht. Nach dem Schreiben der Dateien startet der Prozess NETCONVERT. Dieser wandelt die Knoten und Kanten in ein Verkehrsnetz um. Dieses wird als neue XML-Datei gespeichert. In dieser befindet sich ein durch SUMO erzeugtes Programm für die LSA-Steuerung. Aus diesem müssen Informationen gewonnen werden, die für die Erstellung eigener LSA-Programme notwendig sind. Zum Auslesen wurde eine neue Klasse XML-Reader implementiert. Dabei wird im ersten Schritt die Datei mit dem Verkehrsnetz eingelesen. Für jedes

LSA-Programm wird die anzusprechende Kreuzung benötigt. Der Name dieser kann aus dem LSA-Programm von SUMO entnommen werden. Dafür wird nach dem Element *tlLogic* gesucht und diese Elemente in einer Liste aufbewahrt. Für das erste Element wird der Name der Kreuzung ausgelesen, welcher anschließend allen *TrafficLight*objekten zugeordnet wird.
Im nächsten Schritt folgt die Unterscheidung der beiden Varianten. Bei der kurzen Variante ist es notwendig die Phasen des Beispielprogramms vollständig auszulesen. Dafür wird das Dokument nach dem Element *phase* durchsucht. Da für eine Kreuzung immer nur ein LSA-Programm durch Sumo erzeugt wird, befinden sich anschließend alle benötigten Phasen in einer Liste. Für diese Phasen werden ihre Dauer und ihr Status ausgewertet. Über die Länge der Phase wird eine Unterscheidung in Haupt- und Übergangsphasen getroffen. Die Hauptphasen dauern dabei deutlich länger an. Während die Übergangsphasen mitsamt ihrer Dauer und ihres Zustandes übernommen werden, wird die Dauer bei den Hauptphasen auf den Wert des Nutzers gesetzt. Das *Phasen*objekt wird anschließend an die Liste der *Phasen* für das LSA-Programm angehängt. Nach dem Auslesen aller Phasen ist das LSA-Programm für die kurze Variante fertiggestellt.
Für das Erzeugen eigener Phasen aus den Eingaben werden weiterer Informationen benötigt. Diese sind ebenfalls in der XML-Datei enthalten. Da die von SUMO ausgewertete Zeichenkette mit den Signalangaben die Fußgängerüberwege beinhaltet, muss die Anzahl dieser den einzelnen Armen zugeordnet werden. Dafür wird das Attribut *crossingEdges* ausgewertet. Die Anzahl der Signale ergibt sich aus der Anzahl an Zeichen des *state* aus dem Beispielprogramm. Um den Abbiegerichtungen eine eigenen Signalisierung zu ermöglichen muss ihre Position in der Signalfolge bekannt sein. Dafür kann aus der XML-Datei die Richtung und Position der Verbindungen ausgelesen werden. Diese Informationen werden unter den Verbindungen des Kreuzungsmodells hinzugefügt.
Mithilfe dieser Informationen können die Phasen erzeugt werden. Über den Namen der *Phase* wird überprüft, ob diese zum aktuellen LSA-Programm gehört. Dadurch wird sichergestellt, dass alle Phasen, die zu dem Programm gehören, erzeugt werden. Ist die *Phase* Teil des Steuerprogramms, so wird der Zustand für jeden *Arm* aufgerufen und die entsprechenden Stellen in der Zeichenkette gesetzt. Bei Angabe eines leftTurns oder rightTurns wird überprüft, ob ein anderer Zustand für diese gelten soll. Ist dies der Fall, so werden die Stellen der Links- oder Rechtsabbieger herausgesucht und ihr Wert verändert. Mit der angegebenen Dauer sowie der maximalen und minimalen Dauer wird ein Phasenobjekt erstellt, welches dem LSA-Programm hinzugefügt wird. Durch eine Überprüfung der folgenden Phase werden die Übergangsphasen erstellt. Erfolgt kein Wechsel des Signals so bleibt dieses im aktuellen Zustand. Erfolgt ein Signalwechsel werden die Übergänge gemäß den Verkehrsregeln erstellt. Das bedeutet, dass auf Grün Gelb und dann Rot folgt. Auf Rot folgt Rot und anschließen Gelb-Rot. Die erste dieser Übergangsphasen ist drei Sekunden lang. Die zweite dauert eine Sekunde. Es werden ebenfalls *Phasen*objekte erzeugt und dem *TrafficLight* hinzugefügt.
Nachdem Erzeugen aller *Phasen* und *TrafficLights* müssen diese in eine XML-Datei geschrieben werden. Die dafür vorgesehene Klasse XML-Writer muss erweitert werden. Begonnen mit dem Schreiben wird mit der *tlLogic*. In dieser müssen die Kreuzungs-ID und der Typ angegeben werden. Innerhalb der *tlLogic* werden die einzelnen Pha-

sen geschrieben. Beim Schreiben der LSA-Programme wird die umgekehrte Reihenfolge zur Eingabe verwendet, da SUMO das zuletzt geschriebene Programm zuerst ausführt. Somit wird sichergestellt, dass beim Start der Simulation das korrekte Steuerprogramm für LSA verwendet wird.
Ein Wechsel zwischen den Programmen soll ermöglicht werden. Dafür wird eine weitere Funktion des XML-Writers implementiert. Diese schreibt die switches zwischen den LSA-Programmen. Jedoch erfolgen Wechsel nur bis zum Ende des letzten angegeben Intervalls. Danach wird das zu diesem Zeitpunkt aktive Programm fortgesetzt bis die Simulation endet. Auch die Switchfunktion muss auf die Reihenfolge der LSA-Programme achten. Die Eingabe wird nacheinander aufgerufen. Um den Anfangspunkt des folgenden Programms zu bestimmen, wird auf den Startzeitpunkt des aktuellen Programms die Dauer von diesem multipliziert und mit der Anzahl an Durchläufen addiert. Ist bereits das zuerst definierte Programm länger als die Simulation so werden keine switches erzeugt.
Alle Informationen werden in ein additional-file geschrieben, welches in der SUMO-Konfiguration angegeben wird. Das durch SUMO erstellte Beispielprogramm wird dadurch nicht ausgeführt.

Für die Umsetzung der Erweiterung wurden zwei neue Sprachelemente, das *TrafficLight* und die *Phase*, geschaffen. Auch die beiden Modelle, das TrafficSim-Modell und das SUMO-Modell, wurden um diese Objekte erweitert. Um die Eingaben des Nutzers zu übertragen, wurde der Parser um entsprechende Funktionen ergänzt. Dabei werden sind die Phasen, im Gegensatz zur Eingabe, aber nicht innerhalb des *TrafficLights* angelegt, sondern als Teil der Arme. Dadurch enthält jeder Arm die für ihn geltenden Signale. Aus diesen werden die endgültigen Phasen erzeugt, welche dann Teil des *TrafficLights* sind. Somit enthält das *TrafficLight* alle zu schreibenden *Phasen*.

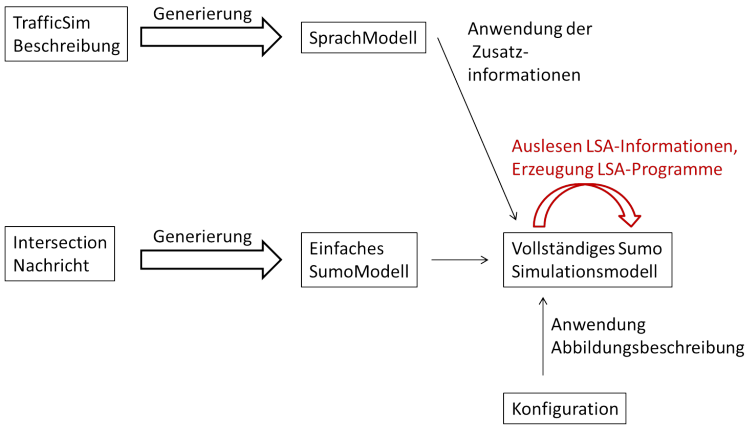

Abbildung 5.1: Ablauf der Simualtionserzeugung durch TrafficSim

Durch das Erzeugen der Phasen verändert sich der Ablauf der Simulationserzeugung. Diese Veränderung ist in Abbildung 5.1 dargestellt. Die ursprüngliche Abbildung mit dem bisherigen Ablauf, in Schwarz dargestellt, wurde durch Sven Beeckmann erstellt [Bee15]. Für das Erstellen der Phasen werden Informationen aus den erzeugten Dateien benötigt. Daher ist ein weiterer Schritt zum Auslesen diese Informationen und

der anschließenden Phasenerzeugung notwendig. In der Abbildung 5.1 auf der vorherigen Seite ist dieser Schritt in Rot dargestellt.

6. Experimente und Verifikation

In diesem Kapitel soll die Erweiterung der Konfigurationssprache TrafficSim überprüft und getestet werden. Dafür werden drei verschiedene, in der Realität existierende, Kreuzungen untersucht. Mit der K10 und der K47 sind zwei vierarmige und mit der K51 eine fünfarmige Kreuzung enthalten. Diese wurden schon von Sven Beeckmann zum Experimentieren verwendet [Bee15]. Ziel ist es, die Erweiterung von TrafficSim auf ihre Korrektheit zu untersuchen. So muss ein lauffähiges LSA-Programm erstellt werden, welches die Vorgaben umsetzt. Dafür wird das Ergebnis der Phasenerzeugung, ebenso wie die Fehlererkennung, überprüft. Zum Schluss wird versucht ein real existierendes LSA-Programm nachzubilden.

