TUHH

Technische Universität Hamburg-Harburg

Diplomarbeit

Verhaltensmodellierung von Agenten mit Hybriden Systemen

Jakob Hennings

4. Oktober 2012

1. Prüfer:	Prof. Dr. Ralf Möller
2. Prüfer:	Prof. Dr. Herbert Werner
Betreuer:	Dipl.-Ing. Karsten Martiny

Technische Universität Hamburg-Harburg
Institute for Software Systems
Schwarzenbergstraße 95
21073 Hamburg

Inhaltsverzeichnis

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Hamburg, den 4. Oktober 2012 _____

Zusammenfassung

„Hybride Systeme“ sind mathematische Modelle, in denen sich Zustände kontinuierlich und diskret verändern können. Sie dienen der Beschreibung physikalischer Systeme, in die eine digitale Steuerung eingebettet ist. In dieser Arbeit beschreibe ich die Verifikation „Hybrider Systeme“ mit den Logiken d$\mathscr{L}$ und DAL und dem Theorembeweiser KeYmaera. Ich werde einfache Systeme beschreiben, die vollständig automatisiert verifiziert werden können, und komplexere, bei denen manuelle Eingriffe durch den Anwender erforderlich sind. Reale Systeme sind häufig so komplex, dass eine Verifikation nur mit einem verallgemeinerten Modell durchgeführt werden kann. Dies trifft auch für Quadrocopter zu, für die ich einen Ansatz zeige, mit dem die komplexe Bewegungsgleichung durch ein einfaches Modell abstrahiert werden kann.

1 Einleitung

"Wenn eine Aufgabe mehr als eine Möglichkeit des Ausganges hat und einer dieser Ausgänge in einem Desaster oder in einem nicht eben wünschenswerten Ergebnis enden wird, dann wird sich jemand finden, der genau diesen Weg nimmt." - (Murphy's Law)

Computer sind heutzutage allgegenwärtig. In vielen Haushaltsgeräten sind Mikrocontroller zu finden. Autos, Züge, Flugzeuge und andere Verkehrsmittel besitzen bereits eine Vielzahl unterschiedlicher digitaler Steuerungen. In der Industrie werden komplexe Aufgaben von autonomen Robotern selbständig oder in Interaktion mit Menschen durchgeführt. Digitale Regler haben sich gegenüber den analogen Reglern durchgesetzt. Sie besitzen eine deutlich höhere Regelgenauigkeit, können flexibel konfiguriert werden und lassen sich einfacher in digitale Systeme einbinden [1].

Jeden Tag werden bereits heute in vielen Bereichen sicherheitskritische Entscheidungen von Computern getroffen. Tritt aber ein Fehler auf, kann dies großen wirtschaftlichen Schaden oder sogar Verlust von Menschenleben bedeuten. Die Modellierung sicherheitskritischer Systeme ist daher mit besonders großer Sorgfalt vorzunehmen.

1.1 Motivation

Eine Herausforderung ist die Vereinigung der kontinuierlichen, physikalischen Welt mit der diskreten und quantisierten Welt im Computer. Die physikalische Welt wird i.d.R. mit Differentialgleichungssystemen beschrieben. In der Softwareentwicklung werden hingegen endliche Zustandsautomaten eingesetzt. Die in dieser Arbeit behandelten „Hybriden Systeme" vereinen diese beiden Konzepte. So wird es möglich, in einem Modell zu beschreiben, wie ein Computer zu diskreten Zeitpunkten Sensordaten einliest und Stellgrößen setzt, und wie sich der physikalische Zustand eines Systems in der Zeit dazwischen kontinuierlich verändert. Zweck der Modellierung in dieser Arbeit ist die maschinelle Verifikation. Im Gegensatz zum Test oder zur Simulation soll durch die Verifikation formal bewiesen werden, dass die gewünschte Spezifikation unter allen Umständen eingehalten wird. Durch Test oder Simulation kann in realen System meist nur ein kleiner Teil aller möglichen Zustände, die das System

annehmen kann, überprüft werden. Das Problem ist die zu große Anzahl möglicher Zustände und der exponentielle Wachstum dieser in Abhängigkeit von der Anzahl der variablen Größen (Zustandsexplosion). Gelingt die maschinelle Verifikation praxisrelevanter Systeme, kann der Entwicklungsprozess sicherheitskritischer Systeme deutlich effizienter gemacht werden. Dies betrifft z.B. Assistenzsysteme für den Straßen-, Luft- oder Schienenverkehr, die zum einen für eine höhere Sicherheit sorgen und zum anderen zu einer besseren Ausnutzung des Verkehrsnetzes führen können [2].

1.2 Ziel

In dieser Arbeit untersuche ich den von André Platzer in [2] eingeführten Kalkül für die von ihm entwickelten Logiken d$\mathscr{L}$ und DAL. Der Kalkül macht die maschinelle Verifikation Hybrider Systeme möglich. Er ist in der Software KeYmaera [3] implementiert, um Eigenschaften Hybrider Systeme vollständig automatisiert oder in Interaktion mit dem Anwender zu beweisen. Ziel dieser Arbeit ist es zu zeigen, wie diese Software und der Kalkül eingesetzt werden. Es geht um die Fragestellungen, wie komplex Hybride Systeme sein dürfen, damit die Verifikation vollständig automatisiert abläuft, und welche Probleme auftreten können. Ich werde beschreiben, wie man reale Systeme durch Verallgemeinerung vereinfachen und damit verifizierbar machen kann und einen Ansatz zeigen, mit dem Quadrocopter mit sehr komplizierten Bewegungsgleichungen durch ein einfaches Modell abstrahiert werden können.

Die in dieser Arbeit gezeigten Methoden finden ihren Einsatz in der Modellierungsphase. Sie dienen der Überprüfung von Spezifikationen und sind dazu in der Lage Verletzungen von Sicherheitsbedingungen frühzeitig aufzudecken. Es findet aber keine Verifikation einer konkreten Implementierung eines Systems statt, bei der u.a. auch Speicherzugriffsverletzungen und andere Fehlerquellen zu berücksichtigen sind. Die Überprüfung, ob eine konkrete Implementierung die gleichen Sicherheitsbedingungen wie das dazugehörige Modell erfüllt, wird in dieser Arbeit nicht weiter behandelt.

2 Aufbau dieser Arbeit

In Kapitel 3 werden die Begriffe „Hybride Systeme“, „Hybride Automaten“ und „Hybride Programme“ eingeführt. Mit einfachen Beispielen wird informell gezeigt, wie Hybride Automaten und Hybride Programme aufgebaut sind. Des Weiteren werden einige theoretische Grundlagen beschrieben, u.a. der Sequenzenkalkül und die Quantorenelimination, die dem Verständnis der folgenden Kapitel dienen.

In Kapitel 4 werden die Logiken d$\mathscr{L}$, DAL und dTL eingeführt. Das Kapitel enthält die Syntax für Formeln dieser Logiken und den Aufbau der Quelldateien für den Theorembeweiser KeYmaera. Zudem werden die wichtigsten Regeln des Kalküls und die von KeYmaera verwendeten Strategien bei der Suche nach einem Beweis gezeigt.

Kapitel 5 enthält Anwendungsfälle zur Kollisionsvermeidung zwischen Fahrzeugen auf der Straße und in der Luft. Es werden Modelle im Detail beschrieben und auf die Schwierigkeiten bei der Modellierung und der Verifikation hingewiesen. Zur Modellierung eines Quadrocopters wird ein Ansatz gezeigt, mit dem die komplexe Bewegungsgleichung deutlich vereinfacht werden kann.

In Kapitel 6 wird gezeigt, wie sich der Theorembeweiser KeYmaera bei Erreichbarkeits- und Sicherheitsanalysen mit einfachen Modellen unterschiedlicher Komplexität verhält.

In Kapitel 7 wird die Logik QdL beschrieben. In den Logiken d$\mathscr{L}$, DAL und dTL ist es notwendig jede Instanz eines Fahrzeuges explizit anzugeben. Dies führt dazu, dass Modelle mit vielen Fahrzeugen zu komplex werden und Modelle mit beliebig vielen Fahrzeugen gar nicht möglich sind. Dieser Nachteil wird mit der Logik QdL beseitigt.

In Kapitel 8 befinden sich Beispiele, die u.a. zeigen, wie die Stabilität eines PID-Reglers nachgewiesen werden kann und wie es möglich ist, die Betragsfunktion zu definieren.

Kapitel 9 enthält eine Zusammenfassung und einen Ausblick. Im Anhang sind Quelltexte und die Beschreibung des von mir für die Verifikationen verwendeten Compu-

tersystems zu finden. Die Quelltexte stehen zudem auf der beigelegten CD in digitaler Form zur Verfügung.

3 Grundlagen

Die in diesem Kapitel beschriebenen Grundlagen dienen dem Verständnis der darauf folgenden Kaptiel.

3.1 Hybride Systeme

Ein Hybrides System ist ein allgemeines mathematisches Modell, bei dem sich Zustände kontinuierlich und diskret verändern können [2, S. 4]. Die Vereinigung von kontinuierlichen und diskreten Zustandsänderungen in ein Modell macht es möglich, dass ein physikalisches System mit eingebetteter digitaler Steuerung (im Folgenden Controller) als Ganzes modelliert und verifiziert werden kann. Die Auswirkungen von in der digitalen Welt bekannten Phenomänen (wie z.B. Quantisierungsfehler, Delay, Jitter) auf das physikalische Verhalten des Systems können somit analysiert werden.

Zwei Darstellungsformen von Hybriden Systemen sind Hybride Automaten und Hybride Programme. Hybride Automaten sind anschaulich und können Sachverhalte schnell verständlich machen. Hybride Programme hingegen sind prägnanter und aufgrund der rein textuellen Darstellung leichter maschinell zu verarbeiten. Im Folgenden werden beide Notationsformen beschrieben.

3.1.1 Hybride Automaten

Ein Hybrider Automat ist eine Notation für Hybride Systeme [2, S. 5]. In [5] ist eine formale Definition für Hybride Automaten zu finden, die anhand eines einfachen Beispiels eingeführt wird.

In Abbildung 3.1 ist ein vereinfachter Hybrider Automat für einen Notbremsassistenten für Kraftfahrzeuge dargestellt. Die Aufgabe des Notbremsassistenten ist es eine Vollbremsung einzuleiten, wenn das Fahrzeug sonst mit einem Hindernis kollidieren würde.

Ein Hybrider Automat ist ein gerichteter Graph. Mit den gerichteten Kanten des Graphens werden diskrete Zustandsübergänge beschrieben. Sie sind gekennzeichnet mit Bedingungen (hier: $p \leq safe$ und $p > safe$) und diskreten Wertzuweisungen (hier: $a := *$, $a := 0$ und $t := 0$). Damit ein diskreter Zustandsübergang ausgeführt werden

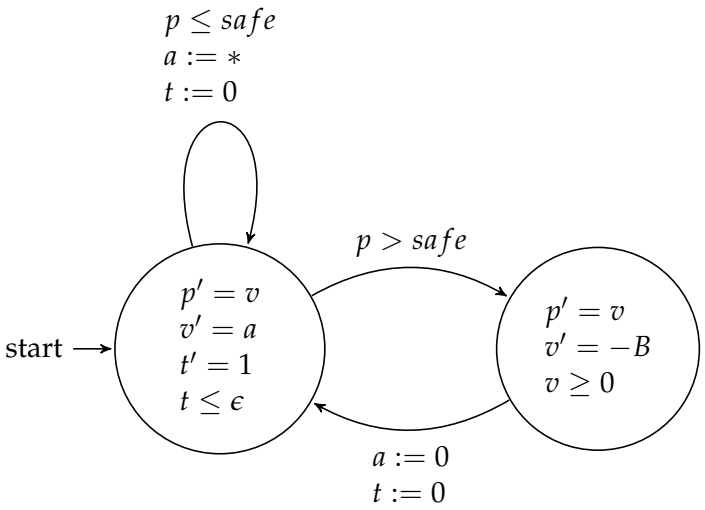

Abbildung 3.1: Hybrider Automat für einen Notbremsassistenten

darf, müssen alle ihm zugeordneten Bedingungen wahr sein. Hat der diskrete Zustandsübergang keine Bedingung, so darf er ohne Einschränkung ausgeführt werden, wenn sich der Automat in dem Knoten befindet, vom dem der diskrete Zustandsübergang ausgeht. Die Ausführung eines diskreten Zustandsüberganges geschieht ohne Zeitverlust. Mit den soeben beschriebenen Mitteln können bereits die aus der Informatik bekannten endliche Automaten erstellt werden.

Im Vergleich zu den endlichen Zustandsautomaten beschreibt ein Knoten nicht einen einzelnen Zustand, sondern eine Menge von Zuständen. Innerhalb eines Knotens kann sich der Zustand des Automaten kontinuierlich verändern. Sie sind gekennzeichnet mit Bedingungen (hier: $t \leq \epsilon$ und $v \geq 0$) und Differentialgleichungen (hier: $p' = v$, $v' = a$, $v' = -B$ und $t' = 1$), die den kontinuierlichen Zustandsübergang beschreiben. Der Automat darf solange in einem Knoten verweilen, solange alle ihm zugeordneten Bedingungen wahr sind. Hat der Knoten keine Bedingung, so darf der Automat dauerhaft in dem Knoten bleiben. Allerdings darf er den Knoten auch verlassen, bevor eine ihm zugeordnete Bedingung falsch wird.

Ein Hybrider Automat ist grundsätzlich nicht deterministisch. Zum einen können von einem Knoten mehrere diskrete Zustandsübergänge ausgehen, deren Bedingungen gleichzeitig wahr sind, zum anderen ist der Zeitpunkt, wann der Automat einen Knoten verlässt nicht exakt bestimmt.

In dem Hybriden Automaten in Abbildung 3.1 sind die physikalischen Eigenschaften eines Kraftfahrzeuges und die Eigenschaften eines digitalen Controllers in einem Modell beschrieben. Mit Differentialgleichungen ($p' = v$ und $v' = a$) ist beschrieben,

wie sich das Kraftfahrzeug bewegt. Mit diskreten Zustandsübergängen und mit den Knoten zugeordneten Bedingungen wird beschrieben, wie der digitale Controller zu diskreten Zeitpunkten Zustandsänderungen vornimmt.

Der Zustand des Automaten wird mit den reellwertigen Variablen p, v, a und t beschrieben. Er besitzt zwei Knoten. Der linke Knoten beschreibt das physikalische Verhalten des Fahrzeuges bei normaler Fahrt ($p' = v$, $v' = a$), der rechte das Verhalten bei einer Vollbremsung ($p' = v$, $v' = -B$). Der Automat startet in dem linken Knoten. Durch die Differentialgleichung $t' = 1$ und mit der Bedingung $t \leq \epsilon$ wird erreicht, dass er maximal ϵ Zeiteinheiten in diesem Knoten verweilen darf. Danach muss der Controller einen diskreten Zustandsübergang wählen. Wenn $p \leq safe$ gilt, muss er den oberen diskreten Zustandsübergang wählen, kann die Beschleunigung a (auf einen beliebigen Wert) setzen und der Automat darf erneut für ϵ Zeiteinheiten in dem linken Knoten bleiben. Hierzu ist es zusätzlich notwendig t auf 0 zurückzusetzen. Gilt $p > safe$, so muss der Controller in den rechten Knoten wechseln. Das Fahrzeug führt dann eine Vollbremsung aus, bis es zum Stillstand kommt ($v \geq 0$). Danach wechselt es wieder in den linken Knoten und setzt die Beschleunigung a und die Zeitvariable t auf 0. Sobald die Bedingung $p \leq safe$ wieder erfüllt ist, kann das Fahrzeug weiterfahren. $safe$ steht für die Position, ab der das Fahrzeug bremsen muss, damit es vor einem Hindernis rechtzeitig zum stehen kommt. Wie dieser Wert ermittelt werden kann, wird in Kapiteln (5.1.2) erläutert.