6.1 Verifikation der Korrektheit der Phasenerzeugung

Bei der Verifikation der Phasenerzeugung wurden für die drei Kreuzungen jeweils die kurze, die ausführliche und eine Mischung der beiden Varianten getestet. Überprüft wurde, ob die Simulation mit dem erzeugten LSA-Programm in SUMO fehlerfrei startet. Dadurch ist sichergestellt, dass ein von SUMO erkanntes Programm erstellt wurde. Jedoch sind hierdurch noch keine Aussagen zum Inhalt der LSA-Steuerung gegeben. Dieser Test dient nur dem Erkennen von schwerwiegenden Fehlern. Deswegen wurde im nächsten Schritt die Startphase überprüft.
Hierfür wurden die Angaben mit der zu erkennenden Signalisierung abgeglichen. Durch diesen Test kann gezeigt werden, dass für die erste Phase die Zuordnung auf die einzelnen Arme und das Schreiben der korrekten Position für die einzelnen Signale funktioniert. Da hiermit nur eine Phase abgedeckt ist, wurde dieser Abgleich für die weiteren Phasen wiederholt.
Auch die erzeugte Datei selbst wurde untersucht. So muss die Reihenfolge der LSA-Programme umgekehrt zur Eingabe erstellt werden. Durch den Namen der Programme lässt sich dieses leicht nachvollziehen. Auch wurden die Übergangsphasen dadurch kontrolliert, dass z.B. auf Grün immer Gelb und dann Rot folgen muss. Es existiert also eine feste Reihenfolge, die nachvollzogen werden kann. Da auch die Übergänge der Programme in dieser Datei enthalten sind, konnten auch diese überprüft werden. Dieses geschah durch einen Vergleich der Zeitpunkte des Wechsels mit der Dauer des Programms und der Anzahl an Durchläufen. Ebenfalls wurde kontrolliert, ob die Reihenfolge der Programmaufrufe mit der der Eingabe übereinstimmt. Durch diese Vergleiche und Überprüfungen soll sichergestellt werden, dass die XML-Datei und ihre LSA-Programme die Eingaben korrekt umsetzen. Als Ausgaben zum Test der laufenden Simulation wurden die Wechselzeitpunkte der einzelnen Phasen sowie zwischen den LSA-Programmen gewählt. Durch die erzeugten Dateien kann nachvollzogen werden zu welchem Zeitpunkt welche Signalisierung erfolgte. Da für

jede der drei Kreuzungen mindestens die drei beschriebenen Varianten getestet wurden, soll hier nur ein Beispiel für die K47 gezeigt werden. Dafür wurde die gemischte Variante ausgesucht. Die beiden weiteren Kreuzungen befinden sich im Anhang.

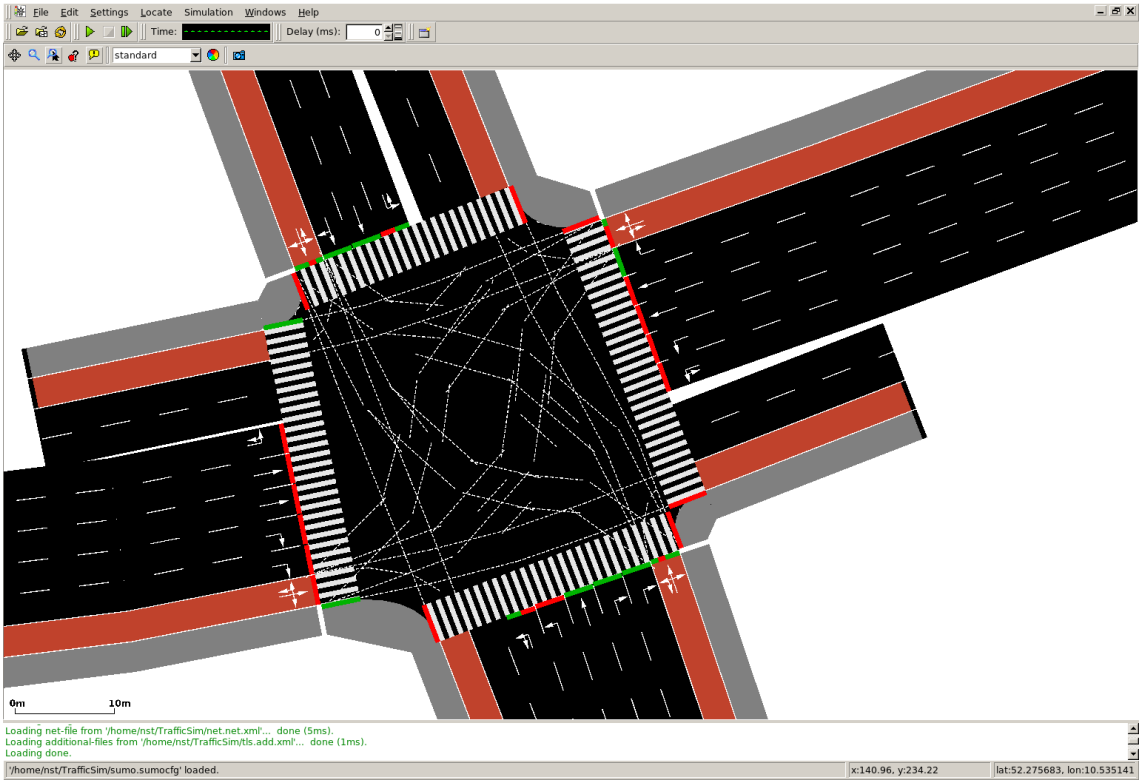

Abbildung 6.1: Mit TrafficSim erzeugte Kreuzung K47

Die Abbildung 6.1 zeigt die so entstandene K47. Zu sehen ist die erste Phase, in der von Norden und Süden das Geradeausfahren sowie das Rechtsabbiegen erlaubt ist. Aus Osten kommend ist das Rechtsabbiegen gestattet. Die Zuordnung der Arme erfolgt in diesem Fall im Uhrzeigersinn, beginnend mit der nördlichen Straße. Durch den Start von Sumo und der erkennbar korrekten Signalisierung kann davon ausgegangen werden, dass ein lauffähiges Programm erstellt wurde. Die weiteren Phasen werden hier nicht mehr dargestellt, wurden jedoch auch überprüft. Ein Blick in die erzeugte Datei zeigt die Reihenfolge der beiden Programme, wobei auch die fünf Durchläufe der kurzen Variante berücksichtigt wurden. In der Ausgabedatei SUMOs, welche aufgrund ihrer Größe hier nur in Auszügen dargestellt wird, ist erkennbar, dass zu den festgelegten Zeitpunkten auch wirklich der Wechsel erfolgte. Ebenfalls ist in der erzeugten Ausgangsdatei zu erkennen, dass die Übergänge der Phasen nach den Vorschriften erstellt wurden. Die erzeugte Datei enthält somit keine erkennbaren Fehler und auch die Zuordnung auf die Signale in SUMO erfolgt korrekt. Auch bei denen im Anhang gezeigten Beispielen ist kein Fehler erkennbar. Es wird von einer korrekten Erzeugung der LSA-Steuerung ausgegangen.

6.2 Test der losen Koppelung

Durch die lose Koppelung sollen Kreuzungen mit der gleichen Anzahl an Armen durch dieselben Eingaben erzeugt werden können. So muss es möglich sein, aus ei-

ner korrekten TrafficSim Beschreibung für die K10 auch die K47 zu erstellen, da diese beide vier Arme besitzen. Bei Anwendung auf die K51, einer fünfarmigen Kreuzung, muss jedoch ein Fehler auftreten. Der Nachweis dieser Eigenschaft, für die nicht erweiterte Version, erfolgte durch Sven Beeckmann [Bee15]. Getestet wird, ob nach der Erweiterung um eine LSA-Steuerung, die lose Koppelung weiterhin besteht. Dafür wurde eine Beschreibung eines LSA-Programmes erstellt. Dieses Programm besteht aus vier Phasen, wobei die Arme eins und drei sowie zwei und vier immer dasselbe Hauptsignal haben. Bei einem Phasenwechsel erfolgt auch ein Wechsel des Signals von Grün auf Rot und von Rot auf Grün. Nur in der ersten Phase werden die Eigenschaften *leftTurn* und *rightTurn* benutzt um die eigene Signalisierung der Abbieger zu testen. Dies ist auch der Grund dafür, dass vier Phasen gewählt wurden, da ansonsten nicht alle Fahrzeuge losfahren könnten. Die dadurch erzeugte Simulation wurde für die K47 bereits in Abbildung 6.1 auf der vorherigen Seite gezeigt. Die erstellte Kreuzung K10 ist in Abbildung A.1 auf Seite 51 zu erkennen.