Der hier beschriebene Automat ist nicht vollständig und enthält einen Fehler. Zum einen ist es notwendig, dass initial $p \leq safe$ und $t = 0$ gilt, zum anderen darf der Automat den rechten Knoten bereits verlassen, bevor das Fahrzeug zum Stillstand gekommen ist. Die Bedingung $v \geq 0$ sagt nur aus, wann der Automat den Knoten spätestens verlassen muss, er darf ihn aber früher verlassen. Das Fahrzeug würde dann mit bis zu ϵ Zeiteinheiten mit der Beschleunigung $a = 0$ weiterfahren und erst danach wieder eine Vollbremsung machen. Durch die in Kapitel 4 beschriebene Verifikation sollen derartige Unvollständigkeiten und Fehler in Hybriden Systemen aufgedeckt werden.

3.1.2 Hybride Programme

Eine weitere Notation für Hybride Systeme sind Hybride Programme. In Abbildung 3.2 ist eine 1:1 Übersetzung des in Abbildung 3.1 gezeigten Automaten dargestellt. Durch eine direkte Übersetzung aus Hybriden Automaten werden Hybride Programme häufig unnötig komplex [6]. In Abbildung 3.3 ist der Notbremsassisten in deutlich kompakterer Form modelliert. Die gewählte Notation kann somit Einfluss auf das Modell haben. Der Vorteil Hybrider Programme ist die kompaktere Darstellung und die einfachere Möglichkeit der maschinellen Verarbeitung.

$$
\begin{aligned}
NBA_{ha} \equiv (\quad & if(state = 0)then \\
& \quad \{p' = v, v' = a, t' = 1, t \leq \epsilon\}; \\
& \quad if(p \leq safe)then \\
& \quad\quad a := *; \\
& \quad\quad t := 0 \\
& \quad else \\
& \quad\quad state := 1 \\
& \quad fi \\
& else \\
& \quad \{p' = v, v' = -B, v \geq 0\}; \\
& \quad a := 0; \\
& \quad t := 0; \\
& \quad state := 0 \\
& fi \qquad)^*
\end{aligned}
$$

Abbildung 3.2: Hybrides Programm für einen Notbremsassistenen, 1:1 Übersetzung des Hybriden Automaten aus Abbildung 3.1

In dem Hybriden Programm NBA (Abbildung 3.3) ist angegeben, dass die Unterprogramme *ctrl* und *drive* sequentiell ausgeführt werden. Mit dem $*$ wird ausgedrückt, dass sich diese Sequenz beliebig oft wiederholen kann (vergl. reguläre Ausdrücke). In dem Unterprogramm *ctrl* wird die Sicherheitsbedingung $p \leq safe$ ausgewertet. Ist sie erfüllt, so wird a auf einen beliebigen Wert gesetzt. Ist sie nicht erfüllt, so wird a auf $-B$ gesetzt, um eine Vollbremsung einzuleiten. Das Unterprogramm *drive* gibt mit der Variable t den Takt des Controllers vor und beschreibt die kontinuierliche Zustandsänderung des Hybriden Systems.

$$
\begin{aligned}
NBA &\equiv (ctrl; drive)^* \\
ctrl &\equiv if\ (p \leq safe)\ then\ a := *\ else\ a := -B\ fi \\
drive &\equiv t := 0; \{p' = v, v' = a, t' = 1, t \leq \epsilon, v \geq 0\}
\end{aligned}
$$

Abbildung 3.3: Hybrides Programm für einen Notbremsassistenen in kompakterer Form

Die Syntax für Hybride Programme in d$\mathscr{L}$, DAL und dTL wird in Kapitel 4.2.2 vollständig eingeführt.

3.2 Sequenzenkalkül

Der erste Sequenzenkalkül wurde im Jahr 1934 von Gerhard Gentzen entwickelt [7]. Ein Sequenzenkalkül ist eine Menge von Regeln, durch deren Anwendung mathematische Aussagen einer bestimmten Logik bewiesen werden können [8]. Auch für die in dieser Arbeit behandelten Logiken d$\mathscr{L}$, DAL und dTL wird bei der Suche nach Beweisen ein Sequenzenkalkül verwendet (s. Abschnitt 4.2.3). Im Folgenden werde ich den grundsätzlichen Aufbau der Regeln beschreiben.

Sequenz

Eine Sequenz hat die Form

$$\Gamma \vdash \Delta \quad . \tag{3.1}$$

Γ wird Antezedent und Δ Sukzedent genannt. Antezedent und Sukzedent sind Mengen von Formeln. Auch die leere Menge ist zulässig. Seien $\gamma_1, ..., \gamma_m, \delta_1, ..., \delta_n$ Formeln, dann hat die Sequenz

$$\gamma_1, ..., \gamma_m \vdash \delta_1, ..., \delta_n \tag{3.2}$$

die Bedeutung, dass aus γ_1, γ_2, ... und γ_m folgt, dass δ_1, δ_2, ... oder δ_n gilt. Der Antezedent ist somit eine Konjunktion, der Sukzedent eine Disjunktion und das Symbol $\vdash$ drückt eine Implikation aus. Die leere Menge steht im Antezedent für *true* und im Sukzedent für $false$. Statt $\vdash \Delta$ kann man also auch $true \vdash \Delta$ und statt $\Gamma \vdash$ auch $\Gamma \vdash false$ schreiben. $\vdash \Delta$ drückt damit aus, dass Δ immer wahr ist, $\Gamma \vdash$ hingegen, dass Γ immer falsch ist.

Axiom

Sequenzen mit der Form

$$\Gamma \cup \{\phi\} \vdash \Delta \cup \{\phi\} \tag{3.3}$$

sind Axiome, da sie offensichtlich wahr sind. Wenn ϕ bereits eine Annahme ist, dann ist ϕ auch eine gültige Schlussfolgerung.

Regel

Eine Regel hat den Aufbau

$$\frac{\Gamma_1 \vdash \Delta_1 \quad ... \quad \Gamma_n \vdash \Delta_n}{\Gamma \vdash \Delta} \quad . \tag{3.4}$$

Oberhalb des Striches stehen n Prämissen, und unterhalb des Striches steht eine Konklusion, die aus den Prämissen geschlossen werden kann.

Herleitung

Eine Herleitung ist ein Baum, dessen Knoten mit Sequenzen beschriftet sind. Die Sequenz eines Knotens und die Sequenzen der Kinder dieses Knotens müssen dabei Konklusion bzw. Prämissen einer Regel des verwendeten Kalküls sein. Die Wurzel

des Baums enthält die hergeleitete Sequenz. Enthalten die Blätter des Baums Axiome, so ist die Herleitung ein Beweis [9].

Beweis

Die Darstellung der baumartigen Herleitung wird auch als Tableau bezeichnet. Die Wurzel des Baumes befindet sich im Tableau ganz unten. Die Suche nach einem Beweis erfolgt somit von unten nach oben.

Tableau

3.3 Relative Vollständigkeit

Das Symbol $\models$ ist der semantische Folgerungsoperator [10]. Mit $\Gamma \models \Delta$ wird ausgedrückt, dass bei jeder Belegung der variablen Größen, bei der Γ gilt, auch Δ gilt. Das Symbol $\vdash$ ist der syntaktische Ableitungsoperator [10]. Mit $\Gamma \vdash \Delta$ wird ausgedrückt, dass Δ mit Regeln eines Kalküls aus Γ abgeleitet werden kann. Das Symbol $\vdash$ wird leider nicht einheitlich verwendet. Im Sequenzenkalkül wird der horizontale Strich als Ableitungsoperator verwendet und das Symbol $\vdash$ hingegen mit der Bedeutung der Implikation um Antezedent und Sukzedent voneinander zu trennen.

$\models$

$\vdash$

Ein Kalkül ist korrekt, wenn sich nur semantisch gültige Sätze der zugrunde liegenden Logik ableiten lassen [10]. D.h.

Korrektheit (Soundness)

$$\text{wenn} \quad \Gamma \vdash \Delta \quad \text{gilt, dann gilt auch} \quad \Gamma \models \Delta \ .$$

Ein Kalkül ist vollständig, wenn sich alle semantisch gültigen Sätze der zugrunde liegenden Logik ableiten lassen [10]. D.h.

Vollständigkeit (Completeness)

$$\text{wenn} \quad \Gamma \models \Delta \quad \text{gilt, dann gilt auch} \quad \Gamma \vdash \Delta \ .$$

Bevor ein Kalkül verwendet wird, sollte nachgewiesen sein, dass er korrekt ist, da mit einem nicht korrekten Kalkül ein Beweis für eine falsche Aussage gefunden werden kann. Ist ein Kalkül zudem vollständig, so kann man mit ihm für jede gültige Aussage einen Beweis finden. Es gibt allerdings Logiken, für die es keinen vollständigen Kalkül geben kann. Auch die in dieser Arbeit beschriebenen Logiken sind nicht vollständig. Für die Hoare Logik wurde 1974 von Cook der Begriff der relativen Vollständigkeit (relative completeness) eingeführt [11, 12]. Mit der relativen Vollständigkeit für die Hoare Logik machte Cook die Aussage

$$\text{wenn} \quad \models \{A\}c\{B\} \quad \text{gilt, dann gilt auch} \quad \vdash \{A\}c\{B\} \ .$$

$\{A\}c\{B\}$ ist ein Hoare Triple, für das nur partielle Korrektheit nachgewiesen werden kann. Die Aussage von Cook drückt somit aus: Wenn eine Hoare Triple partiell korrekt ist, so kann dies auch mit der Hoare Logik bewiesen werden. Partiell korrekt bedeutet, dass die Terminierung von Schleifen nicht nachgewiesen wird. Auch für die in dieser

Arbeit behandelten Logiken kann relative Vollständigkeit nachgewiesen werden (s. Kapitel 4.2.4).

3.4 Quantorenelimination

Bei der Suche nach einem Beweis ist die Quantorenelimination ein wichtiges Hilfsmittel. Eine Formel mit Quantoren ($\forall$ oder $\exists$), die in Prädikatenlogik erster Ordnung vorliegt, kann durch Quantorenelimination vereinfacht werden. Wolfram Mathematica® (im Folgenden nur noch Mathematica) führt durch die Befehle *Reduce* und *Resolve* Quantorenelimination aus.

In Gleichung 3.5 ist zu sehen, dass man die Frage nach der Existenz einer Nullstelle auch mit einer quantorenfreien Formel ausdrücken kann[2].

$$\begin{aligned} QE(\exists x(ax^2 + bx + c = 0)) \;\equiv\; & (a \neq 0 \wedge b^2 - 4ac \geq 0) \;\vee \\ & (a = 0 \wedge (b = 0 \rightarrow c = 0)) \end{aligned} \tag{3.5}$$

Der Mathematica Aufruf sieht wie folgt aus:

```
Resolve[Exists[x, a x^2 + b x + c == 0], Reals]   .
```

Quantorenelimination hat einen starken Einfluss auf die Dauer der Suche nach einem Beweis (s. Kapitel 4.2.5). In [13] wird gesagt, dass die reelle Quantorenelimination im schlimmsten Fall mindestens doppelt exponentiell ist.

Neben Mathematica gibt es weitere Programme, die Quantorenelimination implementieren und mit dem Theorembeweiser KeYmaera verwendet werden können. Hierzu zählen z.B. Redlog [14], QEPCAD B [15] und Cohen-Hörmander Quantifier Elimination [16]. Die Autoren von KeYmaera empfehlen allerdings die Verwendung von Mathematica [3].

4 Verifikation Hybrider Systeme

Mit der Verifikation von Hybriden Systemen soll formell nachgewiesen werden, ob sie ihre Spezifikation erfüllen. Wird die Spezifikation mit einer geeigneten Logik formuliert, kann die Verifikation durch das Anwenden von Regeln eines Kalküls erfolgen. Die Logiken d$\mathscr{L}$, DAL und dTL wurden für diesen Zweck entwickelt und ihr Kalkül in der Software KeYmaera implementiert. Im Folgenden beschreibe ich die Syntax der Logiken, den Kalkül und die Strategie, mit der KeYmaera eine Verifikation automatisiert durchführt.

4.1 Allgemeine Vorgehensweise

In dieser Arbeit werden Hybride Systeme als Hybride Programme modelliert. Typische Probleme sind Sicherheits- und Erreichbarkeitsanalysen. Mit einer Sicherheitsanalyse wird überprüft, ob ein Hybrides System unter gar keinen Umständen in bestimmte Zustände gelangen kann. Mit einer Erreichbarkeitsanalysen soll hingegen gewährleistet werden, dass alle gewünschten Zustände erreicht werden können. Mit einer Sicherheitsanalyse kann beispielsweise überprüft werden, ob für ein Fahrzeug mit Notbremsassistenten (s. Abbildung 3.3) eine Kollision mit einem Hindernis ausgeschlossen ist. Eine Erreichbarkeitsanalyse könnte zeigen, dass das Fahrzeug *trotz* Notbremsassistenten dazu in der Lage ist, alle sicheren Positionen anzufahren. Beide Analysen sind somit wichtig. Fährt ein Fahrzeug aufgrund eines Modellierungsfehlers nicht, könnte es eine Sicherheitsanalyse leicht bestehen, sollte dann aber bei einer Erreichbarkeitsanalyse durchfallen.

$$\phi_0 \rightarrow [NBA]\psi_0 \qquad \text{Sicherheitsanaylse} \tag{4.1}$$

$$\phi_1 \rightarrow \langle NBA\rangle\psi_1 \qquad \text{Erreichbarkeitsanalyse} \tag{4.2}$$

Eine Sicherheits- und eine Erreichbarkeitsanalyse für den Notbremsassistenten sind in Formel 4.1 und 4.2 zu sehen. Für die Analysen wurde das Hybride Programm aus Abbildung 3.3 in eine d$\mathscr{L}$-Formel (s. Abschnitt 4.2.1) eingesetzt. ϕ_0 und ϕ_1 sind Vorbedingungen, die vor Ausführung des Programms als *wahr* angenommen werden. An dieser Stelle wird beispielsweise formuliert, welche maximale Beschleunigung das

Fahrzeug hat oder wie weit es initial vom Hindernis entfernt sein muss. ψ_0 und ψ_1 sind Nachbedingungen, die nach Ausführung des Programms erfüllt sein sollen. Im Falle der Kollisionsvermeidung steht hier die Aussage „Es hat keine Kollision stattgefunden“ bzw. „Fahrzeug X hat Position Y erreicht“.