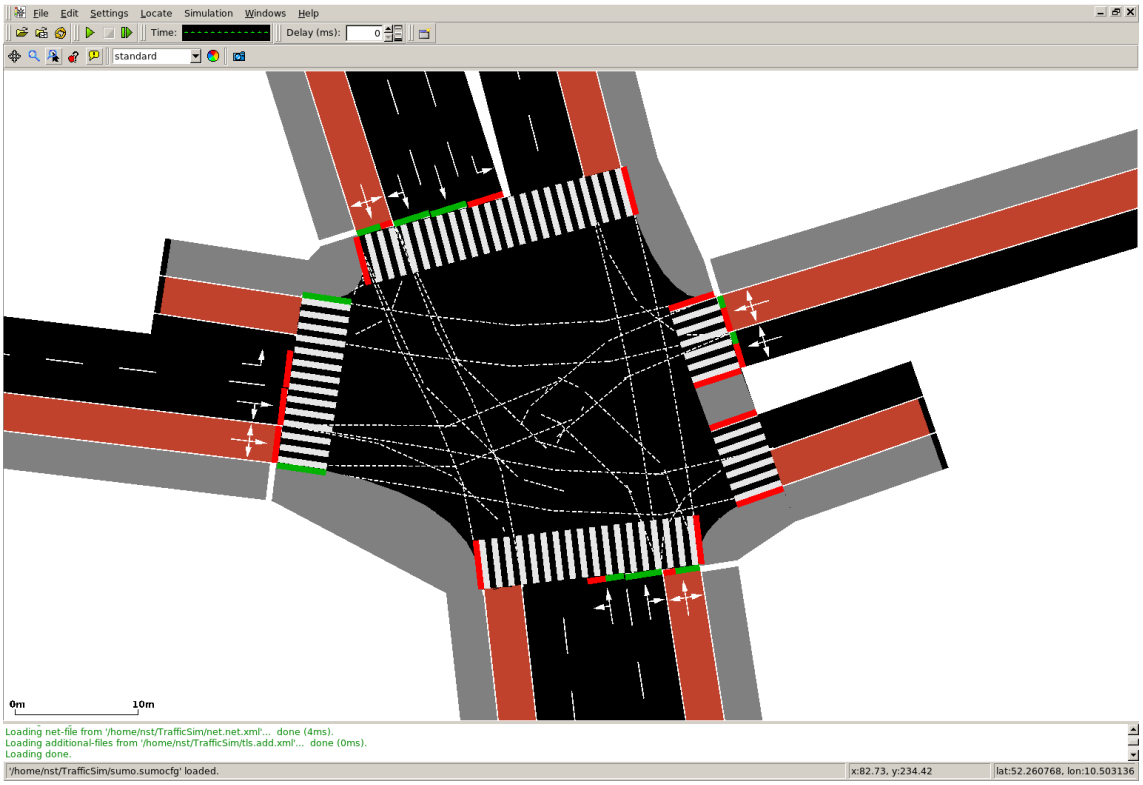

Abbildung 6.2: Mit TrafficSim erzeugte Kreuzung K10

Somit konnte dieselbe Beschreibung auf zwei unterschiedliche Intersection-Nachrichten angewandt werden. Dadurch ist die Teileigenschaft nachgewiesen, dass bei gleicher Anzahl an Armen dieselbe TrafficSim-Datei verwendet werden kann.
Als zweite Eigenschaft muss gezeigt werden, dass bei abweichender Armanzahl diese Beschreibung nicht anwendbar ist. Dafür wurde die K51 zum Test herangezogen. Dafür wurde die Eingabe der Kreuzungsarme um einen Arm ergänzt, da ansonsten die Konvertierung schon an dieser Stelle abbricht. An der Beschreibung der LSA-Steuerung wurden keine Änderungen vorgenommen, sodass dort nur vier Arme beschrieben werden. Bei der Konvertierung der K51 mit diesen Angaben tritt

nun ein Fehler auf. Die K51 kann somit nicht mit demselben LSA-Programm erzeugt werden wie die K47. Wird die getroffene Ergänzung weggelassen, scheitert die Erstellung der Kreuzung, wie durch Sven Beeckmann bereits gezeigt [Bee15]. Bei Verwendung der kurzen Variante ohne Angabe von Phasen, kann dieselbe Eingabe für alle Kreuzungen genutzt werden. Dieses ist möglich, da bei dieser Variante keine Arme referenziert werden müssen. Außerdem wird ein von SUMO für Kreuzungen erstelltes Programm nur abgewandelt, weswegen immer eine lauffähige LSA-Steuerung entsteht. Beim Testen mit allen drei Kreuzungen entstand jedes Mal eine lauffähige Simulation. Somit ist in den überprüften Fällen die lose Koppelung weiterhin gegeben. Auch wird vermutet, dass diese bei weiteren Kreuzungen ebenfalls bestehen bleibt. Dieses konnte jedoch nicht gezeigt werden, da nur die drei beschrieben TOPO-Nachrichten getestet wurden.

6.3 Erkennung und Behandlung von Eingabefehlern

In diesem Abschnitt wird die Erkennung von Fehlern bei der Eingabe einer Steuerung für die LSA behandelt. Diese erfolgt beim Erstellen der Simulation automatisch. Es erfolgt hierbei keine Überprüfung, ob das zu erstellende Programm in der Realität existieren dürfte. Daher erfolgt kein Abgleich mit den bei einer LSA-Steuerung geltenden Regeln. Lediglich die korrekte Schreibweise und die Sinnhaftigkeit der Zahlenwerte bei der Eingabe werden getestet. Dafür wird zuerst das Element *TrafficLight* untersucht und anschließend die dort angegebenen Phasen.
Das wichtigste Attribut ist der Name des *TrafficLights*. Hier werden Fehler, wie die doppelten Verwendung und das Fehlen eines Namens, erkannt. Das ist notwendig, da SUMO einen eigenen Namen für jedes Programm fordert. Ist diese Bedingung nicht erfüllt oder fehlt der Name komplett kann die Simulation nicht starten [DLRb]. Daher erfolgt auf die Erkennung des Fehlers der Abbruch der Simulationserstellung. Auch der angegeben Typ wird auf die Verwendung der beiden möglichen Eingaben überprüft. Dabei sind nur *static* und *actuated* erlaubt. Wird ein anderer Typ angegeben oder es tritt ein Rechtschreibfehler auf, so bricht das Programm mit einer Fehlermeldung ab. Möglich wäre aber auch, stattdessen einen der beiden Typen als Standard festzulegen, welcher bei einem Fehler in der Eingabe automatisch verwendet wird. In dieser Umsetzung wurde sich dagegen entschieden, da der ausgewählte Typ Einfluss auf den Ablauf der Simulation hat. Getestet wurden diese beiden Fehler durch absichtliches Herbeiführen. Es wurde mehrfach mit verschiedenen fehlerhaften Eingaben überprüft. Bei all diesen Versuchen wurde der Fehler erkannt und die korrekte Meldung angezeigt.
Erfolgt die Angabe der kurzen Variante, wird als letztes der Zahlenwert bei der Dauer überprüft. Dieser darf nicht kleiner als eins sein. Jede Phase muss mindestens eine Dauer von eins haben um von SUMO als korrekt anerkannt zu werden. Bei Phasen mit einer Dauer von Null oder kleiner kann die Simulation nicht starten [DLRb]. Daher erfolgt, wenn dieser Fehler auftritt, ein Abbruch der Erstellung der Simulation. Zusätzlich wird eine Meldung an den Nutzer ausgegeben. Ebenso wie beim Namen und Typen erfolgte der Test durch mehrfaches absichtliches Herbeiführen von Fehlern, mittels falscher Eingaben. Der Fehler wurde jedes Mal erkannt und angezeigt. Auf das Setzen eines Standardwertes wurde verzichtet, da die Eingabe der

Dauer großen Einfluss auf das LSA-Programm hat und bewusst durch den Nutzer durchgeführt werden soll.
Bei Verwendung der ausführlichen Variante wird die Überprüfung der Dauer jeder einzelnen Phase nach demselben System durchgeführt. Hierbei wurden alle getesteten fehlerhaften Eingaben erkannt. Bei Angabe von minDur und maxDur muss der Wert für die maximale Dauer größer sein als der für die minimale Dauer.
Ein wichtiger Aspekt der ausführlichen Variante ist das Angeben der Zustände für die jeweiligen Arme. Daher muss die Anzahl der Arme übereinstimmen und das Referenzieren korrekt erfolgen. Wird auf einen nicht bereits definierten Arm verwiesen, kann der Zustand nicht zugeordnet werden. Das ist ebenfalls der Fall, wenn zwei Arme in einer Phase sich auf denselben Arm beziehen. Ebenso wie bei einer zu geringen Anzahl an Armen kann somit nicht jeder Arm der Kreuzung einen Zustand erhalten und es kann kein LSA-Programm erstellt werden. Auch bei der Angabe von zu vielen Armen in einer Phase wird die Erstellung der Simulation unterbrochen. In den durchgeführten Tests wurden zum einen die Namen, über die das Referenzieren erfolgt, verändert und zum anderen die Anzahl der Arme reduziert bzw. erhöht. Die erwarteten Fehlermeldungen sind in allen Testfällen eingeblendet wurden.
Als Letztes werden die eingegebenen Zustände selbst auf ihre Korrektheit untersucht. Erlaubt sind die Zustände *green* und *red*. Die Eingabe von Übergangsphasen würde als Fehler erkannt werden, da diese automatisch erzeugt werden sollen. Da durch *state*, *leftTurn* und *rightTurn* dreimal die Angabe eines Zustands möglich ist, wird diese für alle drei überprüft. Da die Auswahl des Zustandes die wichtigste Funktion bei der Erstellung einer LSA-Steuerung ist, wurde auf eine automatische Korrektur durch das Programm verzichtet. Somit erfolgt ein Abbruch der Simulationserstellung, falls ein solches Problem auftritt. Die Tests erfolgten durch Herbeiführen von Rechtschreibfehlern und der Verwendung nicht erlaubter Zustände. Diese verliefen erfolgreich. Die Fehler wurden erkannt und angezeigt.
Abschließend kann gesagt werden, dass die Fehlererkennung in den Tests wie geplant funktionierte. Grundlegende Probleme wurden erkannt und angezeigt. Bei korrekten Eingaben trat kein Hinweis auf einen Fehler auf.

6.4 Nachbildung einer LSA-Steuerung

Als abschließender Test soll ein reales LSA-Programm mit TrafficSim erstellt und simuliert werden. Für die drei verwendeten Kreuzungen liegen keine originalen Schaltpläne vor. Daher wurde der Schaltplan aus den Richtlinien für Lichtsignalanlagen Ausgabe 2010 von Seite 29 verwendet und für den Test vereinfacht. Dieser Plan ist in der Abbildung 6.3 auf der nächsten Seite dargestellt. Für die Umsetzung wurde er in vier Phasen aufgeteilt. Die erste Phase beginnt im Plan bei Sekunde 28. Die beiden Signalgruppen K1 und K2 haben hier Grün als Signal, während dem Rest Rot signalisiert wird. Diese Phase dauert zehn Sekunden, bis die K2 auf Rot wechselt. Vereinfacht wird angenommen, dass die Gruppen K4 und K5 gleichzeitig ihr Signal auf Grün und später zurück auf Rot ändern. Nach einer Dauer von 20 Sekunden schaltet die K1 auf Rot um. Die übrigen Signalgruppen behalten ihr Signal bei. Bis zur nächsten Phase vergehen zwölf Sekunden. In dieser Phase wechselt das Signal der K4 und K5 von Grün auf Rot. Dafür erhalten die K3 und K6 Grün. Diese abschließende Phase ist 16 Sekunden lang. Insgesamt geht der Plan von einer

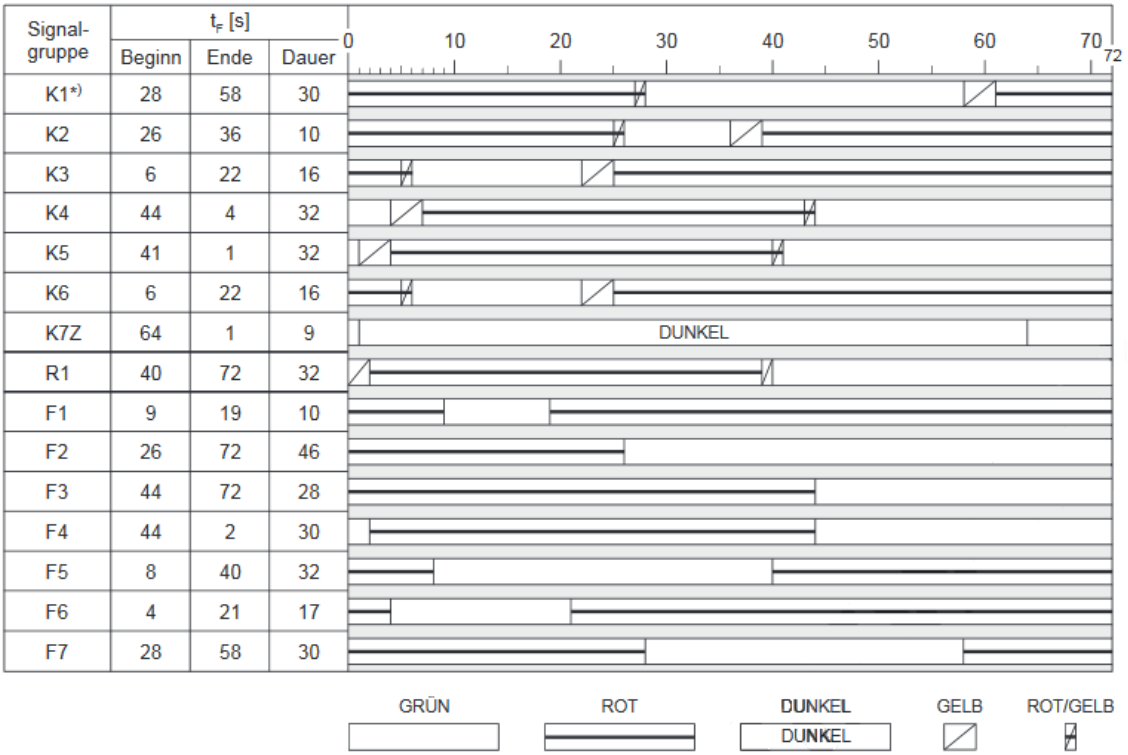

Abbildung 6.3: Phasenplan aus den Richtlinien für Lichtsignalanlagen

Programmlänge von 72 Sekunden aus. Beim erzeugten Programm wird eine Länge von 74 Sekunden erwartet, da bei Phase zwei eine Vereinfachung der Dauer erfolgte. Diese ist dadurch etwas länger als im originalen Plan. [fSuVAV10]
Für die Simulation wird die Kreuzung mit der Bezeichnung K10 verwendet. Diese weißt die wenigsten Abweichungen zu der in den Richtlinien für Lichtsignalanlagen auf Seite 12 dargestellten Kreuzung auf [fSuVAV10]. Beide sind vierarmig und erlauben das Fahren in alle Richtungen am Ende jedes Armes. Des Weiteren stimmte die Anzahl der Spuren auf den Armen besser überein als bei der K47. Die K10 ist jedoch keine exakte Abbildung der Kreuzung aus den RiLSA. Das Programm kann jedoch trotzdem nachgebaut werden. Die entstehende Simulation würde aber, aufgrund der Unterschiede in den Abbiegespuren, keine aussagekräftigen Daten liefern. In diesem Test wird daher auch nicht die Auswirkung der LSA auf die Kreuzung, sondern lediglich die korrekte Umsetzung des Steuerprogramms untersucht.
Im ersten Schritt wurde der beschriebene Plan in TrafficSim umgesetzt. Dafür wurde ein TrafficLight definiert welches den Namen „RiLSA2010“ erhielt. Als Typ ist *static* angegeben, da für die Auswertung keine Abweichungen vom Plan auftreten sollen. Für die Phasen wurden vier Phasenelemente erzeugt. In diesen wurden die Dauer und die einzelnen Zustände festgelegt. Die Dauer entspricht den angegebenen Werten. Die Zuordnung der Signalgruppen auf die Arme erfolgte wie in der Abbildung des RiLSA vorgesehen. Dabei entspricht der erste Arm dem Norden, der zweite dem Osten, der dritte dem Süden und der vierte dem Westen. In Phase eins erhält somit Arm zwei, und dadurch die Signalgruppen K1 und K2, Grün. In Phase zwei bleibt der Grundzustand dieses Armes bei Grün. Der Signalwechsel der Gruppe K2 wurde durch das Setzen des *leftTurn* auf *red* erzeugt. In den weiteren beiden Pha-

sen wurden anschließend nur die Grundzustände geändert. Mit diesem definierten LSA-Programm wurde die Simulation erstellt und durchgeführt. Für die Auswertung wurde die erzeugte XML-Datei mit dem Programm untersucht. Des Weiteren wurde eine Ausgabedatei nach abgeschlossener Simulation erzeugt. In dieser sind die Startzeitpunkte der einzelnen Phasen über die gesamte Simulationszeit angegeben [DLRb]. Mit dem Start der Simulation ist sichergestellt, dass ein lauffähiges LSA-Programm erstellt wurde. Im Weiteren werden daher nur die Korrektheit der einzelnen Phasen und der Ablauf dieser betrachtet.
Bei der Untersuchung der XML-Datei wurde auf die Reihenfolge der Signale und die Dauer dieser geachtet. Dadurch wurden die automatisch erzeugten Übergangsphasen überprüft. Wichtig ist dabei, dass auf Grün eine dreisekündige Gelbphase folgt, bevor sich eine einsekündige Rotphase anschließt, in der beim Wechsel von Rot zu Grün das Gelb-Rote Signal zu sehen ist. Beim Übergang von der ersten in die zweite Phase muss beachtet werden, dass nicht alle Signale diesem Übergang folgen. Erfolgt kein Signalwechsel, muss das rote oder grüne Signal auch in den Übergängen erhalten bleiben. In der XML-Datei sind die erwarteten Werte für die Dauer der Phasen zu erkennen. Auch beim Betrachten der Zustandswechsel sind keine Fehler zu erkennen. Es wird von einem korrekt erzeugten Programm ausgegangen. Ob die Zuordnung auf die Arme richtig umgesetzt wurde, ist aus dieser Datei nicht herauszulesen. Dieser Test erfolgte durch optische Überprüfung der einzelnen Signale auf den Armen der laufenden Simulation. Diese wurde dazu zu den jeweiligen Zeitpunkten angehalten. Es zeigte sich, dass die Zuordnung wie erwartet funktionierte. Durch die von SUMO erzeugte Ausgabedatei konnten die einzelnen Phasen zu jedem Zeitpunkt nachverfolgt werden [DLRb]. Dabei zeigte sich, dass die angegebene Dauer der Phasen umgesetzt wurde und dem Plan entsprach.
Mit diesem Test sollte gezeigt werden, dass sich reale Programme in TrafficSim nachbilden lassen. Dieses gelang für das LSA-Programm aus den Richtlinien für Lichtsignalanlagen und der Kreuzung K10. Es wird davon ausgegangen, dass dieses auch mit anderen Programmen möglich ist.

Alle Tests wurden an den Kreuzungen K10, K47 und K51 vorgenommen. Untersucht wurden beide implementierten Eingabevarianten. Bei der Verifikation der Phasenerzeugung wurde überprüft, ob die erzeugten LSA-Programme in jeder Phase die Eingaben umsetzen. Diese Tests wurden erfolgreich abgeschlossen. Die Erkennung von Fehlern wurde mithilfe von fehlerhaften Eingaben durchgeführt. Hierbei wurden alle Fehler erkannt und nur korrekte Programme erzeugt. Der letzte Test war die Nachbildung eines realen LSA-Programms. Dieses wurde für die K10 umgesetzt. Bei der Umsetzung traten keine Probleme auf, sodass sich das Programm nachbilden ließ. Die Experimente wurden erfolgreich beendet. Es ist davon auszugehen, dass sich LSA-Programme nach eigenen Vorstellungen korrekt erzeugen lassen.

7. Zusammenfassung und Ausblick

7.1 Zusammenfassung

Die Deklarationssprache TrafficSim dient der Konfiguration von Verkehrssimulationen. Sie ermöglicht die Beschreibung einer Kreuzung durch den Nutzer. Auch Simulationsschritte und die Ankunft von Fahrzeugen können definiert werden. Jedoch war es bisher nicht möglich Steuerprogramme für Lichtsignalanlagen zu erstellen.
Für die Erweiterung der Sprache wurden zwei Konzepte erarbeitet, welche umgesetzt wurden.Die Eingabe wurde dabei um zwei Varianten erweitert. Eine kurze Variante ermöglicht die schnelle Veränderung der Dauer von Hauptphasen. Jedoch ist kein Einfluss auf die Signalisierung vorhanden. Die Eingabe erfolgt über das neue Element *TrafficLight*. Die zweite Variante stellt eine ausführlichere Eingabe dar. Mit ihr ist es möglich eigene Phase zu beschreiben. Es ist möglich Links- und Rechtsabbiegern eigene Signale zu übermitteln. Die Dauer der Phasen wird nicht wie in der kurzen Variante im *TrafficLight* festgelegt, sondern innerhalb jeder Phase. Durch die ausführliche Variante können eigene Programme erzeugt oder reale LSA-Programme nachgebildet werden.
Für die Erzeugung der LSA-Programme wurden bestehende Klassen erweitert und mit dem *TrafficLight*, den *Phasen* sowie dem XML-Reader neue Klassen hinzugefügt. Für das Parsen der Eingabe wurde der Parser erweitert.
Getestet wurde die Erweiterung durch die Generierung dreier Kreuzungen mit verschiedenen definierten LSA-Programmen. Dabei wurden beide Varianten einzeln und gemischt getestet. Durch Start der Simulation konnten schwere Fehler ausgeschlossen werden, da SUMO ansonsten die Simulation abbricht. Die Korrektheit der erzeugten Phasen wurde durch optischen Vergleich vorgenommen. Es wurden die jeweiligen Signale der Arme mit den erwarteten Signalen verglichen. Durch eine Ausgabedatei wurde überprüft, ob die Programme zum korrekten Zeitpunkt wechseln. Bei diesen Tests wurden keine Fehler gefunden, sodass von einer korrekten Phasenerzeugung ausgegangen wird.
Die lose Koppelung sollte bei der Erweiterung erhalten bleiben. Ein definiertes LSA-Programm muss daher auf alle Kreuzungen mit gleicher Armanzahl anwendbar sein. Überprüft wurde dieses anhand von zwei Kreuzungen mit vier Armen. Bei beiden wurde mit gleicher Eingabe das entsprechende LSA-Programm erzeugt. Die lose Koppelung ist daher weiterhin gegeben.
Beim abschließenden Test wurde ein reales LSA-Programm nachgebildet. Da für die Kreuzungen kein solches Programm vorlag, wurde ein LSA-Programm einer anderen Kreuzung angepasst. Die Beschreibung erfolgte durch die ausführliche Variante. Das Programm konnte vollständig beschrieben werden. Bei der Erzeugung der Phasen trat kein Fehler auf. Somit konnte ein reales LSA-Programm nachgebildet werden.