Die eckigen Klammern ($[...]$) in der Sicherheitsanalyse drücken aus, dass die Nachbedingung bei allen möglichen Programmabläufen erfüllt sein muss. Mit den spitzen Klammern ($\langle ... \rangle$) wird angegeben, dass es mindestens einen Programmablauf geben muss, der die Nachbedingung erfüllt.

4.2 KeYmaera

KeYmaera ist ein Open-Source Theorembeweiser für Hybride Systeme und kann unter [3] heruntergeladen werden. Der Theorembeweiser wurde entwickelt um die Gültigkeit von Anforderungen an Hybride Systeme automatisiert zu beweisen. Es ist aber möglich interaktiv in die Suche nach einem Beweis einzugreifen. Als Eingabe erwartet KeYmaera ein Verifikations-Problem (wie es z.B. in Formel 4.1 oder 4.2 gegeben ist), das dem Theorembeweiser als Quelldatei im ANSI-Format bereitgestellt wird. Im Folgenden wird beschrieben, wie diese Quelldatei aufgebaut ist, und die Syntax der von KeYmaera unterstützten Logiken d$\mathscr{L}$, DAL und dTL wird eingeführt. In der von mir genutzten Literatur wird häufig eine verkürzte, mathematische Syntax verwendet, der ich mich aufgrund der besseren Lesbarkeit in dieser Arbeit auch bedient habe. In der Quelldatei müssen allerdings einige mathematische Symbole durch Synonyme ersetzt werden, da diese sonst nicht dargestellt werden könnten. Diese Synonyme und weitere Unterschiede zwischen verkürzter, mathematischer Syntax und Syntax in der Quelldatei werden in diesem Abschnitt im Detail beschrieben. Einige Beispiele in dieser Arbeit enthalten auch mathematische Ausdrücke wie die euklidische Norm ($\|...\|_2$) oder Ausdrücke der Vektorrechnung. Diese Ausdrücke werden so nicht unterstützt, sondern müssen in einfachere umgeformt werden. Die Umformungen sollten dem Leser aber klar sein und werden daher nicht näher erläutert. Als Quelle für die Syntax dienten mir [2], [6] und die in dem Theorembeweiser enthaltenen Beispiele. Für den exakten Aufbau der Quelldatei habe ich leider keine Definition gefunden, so dass ich nur den grundsätzlichen Aufbau aus den mir gegebenen Beispielen ableiten konnte.

4.2.1 Die Logiken d$\mathscr{L}$, DAL und dTL

KeYmaera unterstützt die Logiken

- Differential Dynamic Logic d$\mathscr{L}$,
- Differential-Algebraic Dynamic Logic DAL und
- Differential Temporal Dynamic Logic dTL.

Die Logik d$\mathscr{L}$ kann für Hybride Programme mit einfachen, lösbaren Differentialgleichungen verwendet werden. Die Logiken DAL und dTL sind Erweiterungen der Logik d$\mathscr{L}$. Mit DAL können Differential-Ungleichungen verwendet werden ($p' \leq v$) und es ist nicht mehr notwendig die Lösung einer Differentialgleichung zu bestimmen. Die Logik dTL erweitert d$\mathscr{L}$ um die Temporale Logik.

4.2.2 Syntax und Semantik

In diesem Abschnitt beschreibe ich die Syntax und Semantik der von KeYmaera unterstützten Logiken d$\mathscr{L}$, DAL und dTL und den Aufbau einer Quelldatei. Ich führe alle Logiken gemeinsam ein und werde ggf. darauf hinweisen, wenn die Operatoren nur in DAL oder dTL zur Verfügung stehen. Eine Quelldatei enthält ein Verifikations-Problem, dass durch eine d$\mathscr{L}$-, DAL oder dTL-Formel beschrieben wird. In der Formel ist das zu verifizierende Hybride Programm enthalten.

KeYmaera unterstützt nur die Datentypen „Boolean“ und „Reelle Zahl“ [1]. Der Datentyp „Boolean“ tritt nur als Rückgabewert von Prädikaten oder im Quelltext durch die Bezeichner *true* und $false$ auf. In allen anderen Fällen kann nur der Datentyp „Reelle Zahl“ verwendet werden. KeYmaera rechnet exakt, so dass bei der Suche nach einem Beweis für eine Formel Rundungsfehler, Auslöschung oder andere Phänomen, wie sie aus der Gleitkommaarithmetik bekannt sind, nicht auftreten können. Im Folgenden definiere ich Zustandsvariablen, Konstanten, Prädikate, logische Variablen und Funktionen. Sie bilden die Grundlage für Terme, Formeln der Prädikatenlogik erster Ordnung, Diskrete Sprungbedingungen, Differentiell-algebraische Bedingungen, Hybride Programme und Formeln der Logiken d$\mathscr{L}$, DAL und dTL.

Zustandsvariablen

Zustandsvariablen sind reellwertige Variablen, die den Zustand eines Hybriden Programms beschreiben. Sie werden in der Quelldatei global in dem Block „\programVariables{...}“ oder am Anfangs des Blocks „\problem{...}“ in eckigen Klammern ([...])

[1] Ich beschreibe hier KeYmaera in der Version 2.0. Während meiner Arbeit wurde die Version 2.1 veröffentlicht und der Datentyp „Sorte“ eingeführt (s. Kapitel 7).

mit „R a;“ definiert (s. Listing 4.1). Ihnen kann mit $a := \theta$ zu einem diskreten Zeitpunkt ein Wert zugewiesen werden oder sie können sich kontinuierlich durch Angabe einer Differentialgleichung verändern, z.B. $v' = a$.

Konstanten

Konstanten werden im Block „\functions{...}“ deklariert oder im Quelltext als rationale Zahl angegeben (im Dezimalformat mit „.“ als Trennzeichen). Der Wert einer deklarierten Konstanten kann in der Vorbedingung definiert werden. Es ist aber nicht erforderlich den Wert eindeutig oder überhaupt festzulegen. So sind Definitionen wie „$(g = 9.81) \vee (g = 3.69)$“ oder „$(g = 9.81 + h) \wedge (-0.1 < h < 0.1)$“ nützlich, um ein System für unterschiedlichen Umgebungen oder mit einer Toleranz zu modellieren. Der Unterschied einer Konstanten zur Variablen ist, dass ihr Wert zur Ausführungszeit nicht verändern werden kann. Konstanten werden in KeYmaera als 0-stellige Funktionen betrachtet. Sie repräsentieren immer eine reellwertige Zahl.

Prädikate

Prädikate sind die binären logischen Operatoren =, $\neq$, $\leq$, <, $\geq$ und > mit den allgemein üblichen Bedeutungen. Sie erwarten reellwertige Parameter und geben den Wert *true* oder $false$ zurück.

Logische Variablen

Logische Variablen sind reellwertige Variablen, über die quantifiziert wird. Sie werden im Quelltext direkt hinter einem Quantor ($\forall$ bzw. $\exists$) definiert („\forall R x“ oder „\exists R x“). Durch Regeln des Kalküls können auch nicht quantifizierte logische Variablen entstehen.

Funktionen

Funktionen sind die binären arithmetischen Operatoren +, -, $\cdot$, /, a^b (mit $b \in \mathbb{Z}$ [2, S. 30]) und die unären Operatoren -, $sin(x)$, $cos(x)$ und $tan(x)$ mit den allgemein üblichen Bedeutungen. Zudem können weitere Funktionen in dem Block „\functions{...}“ deklariert werden (z.B. „R Foo(R, R);“). Der Rückgabewert und die Parameter einer Funktion sind immer reellwertig. Die Definition einer Funktion in KeYmaera ist nicht zwingend erforderlich, ausreichend ist die Deklaration. Beispielsweise kann durch die Deklaration der Funktion „R Abs(R);“ die in Mathematica definierte Funktion mit dem Bezeichner „Abs“ in KeYmaerae nutzbar gemacht werden. Durch die Vorbedingung

$$\forall Rx; ((x < 0 \rightarrow Abs(x) = -x) \wedge (x \geq 0 \rightarrow Abs(x) = x))$$

kann die Betragsfunktion aber auch für KeYmaera definiert werden.

In Abbildung 4.1 ist die Syntax für Prädikate und Funktionen im Quelltext angegeben. In [2] werden Zustandsvariablen, Konstanten, Prädikate und Funktionen in der Menge Σ zusammengefasst. Zustandsvariablen und Konstanten werden dabei als 0-stellige Funktionen betrachtet. Logische Variablen bilden die Menge V. Diese Definitionen werde ich im Folgenden ebenfalls verwenden. Σ, V

$=$	`=`	$+$	`+`
$\neq$	`!=`	$-$	`-`
$\leq$	`<=`	$\cdot$	`*`
$<$	`<`	a^x	`a^x`
$\geq$	`>=`	$sin(x)$	`sin(x)`
$>$	`>`	$cos(x)$	`cos(x)`
		$tan(x)$	`tan(x)`

Abbildung 4.1: Syntax für Funktionen und Prädikate im Quelltext

Terme

Sei $x \in V$ eine logische Variable und $f(\theta_1, ..., \theta_n) \in \Sigma$ eine Funktion mit n reellwertigen Parametern, dann sind Terme wie folgt definiert:

$$\theta ::= x \mid f(\theta_1, ..., \theta_n). \tag{4.3}$$

f umfasst hier die Menge der Zustandsvariablen, Konstanten, Prädikate und Funktionen. Hierzu gehören somit auch die logischen und arithmetischen Operationen ($=$, $\leq$, ..., $+$, $-$, ...), die ich in dieser Arbeit (wie allgemein üblich) in Infix-Notation verwende. Beispiele für Terme sind

- $x + 5y \cdot (x - 3.3y + za)$ mit x, y, z und a als Zustandsvariablen oder Konstanten und 5 und 3.3 als Konstanten und
- $x^2 \leq sin(y)$ mit x und y als Zustandsvariablen oder Konstanten, 2 als Konstante und $sin(R)$ als 1-stellige Funktion.

Formeln der Prädikatenlogik erster Ordnung (first-order logic)

Mit der Prädikatenlogik erster Ordnung können in Hybriden Programmen Bedingungen formuliert werden (dort treten sie mit dem Bezeichner χ auf). Seien ϕ und ψ Formeln der Prädikatenlogik erster Ordnung, θ_i Terme, p ein Prädikat der Stelligkeit n und x eine logische Variable, dann ist die Syntax für Formeln der Prädikatenlogik erster Ordnung wie folgt definiert:

$$\phi, \psi ::= p(\theta_1, ..., \theta_n) \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \rightarrow \psi \mid \phi \leftrightarrow \psi \mid \forall x\, \phi \mid \exists x\, \phi. \quad (4.4)$$

Die Bedeutung der Operatoren ist die gleiche wie die der Operatoren der Logiken d$\mathscr{L}$, DAL und dTL (s. Tabelle 4.2). Die im Quelltext zu verwendende Syntax ist in Abbildung 4.3 angegeben.

Diskrete Sprung-Bedingungen (Discrete jump constraints)

$\mathscr{J}$

DJ-Bedingung

Diskrete Sprung-Bedingungen ($\mathscr{J}$, DJ-Bedingung) werden in DAL eingeführt. Es handelt sich um Formeln der Prädikatenlogik erster Ordnung, die um diskrete Wertzuweisungen $x := \theta$ erweitert werden.

Beispielsweise drückt die Formel $(x := \theta \wedge y > 0) \vee (x := \vartheta \wedge y < 0)$ aus, dass x der Wert von θ zugewiesen wird, wenn $y > 0$ gilt. Gilt $y < 0$ wird x der Wert von ϑ zugewiesen. Gilt $y = 0$ findet keine Wertzuweisung statt und die Programmausführung wird abgebrochen. Wenn sich Fälle überlappen (z.B. $(x := \theta \wedge y > 0) \vee x := \vartheta$), dürfen (wenn in diesem Beispiel $y > 0$ gilt) beide Möglichkeiten gewählt werden.

Differentiell-allgebraische Bedingungen (Differential-algebraic contraints)

$\mathscr{D}$

DA-Bedingung

Differentiell-allgebraische Bedingungen ($\mathscr{D}$, DA-Bedingung) werden in DAL eingeführt. Es handelt sich um Formeln der Prädikatenlogik erster Ordnung, die um Differentialgleichungen ($x' = \theta$) und Differential-Ungleichungen (z.B. $x' \leq \theta$) erweitert werden.

Beispielsweise drückt die Formel $(x' = \theta \wedge x > 0) \vee (x' = -x^2, x < 0)$ aus, dass sich der Zustand des Systems mit $x' = \theta$ verändert, wenn $x > 0$ gilt und mit $x' = -x^2$, wenn $x < 0$ gilt. Die kontinuierliche Zustandsänderung darf solange ausgeführt werden, wie $x > 0$ oder $x < 0$ gilt. Eine Abbruch ist aber auch vorher möglich. Wenn sich Fälle überlappen (z.B. $(x' = \theta \wedge x > 0) \vee x' = -x^2$), dürfen (wenn in diesem Beispiel $x > 0$ gilt) beide Möglichkeiten gewählt werden. Auch ein Wechsel zwischen den Fällen ist jederzeit möglich, wenn die Bedingungen es zulassen.

Durch die Verwendung von Quantoren ist es möglich nichtdeterministisches Verhalten in Differentialgleichungen zu modellieren. Es können z.B. Störungen abgebildet werden, die durch ein unzureichend genaues Modell oder durch Rauschen entstehen. Beispielsweise drückt die Bedingung $\exists u(d_1' = -(\omega + u)d_2 \wedge d_2' = (\omega + u)d_1 \wedge -0.1 \leq u \leq 0.1)$ aus, dass sich die Zustandsvariablen d_1 und d_2 kontinuierlich in Abhängigkeit zu einem beliebigen u aus dem Intervall $[-0.1, 0.1]$ ändern. Während der Ausführung der Differentialgleichungen, kann die Variable u jederzeit einen anderen Wert annehmen [2, S. 135].

Hybride Programme

Seien α und β Hybride Programme, θ_i Terme, x_i Zustandsvariablen, χ eine Formel der Prädikatenlogik erster Ordnung, $\mathscr{J}$ eine DJ-Bedingung und $\mathscr{D}$ eine DA-Bedingung dann ist die Syntax Hybrider Programme wie folgt definiert:

$$\begin{aligned} \alpha, \beta ::= \; & x_1 := \theta_n, ..., x_n := \theta_n \mid x_1' = \theta_1, ..., x_n' = \theta_n \& \chi \mid ?\chi \mid \\ & \alpha \cup \beta \mid \alpha; \beta \mid \alpha^* \mid \mathscr{J} \mid \mathscr{D}. \end{aligned} \tag{4.5}$$

DJ- oder DA-Bedingung können nur in der Logik DAL verwendet werden. Im Folgenden ist die Semantik der einzelnen Operationen erklärt und es sind weitere abgeleitete Operationen aufgeführt. Die Syntax, die im Quelltext zu verwenden ist, ist in Tabelle 4.1 angegeben.