7.2 Erweiterungsmöglichkeiten für TrafficSim

In diesem Abschnitt sollen andere Möglichkeiten der Erweiterungen für TrafficSim genannt und erläutert werden. Da bisher nur eine prototypische Umsetzung für SUMO existiert, wird im folgenden Abschnitt speziell auf Erweiterungen für diese Simulationssoftware eingegangen. Die dort genannten Ideen sind jedoch gegebenenfalls auch auf andere Simulationen übertragbar, insofern diese das Feature allgemein unterstützen. Eine mögliche Erweiterung von TrafficSim wäre somit die Umsetzung für eine weitere Simulation z.B. Anylogic oder VISSIM. Daraus würden sich neue Voraussetzungen und Einschränkungen ergeben. Das aktuelle Programm müsste dementsprechend angepasst werden. Neue Eingaben und Einstellungen könnten benötigt werden. Auch auf das Ausgabeformat müsste geachtet werden. Es könnten aber auch Möglichkeiten entstehen die durch SUMO nicht umsetzbar sind.

7.2.1 Erweiterungen für die Simulationssoftware SUMO

Bislang kann durch TrafficSim nur eine einzelne Kreuzung erstellt werden. Daher könnte die Sprache für den Umgang mit mehreren Kreuzungen erweitert werden. Durch Verbinden dieser Kreuzung würde ein Verkehrsnetz entstehen. Eine Idee zur Umsetzung ist das Erstellen einzelner Kreuzungen unabhängig voneinander. Dafür bräuchte jede Kreuzung ihre eigene TOPO-Nachricht und eine eigene Konfigurationsdatei. In der mit TrafficSim geschriebenen Datei würden dann alle Kreuzungen einzeln definiert. Durch das Element *Intersection* wäre dies möglich, es müsste nur eine Liste mit allen Kreuzungen erstellt werden. Da weiterhin nur eine Simulation erstellt wird gelten die Simulationsschritte für alle Kreuzungen. Hierbei müsste nur die Möglichkeit geschaffen werden, die Angaben zur Ankunft der Fahrzeuge, für alle in das System führenden Straßen, zu regeln. Das Problem dabei ist, dass nicht jede Kreuzung hinführende Straßen besitzt, sondern ein innerer Teil des Netzes ist. Für diese Kreuzung dürften keine Angaben akzeptiert werden. Um aus den so entstandenen einzelnen Kreuzungen ein Netz zu gestalten, könnten diese miteinander verbunden werden. Dabei würden die ausgehenden Kanten einer Kreuzung zu den eingehenden Kanten einer anderen. Ausnahmen sind die aus dem System rein- und rausführenden Straßen, welche ermittelt werden müssten. Auch stellt sich die Frage, welche Verbindungen korrekt sind und erstellt werden sollen. Dafür könnte eine neue Eingabe geschaffen werden, in der die miteinander verknüpften Kreuzungen angegeben werden. Aus den Koordinaten der Kreuzungen könnte die Richtung der Verbindung bestimmt werden. Für die beiden Kreuzungen müssten dann nur noch die entsprechenden Arme gefunden und verknüpft werden. In einem solchen Verkehrsnetz wäre es auch möglich die Lichtsignalanlagen aufeinander abzustimmen. Dies könnte durch die Eingabe vom Nutzer selbst geschehen oder über einen Parameter, bei dem für die einzelnen Kreuzungen die Programme leicht zeitlich verschoben und angepasst werden. Die Eingabe für die LSA-Steuerung insgesamt könnte aus den einzelnen Kreuzungen ausgelagert werden. Dadurch könnten Programme angegeben werden die für alle Anlagen gelten. Alternativ können auch die so gesteuerten Anlagen explizit angegeben werden. Ebenfalls können weitere Situationen wie Straßensperrungen integriert werden. Durch die Einführung einiger Parameter könnten vordefinierte Szenarien aufgerufen werden. Ein Beispiel hierfür sind Straßensperrungen. Durch die Angabe der betroffenen Straße könnte auf dieser eine Spur

gesperrt oder alternative weniger erzeugt werden. Mithilfe von TraCI könnte auch das Verhalten bei solchen Situationen angepasst werden. Es wäre möglich die Fahrer auf Alternativrouten umzulenken. Durch die Eingabe über Parameter soll die Menge der benötigten Eingaben gering gehalten werden. Deswegen würden durch die Auswahl einiger Szenarien nicht alle denkbaren Fälle abbildbar sein. Es können jedoch einige Hauptszenarien abgebildet sein. Die Liste der so auswählbaren Fälle wäre auch weiterhin erweiterbar.
Neben der Erweiterung auf ein Verkehrsnetz und den damit einhergehenden Möglichkeiten gibt es auch Erweiterungen für eine einzelne Kreuzung. Dabei könnten öffentliche Verkehrsmittel stärker integriert werden. So sind aktuell keine Straßenbahnen in der Simulation verfügbar. Diese könnten z.B. mit eigener Spur und eigener Signalisierung, ähnlich den Fahrradspuren, erstellt werden. In einem Verkehrsnetz könnte für einzelne Straßen kreuzender Schienenverkehr erstellt werden. Für die bereits enthaltenen Busse könnte die Möglichkeit, Bushaltestellen auf den einzelnen Armen festzulegen, integriert werden.

7.2.2 Erweiterungsmöglichkeiten für die Lichtsignalanlagensteuerung

Auch für die in dieser Arbeit eingebunden Lichtsignalanlagen bestehen zusätzliche Möglichkeiten die Steuerung dieser bei Bedarf zu erweitern. So ist eine noch ausführlichere Variante denkbar, in der auch die einzelnen Übergangsphasen eingestellt werden können. Bisher werden diese vom Programm automatisch erzeugt. Durch das Einstellen dieser könnten auch fehlerhafte LSA besser simuliert werden. Da somit aber auch eine größere Komplexität entsteht, wurde diese Idee im Zuge dieser Arbeit nicht umgesetzt. Ebenfalls kann die LSA-Steuerung um die Beeinflussung der Fußgänger-LSA erweitert werden. Die Signalisierung der Fußgänger richtet sich bisher nach den eingegebenen Phasen für die Fahrzeuge. Dabei werden die geltenden Verkehrsregeln umgesetzt. Vorstellbar wäre, durch einen einzelnen Wert allen Fußgänger Grün zu geben. Über eine Zeitangabe kann die Dauer dieser Phase beeinflusst werden. Der Vorteil dieser Variante wäre, dass nur noch zwei Eingaben für das Erstellen einer solchen Phase notwendig wären. Da aber dasselbe Ergebnis auch durch das Signalisieren von Rot für alle Fahrzeuge möglich ist, wurde diese Erweiterung bisher nicht umgesetzt. In der bisherigen Umsetzung kann nur durch die Reihenfolge der Definition von LSA-Programmen Einfluss auf die Reihenfolge der Ausführung dieser genommen werden. Durch eine Liste, in der die Namen der einzelnen Programme mit einem | getrennt sind, könnte die Reihenfolge nachträglich noch leicht geändert werden. Aber auch die Angabe des folgenden Programms im Programm selber ist möglich, jedoch kann dabei keine wechselnde Reihenfolge abgebildet werden. Dies wäre mit der ersten Variante möglich. Ebenfalls könnte der Wechsel eines Programms nicht nach Durchläufen, sondern nach einer festgelegten Zeit erfolgen. Dabei ist es jedoch möglich, dass Phasen plötzlich springen und kein flüssiger Übergang gewährleistet ist.
Eine größere Erweiterung wäre das Einbinden von TraCI bei SUMO, bzw. vergleichbaren Funktionalitäten bei anderen Simulationen. Dadurch würde es ermöglicht werden, bestimmten Fahrzeugtypen Vorfahrt zu gewähren. Als Eingabe könnten der Fahrzeugtyp und die Arme, auf denen die Vorfahrt gelten soll, dienen. Daraus müsste zusammen mit den angegebenen LSA-Programmen ein neues Programm erstellt

werden, welches die LSA-Steuerung erzeugt und die benötigten Sensoren abfragt. Diese Sensoren müssten zusätzlich in einem Additional-File beschrieben werden. Das erzeugte Programm würde in Python geschrieben werden, da SUMO aktuell C++ nicht vollständig unterstützt. Das Einbinden dieser Funktionalität in TrafficSim erfordert jedoch noch genauere Einarbeitung und wurde deswegen in dieser Arbeit nicht umgesetzt.

7.3 Bewertung

Innerhalb dieser Arbeit sollte die Deklarationssprache TrafficSim um eine Möglichkeit zur Konfiguration von Lichtsignalanlagen erweitert werden. Dafür mussten verschiedenen Konzepte zu Spracherweiterung erarbeitet werden. Diese sollten anschließend prototypisch für SUMO umgesetzt werden. Mit Abschluss der Arbeit ist es möglich, eigene LSA-Programme zu erzeugen. Real existierende LSA-Steuerungen können nachgebaut werden. Die Bedingungen der Deklarationssprache und der Lichtsignalanlagen müssen beachtet werden. Dem Nutzer ist durch einfache Eingaben die Beeinflussung des LSA-Programms gestattet.
Das Ziel, LSA-Programme definieren zu können, wurde erreicht. Es lassen sich verschiedene Programme erstellen. Dabei wird die vom Nutzer getätigte Eingabe vollständig umgesetzt. Bedingungen, wie die lose Koppelung, wurden beachtet. So ist es möglich, bei gleicher Anzahl an Armen dieselbe Beschreibung zu verwenden. Auch die Bedingungen der LSA sind umgesetzt wurden. So werden die Übergangsphasen automatisch erzeugt, um dieses zu gewährleisten. Durch ein Experiment konnte gezeigt werden, dass sich reale LSA-Steuerungen umsetzen lassen. Durch zwei umgesetzte Eingabevarianten kann der Grad der Beeinflussung durch den Nutzer ausgewählt werden. Es existiert eine kurze Variante, in der nur die Dauer der Phasen beeinflusst werden kann. In einer ausführlicheren Variante ist auch die Signalisierung veränderbar. Mit diesen zwei Varianten wurden den Nutzern einfache Eingabemethoden zur Verfügung gestellt.
Einschränkungen ergeben sich aus der Simulationssoftware SUMO. Es wird auf jeder Spur für jede Abbiegemöglichkeit ein eigenes Signal erstellt. Da diese unterschiedliche Signale erhalten können besteht die Möglichkeit, dass ein Fahrzeug, welches anhalten muss, weitere Fahrzeuge aufhält, obwohl diese fahren dürften. Der Sinn eines erstellten Programms wird nicht überprüft. So ist es möglich allen Spuren gleichzeitig Grün zu Signalisieren. Da somit eine Störung der LSA simuliert werden kann, wurde auf diese Überprüfung verzichtet. Eine weitere Einschränkung ist, dass die Umsetzung nur für vier- und fünfarmige Kreuzungen getestet wurde.
Zusammenfassend kann gesagt werden, dass das Ziel die Sprache TrafficSim um die Konfiguration von LSA zu erweitern, erreicht wurde. Es können verschiedene LSA-Programme definiert werden. Diese werden in der Simulation korrekt umgesetzt. Dafür geltende Bedingungen werden beachtet.

A. Anhang

Quelltext A.1: Durch TrafficSim erzeugtes LSA-Programm für die K10

```
<tlLogic type="static" offset="0"
programID="long" id="cluster_0_10_12_14_2_4_6_8">
<phase maxDur="12.5" minDur="7.5"
state="grrgrrggrgggrrrrrrrrggrgggrrrgr" duration="10"/>
<phase maxDur="5"    minDur="3"
state="grrgrrgyrgyyrrrrrrrryyryyyyrrrrr" duration="3"/>
<phase maxDur="4"    minDur="1"
state="guuguugrrgrrruuuuuuuurrurrrrrrrr" duration="1"/>
<phase maxDur="25"   minDur="15"
state="ggggggggrrgrrrgggggggrrgrrrrrrrr" duration="20"/>
<phase maxDur="5"    minDur="3"
state="gyygyygrrgrrryyyyyygrrgrrrrrrrr" duration="3"/>
<phase maxDur="4"    minDur="1"
state="grrgrrguuguuurrrrrrguuguuurrrrr" duration="1"/>
<phase maxDur="15"   minDur="9"
state="grrgrrgggggggrrrrrrgggggggrrrgr" duration="12"/>
<phase maxDur="5"    minDur="3"
state="grrgrrggygggyrrrrrryggygggrrrrr" duration="3"/>
<phase maxDur="4"    minDur="1"
state="grrgrrggrgggrrrrrrrrggrgggrrrrr" duration="1"/>
</tlLogic>
```

Quelltext A.2: Auszug aus der Ausgabedatei für LSA der Simulation für die K10

```
<tlsstate state="grrgrrggrgggrrrrrrrrggrgggrrrgr" phase="0"
programID="long" id="cluster_0_10_12_14_2_4_6_8" time="0.00"/>
<tlsstate state="grrgrrgyrgyyrrrrrrrryyryyyyrrrrr" phase="1"
programID="long" id="cluster_0_10_12_14_2_4_6_8" time="10.00"/>
<tlsstate state="guuguugrrgrrruuuuuuuurrurrrrrrrr" phase="2"
programID="long" id="cluster_0_10_12_14_2_4_6_8" time="13.00"/>
<tlsstate state="ggggggggrrgrrrgggggggrrgrrrrrrrr" phase="3"
programID="long" id="cluster_0_10_12_14_2_4_6_8" time="14.00"/>
<tlsstate state="gyygyygrrgrrryyyyyygrrgrrrrrrrr" phase="4"
programID="long" id="cluster_0_10_12_14_2_4_6_8" time="34.00"/>
<tlsstate state="grrgrrguuguuurrrrrrguuguuurrrrr" phase="5"
programID="long" id="cluster_0_10_12_14_2_4_6_8" time="37.00"/>
<tlsstate state="grrgrrggggggrrrrrrgggggggrrrgr" phase="6"
programID="long" id="cluster_0_10_12_14_2_4_6_8" time="38.00"/>
<tlsstate state="grrgrrggygggyrrrrrryggygggrrrrr" phase="7"
programID="long" id="cluster_0_10_12_14_2_4_6_8" time="50.00"/>
```

```
<tlsstate state="grrgrrggrgggrrrrrrrrggrgggrrrrr" phase="8"
programID="long" id="cluster_0_10_12_14_2_4_6_8" time="53.00"/>
<tlsstate state="rrrrrrgGgggggrrrrrrgGgggggGGrGr" phase="0"
programID="short" id="cluster_0_10_12_14_2_4_6_8" time="54.00"/>
<tlsstate state="rrrrrrgGgggggrrrrrrgGgggggrrrrr" phase="1"
programID="short" id="cluster_0_10_12_14_2_4_6_8" time="66.00"/>
<tlsstate state="rrrrrryygyyygrrrrrryygyyygrrrrr" phase="2"
programID="short" id="cluster_0_10_12_14_2_4_6_8" time="69.00"/>
<tlsstate state="rrrrrrrrGrrrGrrrrrrrrGrrrGrrrrr" phase="3"
programID="short" id="cluster_0_10_12_14_2_4_6_8" time="72.00"/>
<tlsstate state="rrrrrrrryrrryrrrrrrrryrrryrrrrr" phase="4"
programID="short" id="cluster_0_10_12_14_2_4_6_8" time="77.00"/>
```

Quelltext A.3: Durch TrafficSim erzeugtes LSA-Programm für die K47

```
<tlLogic type="static" offset="0"
programID="long" id="cluster_0_10_12_14_2_4_6_8">
<phase maxDur="12.5" minDur="7.5" duration="10"
state="grrrgrrrrrggrgggggrrgrrrrrrrrrrrgggrgggggrrgr"/>
<phase maxDur="5"    minDur="3"   duration="3"
state="grrrgrrrrrgyryggyrryrrrrrrrrrrryyyryyyyyrrrr"/>
<phase maxDur="4"    minDur="1"   duration="1"
state="guuuguuuuugrrrggrrrruuuuuuuuuuuurrrurrrrrrrr"/>
<phase maxDur="25"   minDur="15"  duration="20"
state="gggggggggggrrrggrrrrgggggggggggrrrgrrrrrrrr"/>
<phase maxDur="5"    minDur="3"   duration="3"
state="gyyygyyyyygrrrggrrrryyyyyyyyyygrrrgrrrrrrrr"/>
<phase maxDur="4"    minDur="1"   duration="1"
state="grrrgrrrrrguuugguuuurrrrrrrrrguuuguuuurrrr"/>
<phase maxDur="15"   minDur="9"   duration="12"
state="grrrgrrrrrgggggggggggrrrrrrrrrrgggggggggrrgr"/>
<phase maxDur="5"    minDur="3"   duration="3"
state="grrrgrrrrrggygggggyygrrrrrrrrrrygggygggggrrrr"/>
<phase maxDur="4"    minDur="1"   duration="1"
state="grrrgrrrrrggrgggggrrgrrrrrrrrrrgggrgggggrrrr"/>
</tlLogic>
```

Quelltext A.4: Auszug aus der Ausgabedatei für LSA der Simulation für die K47

```
<tlsstate state="grrrgrrrrrggrgggggrrgrrrrrrrrrrgggrgggggrrgr"
phase="0" programID="long"
id="cluster_0_10_12_14_2_4_6_8" time="0.00"/>
<tlsstate state="grrrgrrrrrgyryggyrryrrrrrrrrrrryyyryyyyyrrrr"
phase="1" programID="long"
id="cluster_0_10_12_14_2_4_6_8" time="10.00"/>
<tlsstate state="guuuguuuuugrrrggrrrruuuuuuuuuuuurrrurrrrrrrr"
phase="2" programID="long"
id="cluster_0_10_12_14_2_4_6_8" time="13.00"/>
<tlsstate state="gggggggggggrrrggrrrrgggggggggggrrrgrrrrrrrr"
```

```
phase="3" programID="long"
id="cluster_0_10_12_14_2_4_6_8" time="14.00"/>
<tlsstate state="gyyygyyyyygrrrggrrrryyyyyyyyygrrrgrrrrrrrr"
phase="4" programID="long"
id="cluster_0_10_12_14_2_4_6_8" time="34.00"/>
<tlsstate state="grrrgrrrrrguuugguuuurrrrrrrrrrguuuguuuurrrr"
phase="5" programID="long"
id="cluster_0_10_12_14_2_4_6_8" time="37.00"/>
<tlsstate state="grrrgrrrrrgggggggggrrrrrrrrrrgggggggggrrgr"
phase="6" programID="long"
id="cluster_0_10_12_14_2_4_6_8" time="38.00"/>
<tlsstate state="grrrgrrrrrggygggggyygrrrrrrrrrrygggygggggrrrr"
phase="7" programID="long"
id="cluster_0_10_12_14_2_4_6_8" time="50.00"/>
<tlsstate state="grrrgrrrrrggrggggrrgrrrrrrrrrrrgggrggggrrrr"
 phase="8" programID="long"
 id="cluster_0_10_12_14_2_4_6_8" time="53.00"/>
<tlsstate state="gGgggggggggrrrrrrrrrrgGgggggggggrrrrrrrrrrGrG"
phase="0" programID="short"
id="cluster_0_10_12_14_2_4_6_8" time="54.00"/>
<tlsstate state="gGgggggggggrrrrrrrrrrgGgggggggggrrrrrrrrrrrrr"
phase="1" programID="short"
id="cluster_0_10_12_14_2_4_6_8" time="66.00"/>
<tlsstate state="yyggyyyyggrrrrrrrrrryyggyyyyyggrrrrrrrrrrrrr"
phase="2" programID="short"
id="cluster_0_10_12_14_2_4_6_8" time="69.00"/>
<tlsstate state="rrGGrrrGGGrrrrrrrrrrrGGrrrrGGrrrrrrrrrrrrr"
phase="3" programID="short"
id="cluster_0_10_12_14_2_4_6_8" time="72.00"/>
<tlsstate state="rryyrrryyyyrrrrrrrrrrrryyrrrryyrrrrrrrrrrrrr"
phase="4" programID="short"
id="cluster_0_10_12_14_2_4_6_8" time="77.00"/>
```