$x_1 := \theta_1, ..., x_n := \theta_n$	**Diskrete Wertzuweisung** Der Wert des Termes θ_i wird der Zustandsvariable x_i zugewiesen; Dies entspricht einem diskreten Sprung im Zustandsautomaten.
$x_1' = \theta_1, ..., x_n' = \theta_n \& \chi$	**Kontinuierliche Entwicklung** Der Zustand ändert sich kontinuierlich wie mit den Differentialgleichungen angegeben. Die Zustandsänderung darf ausgeführt werden, solange die Formel χ wahr ist. Ein vorzeitiger Abbruch ist aber möglich.
$?\chi$	**Zustandsbehauptung** Wenn χ im aktuellen Zustand wahr ist, wird die Ausführung fortgesetzt, anderfalls wird die Ausführung abgebrochen.

$\alpha \cup \beta$	**Nicht deterministische Wahl** Die Ausführung kann mit α oder mit β fortgesetzt werden.
$\alpha; \beta$	**Sequentielle Ausführung** β startet, wenn α beendet ist.
α^*	**Nicht deterministische Wiederholung** Wiederholt α beliebig oft (auch keinmal möglich).
$x := *$	**Zufällige Wertzuweisung** Der Zustands-Variable x wird ein zufälliger Wert (aus der Menge der reellen Zahlen) zugewiesen; Dies entspricht einem diskreten Sprung im Zustandsautomaten. Diese Operation ist äquivalent zu $\exists a(x := a)$ und zu $\exists a(x' := a)$ (s. [2, S. 136]).
$if\ \chi\ then\ \alpha\ else\ \beta\ fi$	**If-then-else-Anweisung** α wird ausgeführt, wenn die Formel χ im aktuellen Zustand wahr ist, andernfalls wird β ausgeführt. Diese Operation ist äquivalent zu $(?\chi; \alpha) \cup (?\neg\chi; \beta)$.
$if\ \chi\ then\ \alpha\ fi$	**If-then-Anweisung** α wird ausgeführt, wenn die Formel χ im aktuellen Zustand wahr ist. Diese Operation ist äquivalent zu $(?\chi; \alpha) \cup (?\neg\chi)$.
$while\ \chi\ do\ \alpha\ end$	**While-Schleife** Wiederholt α solange χ wahr ist. Diese Operation ist äquivalent zu $(?\chi; \alpha)^*; ?\neg\chi$.
$repeat\ \alpha\ until\ \chi$	**Repeat-Schleife** Führt α einmal aus und wiederholt es, wenn χ wahr ist. Diese Operation ist äquivalent zu $\alpha; (?\neg\chi; \alpha)^*; ?\chi$.
$skip$	**Keine Operation** Diese Operation ist äquivalent zu $?true$.
$abort$	**Abbruch** Bricht die Programmausführung ab. Diese Operation ist äquivalent zu $?false$.

Der Abbruch der Ausführung eines Hybrides Programms, z.B. durch eine Zustandsbehauptung $?\chi$, ist nicht, wie in der klassischen Programmierung üblich, als die Be-

$x_1 := \theta_1, ..., x_n := \theta_n$	$x_1 := \theta_1; ...; x_n := \theta_n$
$x_1' = \theta_1, ..., x_n' = \theta_n \& \chi$	$\{x_1' = \theta_1, ..., x_n' = \theta_n, \chi_0, ...\chi_m\}$
$?\chi$	$?\chi$
$\alpha \cup \beta$	α ++ β
$\alpha; \beta$	$\alpha; \beta$
α^*	$(\alpha)*$
$x := *$	$x := *$
if χ *then* α *else* β *fi*	*if* (χ) *then* α *else* β *fi*
if χ *then* α	*if* (χ) *then* α *fi*
while χ *do* α *end*	*while* (χ) *do* α *end*
repeat α *until* χ	wird nicht unterstützt
skip	wird nicht unterstützt
abort	wird nicht unterstützt

Tabelle 4.1: Syntax für Hybride Programme im Quelltext

endigung der Berechnung zu verstehen. Vielmehr kennzeichnet der Abbruch Zustände, die ab diesem Punkt für die Verifikation irrelevant sind. Der Programmcode $a := *; ?(a \geq 5)$ drückt beispielsweise aus, dass die Variable a auf einen beliebigen Wert gesetzt wird und die Ausführung abgebrochen wird, wenn dieser Wert kleiner als 5 ist. Dies bedeutet, dass bei der Verifikation ab dem Punkt $?(a \geq 5)$ alle Zustände, für die $a < 5$ gilt, nicht weiter berücksichtigt werden. Bezogen auf ein reales System bedeutet dies, dass man voraussetzen kann, dass die Variable a unter keinen Umständen auf einen Wert kleiner als 5 gesetzt wird. Bei realen Systemen kann man auch davon ausgehen, dass a deterministisch ein Wert zugewiesen wird. Der Befehl $a := *; ?\chi$ kann somit als Verallgemeinerung eingesetzt werden, um den Determinismus nicht exakt beschreiben zu müssen, sondern mit χ nur soweit wie für die Verifikation erforderlich.

Durch die kontinuierliche Entwicklung wird ebenfalls eine Menge von Zuständen beschrieben, die für die weitere Verifikation von Bedeutung sind. Nach Ausführung des Programmcodes $x := 0; x' = 1 \wedge x < 2$ werden beispielsweise alle Zustände, für die $x \in [0, 2)$ gilt, berücksichtigt.

Notation	Bedeutung
$p(\theta_1, ..., \theta_n)$	wahr genau dann, wenn p für θ_1, ..., θ_n gilt
$\neg\phi$	wahr, wenn ϕ falsch ist
$\phi \wedge \psi$	wahr, wenn ϕ und ψ wahr sind
$\phi \vee \psi$	wahr, wenn ϕ oder ψ wahr sind
$\phi \rightarrow \psi$	wahr, wenn ϕ falsch ist oder ψ wahr
$\phi \leftrightarrow \psi$	wahr, wenn ϕ und ψ beide wahr oder beide falsch sind
$\forall x \ \phi$	wahr, wenn ϕ für alle Werte der logischen Variable x wahr ist
$\exists x \ \phi$	wahr, wenn ϕ für mindestens einen Wert der logischen Variable x wahr ist
$[\alpha]\phi$	wahr, wenn ϕ nach allen möglichen Ausführungen des Hybriden Programmes α wahr ist
$\langle\alpha\rangle\phi$	wahr, wenn ϕ nach mindestens einer möglichen Ausführung des Hybriden Programmes α wahr ist
$[\alpha]\Box\phi$	wahr, wenn ϕ während und nach allen möglichen Ausführungen des Hybriden Programmes α wahr ist
$\langle\alpha\rangle\Diamond\phi$	wahr, wenn ϕ zu einem Zeitpunkt während oder nach mindestens einer möglichen Ausführung des Hybriden Programms α wahr ist
$[\alpha]\Diamond\phi$	wahr, wenn ϕ zu einem Zeitpunkt während oder nach allen möglichen Ausführungen des Hybriden Programms α wahr ist
$\langle\alpha\rangle\Box\phi$	wahr, wenn ϕ während und nach mindestens einer möglichen Ausführung des Hybriden Programms α wahr ist

Tabelle 4.2: Semantik der Operatoren der Logiken d$\mathscr{L}$, DAL und dTL

d$\mathscr{L}$/DAL/dTL-Formeln

Seien ϕ und ψ d$\mathscr{L}$, DAL oder dTL-Formeln, θ_i Terme, p ein Prädikat der Stelligkeit n, x eine logische Variable und α ein Hybrides Programm, dann ist die Syntax von d$\mathscr{L}$, DAL bzw. dTL-Formeln wie folgt definiert:

$$\begin{aligned}\phi, \psi ::= \ & p(\theta_1, ..., \theta_n) \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \rightarrow \psi \mid \phi \leftrightarrow \psi \mid \forall x \ \phi \mid \exists x \ \phi \mid \\ & [\alpha]\phi \mid \langle\alpha\rangle\phi \mid [\alpha]\Box\phi \mid \langle\alpha\rangle\Diamond\phi \mid [\alpha]\Diamond\phi \mid \langle\alpha\rangle\Box\phi.\end{aligned} \tag{4.6}$$

Die Semantik der einzelnen Operationen ist in Tabelle 4.2 angegeben. Aus Tabelle 4.3 ist die Syntax für den Quelltext ersichtlich.