Quelltext A.5: Durch TrafficSim erzeugtes LSA-Programm für die K51

```
<tlLogic type="static" offset="0"
programID="long" id="cluster_0_10_12_14_16_18_2_4_6_8">
<phase maxDur="12.5" minDur="7.5"  duration="10"
state="grrrgrrrrggrrrrrrrrrrrrrgggrgggrrrrgggr"/>
<phase maxDur="5"    minDur="3"    duration="3"
state="grrrgrrrrggrrrrrrrrrrrrryyyryyyyrrrrrrrr"/>
<phase maxDur="4"    minDur="1"    duration="1"
state="guuuguuuuggguuuuuuuuuuuuurrrurrrrrrrrrrr"/>
<phase maxDur="25"   minDur="15"   duration="20"
state="gggggggggggggggggggggggggrrrgrrrrrrrrrrr"/>
<phase maxDur="5"    minDur="3"    duration="3"
state="gyyygyyyyyggyyyyyyyyyyyygrrrgrrrrrrrrrrr"/>
<phase maxDur="4"    minDur="1"    duration="1"
```

```
state="grrrgrrrrggrrrrrrrrrrrrguuuguuurrrrrrrr"/>
<phase maxDur="15" minDur="9" duration="12"
state="grrrgrrrrggrrrrrrrrrrrrgggggggggrrrrgggr"/>
<phase maxDur="5" minDur="3" duration="3"
state="grrrgrrrrggrrrrrrrrrrrrygggygggrrrrrrrr"/>
<phase maxDur="4" minDur="1" duration="1"
state="grrrgrrrrggrrrrrrrrrrrrgggrgggrrrrrrrr"/>
</tlLogic>
```

Quelltext A.6: Auszug aus der Ausgabedatei für LSA der Simulation für die K51

```
<tlsstate state="grrrgrrrrggrrrrrrrrrrrrgggrgggrrrrgggr"
phase="0" programID="long"
id="cluster_0_10_12_14_16_18_2_4_6_8" time="0.00"/>
<tlsstate state="grrrgrrrrggrrrrrrrrrrrryyyryyyyrrrrrrr"
phase="1" programID="long"
id="cluster_0_10_12_14_16_18_2_4_6_8" time="10.00"/>
<tlsstate state="guuuguuuugguuuuuuuuuuuurrrurrrrrrrrrr"
phase="2" programID="long"
id="cluster_0_10_12_14_16_18_2_4_6_8" time="13.00"/>
<tlsstate state="gggggggggggggggggggggggrrrgrrrrrrrrrr"
phase="3" programID="long"
id="cluster_0_10_12_14_16_18_2_4_6_8" time="14.00"/>
<tlsstate state="gyyygyyyyggyyyyyyyyyyygrrrgrrrrrrrrrr"
phase="4" programID="long"
id="cluster_0_10_12_14_16_18_2_4_6_8" time="34.00"/>
<tlsstate state="grrrgrrrrggrrrrrrrrrrrguuuguuurrrrrrrr"
phase="5" programID="long"
id="cluster_0_10_12_14_16_18_2_4_6_8" time="37.00"/>
<tlsstate state="grrrgrrrrggrrrrrrrrrrrrgggggggggrrrrgggr"
phase="6" programID="long"
id="cluster_0_10_12_14_16_18_2_4_6_8" time="38.00"/>
<tlsstate state="grrrgrrrrggrrrrrrrrrrrrygggygggrrrrrrrr"
phase="7" programID="long"
id="cluster_0_10_12_14_16_18_2_4_6_8" time="50.00"/>
<tlsstate state="grrrgrrrrggrrrrrrrrrrrrgggrgggrrrrrrrr"
phase="8" programID="long"
id="cluster_0_10_12_14_16_18_2_4_6_8" time="53.00"/>
<tlsstate state="gGgggggggrrrrrrrrrGGgGGrrrrrrrrrrGGrrrG"
phase="0" programID="short"
id="cluster_0_10_12_14_16_18_2_4_6_8" time="54.00"/>
<tlsstate state="gGgggggggrrrrrrrrrGGgGGrrrrrrrrrrrrrrrr"
phase="1" programID="short"
id="cluster_0_10_12_14_16_18_2_4_6_8" time="66.00"/>
<tlsstate state="yyggyyyygrrrrrrrrryyyyyrrrrrrrrrrrrrrr"
phase="2" programID="short"
id="cluster_0_10_12_14_16_18_2_4_6_8" time="69.00"/>
<tlsstate state="rrGGrrrrGrrrrrrrrrrrrrrrrrrrrrrrrrrr"
```

```
phase="3" programID="short"
id="cluster_0_10_12_14_16_18_2_4_6_8" time="72.00"/>
<tlsstate state="rryyrrrryrrrrrrrrrrrrrrrrrrrrrrrrrr"
phase="4" programID="short"
id="cluster_0_10_12_14_16_18_2_4_6_8" time="77.00"/>
```

Quelltext A.7: Durch TrafficSim erzeugtes LSA-Programm für die Nachbildung eines realen LSA-Programms

```
<tlLogic type="static" offset="0"
programID="RiLSA_2010" id="cluster_0_10_12_14_2_4_6_8">
<phase maxDur="12.5" minDur="7.5"
state="gggggggrrrrrrrrrrrrrrrrrrrrggg" duration="10"/>
<phase maxDur="5"    minDur="3"
state="ggyggyrrrrrrrrrrrrrrrrrrrrrrrr" duration="3"/>
<phase maxDur="4"    minDur="1"
state="ggrggrrrrrrrruuuuuurrrrrrrrrrr" duration="1"/>
<phase maxDur="25"   minDur="15"
state="ggrggrrrrrrrrgggggggrrrrrrrrrgrg" duration="20"/>
<phase maxDur="5"    minDur="3"
state="yyryyyrrrrrrrrgggggggrrrrrrrrrrr" duration="3"/>
<phase maxDur="4"    minDur="1"
state="rrrrrrrrrrrrrgggggggrrrrrrrrrrr" duration="1"/>
<phase maxDur="15"   minDur="9"
state="rrrrrrrrrrrrrgggggggrrrrrrrgggrg" duration="12"/>
<phase maxDur="5"    minDur="3"
state="rrrrrrrrrrrrryyyyyyyrrrrrrrrrrr" duration="3"/>
<phase maxDur="4"    minDur="1"
state="rrrrrruuuuuuuurrrrrruuuuuuurrrrr" duration="1"/>
<phase maxDur="20"   minDur="12"
state="rrrrrrgggggggrrrrrrgggggggggrgr" duration="16"/>
<phase maxDur="5"    minDur="3"
state="rrrrrryyyyyyyrrrrrryyyyyyyrrrrr" duration="3"/>
<phase maxDur="4"    minDur="1"
state="uuuuuurrrrrrrrrrrrrrrrrrrrrrrr" duration="1"/>
</tlLogic>
```

Quelltext A.8: Auszug aus der Ausgabedatei für LSA der Simulation für die Nachbildung eines realen LSA-Programms

```
<tlsstate state="gggggggrrrrrrrrrrrrrrrrrrrrggg"
phase="0" programID="RiLSA_2010"
id="cluster_0_10_12_14_2_4_6_8" time="0.00"/>
<tlsstate state="ggyggyrrrrrrrrrrrrrrrrrrrrrrrr"
phase="1" programID="RiLSA_2010"
id="cluster_0_10_12_14_2_4_6_8" time="10.00"/>
<tlsstate state="ggrggrrrrrrrruuuuuurrrrrrrrrrr"
phase="2" programID="RiLSA_2010"
id="cluster_0_10_12_14_2_4_6_8" time="13.00"/>
```

```
<tlsstate state="ggrggrrrrrrrrggggggrrrrrrrrrgrg"
phase="3" programID="RiLSA_2010"
id="cluster_0_10_12_14_2_4_6_8" time="14.00"/>
<tlsstate state="yyryyyrrrrrrrrgggggggrrrrrrrrrrrr"
phase="4" programID="RiLSA_2010"
id="cluster_0_10_12_14_2_4_6_8" time="34.00"/>
<tlsstate state="rrrrrrrrrrrrrgggggggrrrrrrrrrrrr"
phase="5" programID="RiLSA_2010"
id="cluster_0_10_12_14_2_4_6_8" time="37.00"/>
<tlsstate state="rrrrrrrrrrrrrgggggggrrrrrrrgggrg"
phase="6" programID="RiLSA_2010"
id="cluster_0_10_12_14_2_4_6_8" time="38.00"/>
<tlsstate state="rrrrrrrrrrrrryyyyyyrrrrrrrrrrrr"
phase="7" programID="RiLSA_2010"
id="cluster_0_10_12_14_2_4_6_8" time="50.00"/>
<tlsstate state="rrrrrruuuuuuuurrrrrruuuuuuurrrrr"
phase="8" programID="RiLSA_2010"
id="cluster_0_10_12_14_2_4_6_8" time="53.00"/>
<tlsstate state="rrrrrrgggggggrrrrrrgggggggggrgr"
phase="9" programID="RiLSA_2010"
id="cluster_0_10_12_14_2_4_6_8" time="54.00"/>
<tlsstate state="rrrrrryyyyyyyrrrrrryyyyyyyrrrrr"
phase="10" programID="RiLSA_2010"
id="cluster_0_10_12_14_2_4_6_8" time="70.00"/>
<tlsstate state="uuuuuurrrrrrrrrrrrrrrrrrrrrrr"
phase="11" programID="RiLSA_2010"
id="cluster_0_10_12_14_2_4_6_8" time="73.00"/>
<tlsstate state="gggggrrrrrrrrrrrrrrrrrrrrrggg"
phase="0" programID="RiLSA_2010"
id="cluster_0_10_12_14_2_4_6_8" time="74.00"/>
<tlsstate state="ggyggyrrrrrrrrrrrrrrrrrrrrrrr"
phase="1" programID="RiLSA_2010"
id="cluster_0_10_12_14_2_4_6_8" time="84.00"/>
<tlsstate state="ggrggrrrrrrrruuuuuurrrrrrrrrrr"
phase="2" programID="RiLSA_2010"
id="cluster_0_10_12_14_2_4_6_8" time="87.00"/>
<tlsstate state="ggrggrrrrrrrrgggggggrrrrrrrrrgrg"
phase="3" programID="RiLSA_2010"
id="cluster_0_10_12_14_2_4_6_8" time="88.00"/>
<tlsstate state="yyryyyrrrrrrrrgggggggrrrrrrrrrrr"
phase="4" programID="RiLSA_2010"
id="cluster_0_10_12_14_2_4_6_8" time="108.00"/>
```

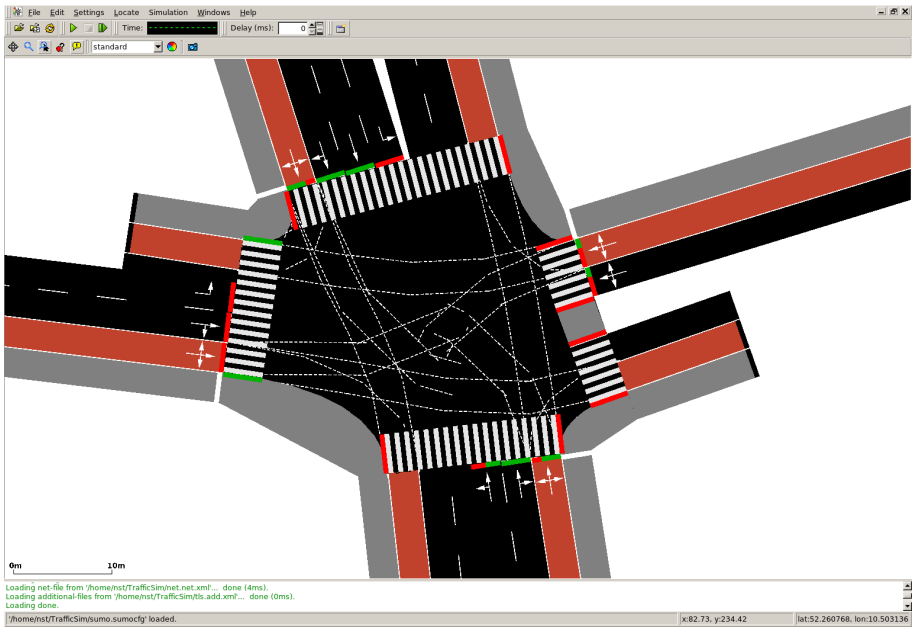

Abbildung A.1: Mit TrafficSim erzeugte Kreuzung K51

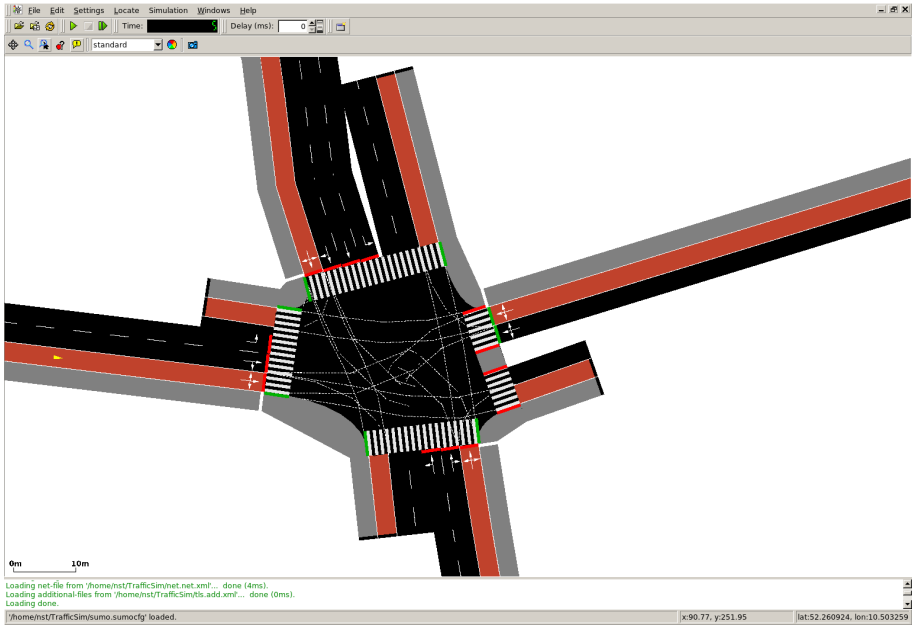

Abbildung A.2: Mit TrafficSim erzeugte Kreuzung für die Nachbildung eines realen LSA-Programms

Literaturverzeichnis

[36313] VDI-Richtlinie 3633. *Simulation von Logistik-, Materialfluss- und Produktionssystemen - Grundlagen.* VDI Verlag, 2013. (zitiert auf Seite 5)

[Bee15] Sven Beeckmann. *TrafficSim - Beschreibungssprache zur Konfiguration von Verkehrssimulationen zur Generierung von Testdaten.* 2015. (zitiert auf Seite 1, 2, 7, 9, 10, 11, 12, 13, 14, 15, 18, 30, 33, 35 und 36)

[Bro06] Brockhaus. *Enzyklopädie in 30 Bänden.* FA Brockhaus, 2006. 21.te Auflage. (zitiert auf Seite 3, 4, 5 und 8)

[Buna] Statitisches Bundesamt. *Fahrzeugbestand.* https://www.destatis.de/DE/ZahlenFakten/Wirtschaftsbereiche/TransportVerkehr/UnternehmenInfrastrukturFahrzeugbestand/Tabellen/Fahrzeugbestand.html,[Online: Stand 21.Februar 2016]. (zitiert auf Seite 1)

[Bunb] Statitisches Bundesamt. *Güterbeförderung.* https://www.destatis.de/DE/ZahlenFakten/Wirtschaftsbereiche/TransportVerkehr/Gueterverkehr/Tabellen/VerkehrstraegerHauptverkehrsRelationA.html,[Online: Stand 21.Februar 2016]. (zitiert auf Seite 1)

[Bunc] Statitisches Bundesamt. *Polizeilich erfasste Unfälle.* https://www.destatis.de/DE/ZahlenFakten/Wirtschaftsbereiche/TransportVerkehr/Verkehrsunfaelle/Tabellen/UnfaelleVerunglueckte.html,[Online: Stand 21.Februar 2016]. (zitiert auf Seite 1)

[Bund] Statitisches Bundesamt. *Verkehrsmittelbestand und Infrastruktur.* https://www.destatis.de/DE/ZahlenFakten/Wirtschaftsbereiche/TransportVerkehr/UnternehmenInfrastrukturFahrzeugbestand/Tabellen/Verkehrsinfrastruktur.html,[Online: Stand 21.Februar 2016]. (zitiert auf Seite 1)

[DLRa] DLR. *SUMO - Simulation of Urban MObility.* http://www.dlr.de/ts/de/desktopdefault.aspx/tabid-9883/16931_read-41000/,[Online: Stand 21.Februar 2016]. (zitiert auf Seite 6)

[DLRb] DLR. *SUMO User Documentation.* http://sumo.dlr.de/wiki/SUMO_User_Documentation,[Online: Stand 21.Februar 2016]. (zitiert auf Seite ix, 6, 7, 18, 19, 20, 21, 22, 36 und 39)

[Dre13] Patrick Dreßler. *Funksysteme, Protokolle und Anwendungen der Car-to-X-Kommunikation.* 2013. (zitiert auf Seite 7)

[For] Forschungsinformationssystem. *Lichtsignalanlagen.* http://www.forschungsinformationssystem.de/servlet/is/342328/,[Online: Stand 21.Februar 2016]. (zitiert auf Seite 5)

[fSuVAV10] Forschungsgesellschaft für Straßen-und Verkehrswesen Arbeitsgruppe Verkehrsmanagment. *Richtlinien für Lichtsignalanlagen RILSA Lichtzeichenanlagen für den Straßenverkehr.* FGSV Verlag GmbH, 2010. (zitiert auf Seite 4, 5, 17 und 38)

[Gra09] Felix Graf. *„Car 2 Car/Car 2 X" Kommunikation.* 2009. (zitiert auf Seite 1 und 7)

[iMe] ifak Magdeburg e.V. *Institut für Automation und Kommunikation.* https://www.ifak.eu/,[Online: Stand 21.Februar 2016]. (zitiert auf Seite 1 und 9)

[SL] Hans-Werner Schaal and Thomas Löffler. *Car2X – von der Forschung zur Serienentwicklung.* https://vector.com/portal/medien/cmc/press/PON/Car2x_ElektronikAutomotive_201212_PressArticle_DE.pdf,[Online: Stand 21.Februar 2016]. (zitiert auf Seite 7)

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

Magdeburg, den 29.03.2016