Die $\Box$- und $\Diamond$-Operatoren werden nur in der Logik dTL untersützt. Für die in dieser Arbeit gezeigten Hybriden Systeme mit dem Aufbau $(ctrl; drive)*$ bringen diese Operatoren allerdings keinen weiteren Nutzen. Das Unterprogramm *ctrl* führt nur diskrete Wertzuweisungen aus und in dem Unterprogramm *drive* findet eine kontinuierliche

$\neg$	`!`
$\wedge$	`&`
$\vee$	`|`
$\rightarrow$	`->`
$\leftrightarrow$	`<->`
$\forall x\ \phi$	`(\forall R x; ` ϕ`)`
$\exists x\ \phi$	`(\exists R x; ` ϕ`)`
$[\alpha]\phi$	`\[`α`\]`ϕ
$\langle\alpha\rangle\phi$	`\<`α`\>`ϕ
$[\alpha]\Box\phi$	`\[[`α`\]]`ϕ
$\langle\alpha\rangle\Diamond\phi$	wird nicht unterstützt
$[\alpha]\Diamond\phi$	wird nicht unterstützt
$\langle\alpha\rangle\Box\phi$	wird nicht unterstützt

Tabelle 4.3: Syntax der Operatoren der Logiken d$\mathscr{L}$, DAL und dTL im Quelltext

Entwicklung statt. Die kontinuierliche Entwicklung kann jederzeit abgebrochen werden, so dass jeder Wert, den eine Zustandsvariable in *ctrl* und in *drive* annehmen kann auch der Wert sein kann, den die Zustandsvariable am Ende des Hybriden Programms hat. Damit ist jeder Zustand, der während der Ausführung des Hybriden Programms auftreten kann, auch ein Zustand, den das Hybride Programm am Ende haben kann und der in der Nachbedingung überprüft wird. Dieser Sachverhalt wäre nicht gegeben, wenn das Hybride Programm z.B. den Aufbau $(ctrl; drive; ctrl; drive)*$ hätte [2, S. 209].

Grundgerüst der Quelldatei

KeYmaera akzeptiert Eingabedateien im KeY-Format, das in [23, Anhang B] beschrieben ist. Für gewöhnlich hat eine KeYmaera-Eingabedatei den in Listing 4.1 gezeigten Aufbau. Der Block „\functions{...}“ dient der Deklaration von Funktionen und Konstanten. Im Block „\programVariables{...}“ werden Zustandsvariablen deklariert. Diese können alternativ auch in eckigen Klamern ([...]) im Block „\problem{...}“ deklariert werden. Das Verifikations-Problem wird als d$\mathscr{L}$-, DAL- oder dTL-Formel in den Block „\problem{...}“ eingefügt.

```
\functions{
  /* function symbols */
  R Const_1;
  R Func_1(R);
  R Func_2(R, R);
  /* ... */
}
\programVariables {
  /* state variable declarations (1) */
  R p,v,a;
  R t;
}
\problem{
  /* state variable declaration (2) */
  \[ R var_0, var1, /* ..., */ var_n \] (
    /* a dL/DAL/dTL-Formula */
  )
}
```

Listing 4.1: Grundgerüst der Eingabedatei

4.2.3 Kalkül

Mit Hilfe der Software KeYmaera ist es möglich, die Sequenz $\vdash \phi$ vollständig automatisiert oder mit Benutzerinteraktionen zu beweisen. Mit der Sequenz wird überprüft, ob ϕ unter allen Umständen wahr ist. Häufig hat ϕ den in Formel 4.1 oder 4.2 gezeigten Aufbau. Grundlage für die Suche nach einem Beweis ist ein Kalkül für die Logiken d$\mathscr{L}$, DAL und dTL. Die Regeln dieses Kalküls werden im Folgenden erläutert. Die Semantik des Ausdrucks $\Gamma \vdash \Delta$ wird in [2, S. 76] als $\bigwedge_{\phi \in \Gamma} \phi \rightarrow \bigvee_{\psi \in \Delta} \psi$ definiert. Γ und Δ sind endliche Mengen von Formeln. Statt $true \vdash \Delta$ schreibt man auch verkürzt $\vdash \Delta$ und statt $\Gamma \vdash false$ verkürzt $\Gamma \vdash$.

Ein einfaches Beispiel für einen Beweis (aus [2, S. 87]) ist in Abbildung 4.2 gegeben. Dieser ist von unten nach oben zu lesen. Die Sequenz, die bewiesen werden soll, ist in der Abbildung demnach ganz unten zu finden. Im ersten Schritt wird die Regel $(\rightarrow r)$ angewendet. Die Bedeutung der Regeln folgt weiter unten. Die zweite angewendete Regel $(\wedge r)$ teilt die Sequenz in zwei Zweige auf. Um die Gültigkeit der Ausgangssequenz nachzuweisen, müssen beide Zweige bewiesen werden. Durch Anwenden der Regeln $(\wedge l)$ bzw. $(\vee r)$ und $(\wedge l)$ können beide Zweige in eine Form gebracht werden, bei der auf der linken und rechten Seite des $\vdash$-Symbols die gleichen Terme ($b > 0$ bzw. $v^2 \leq 10$) auftreten. Es handelt sich bei den Sequenzen somit um Axiome und der Beweis kann durch Anwenden der Regel ax auf beiden Zweigen abgeschlossen werden.

$$
\dfrac{\dfrac{\dfrac{}{v^2 \leq 10, b > 0 \vdash b > 0}\,ax}{v^2 \leq 10 \land b > 0 \vdash b > 0}\,\land l \qquad \dfrac{\dfrac{\dfrac{}{v^2 \leq 10, b > 0 \vdash \neg(v \geq 0), v^2 \leq 10}\,ax}{v^2 \leq 10 \land b > 0 \vdash \neg(v \geq 0), v^2 \leq 10}\,\land l}{v^2 \leq 10 \land b > 0 \vdash \neg(v \geq 0) \lor v^2 \leq 10}\,\lor r}{\dfrac{v^2 \leq 10 \land b > 0 \vdash b > 0 \land (\neg(v \geq 0) \lor v^2 \leq 10)}{\vdash v^2 \leq 10 \land b > 0 \to b > 0 \land (\neg(v \geq 0) \lor v^2 \leq 10)}\,\to r}\,\land r
$$

Abbildung 4.2: Einfaches Beweis-Beispiel

Regeln für d$\mathscr{L}$

Der Theorembeweiser KeYmaera verwendet über 300 Regeln, wenn einfache arithmetische Umformungen und Abwandlungen bestimmter Regeln mitgezählt werden. In [2] werden nur die wichtigsten dieser Regeln beschrieben. Eine Übersicht ist in Abbildung 4.3 zu sehen. Im Folgenden werde ich diese Regeln kurz erläutern.

Die Regeln sind zur Verbesserung der Verständlichkeit nur verkürzt dargestellt. Die Regeln der Aussagenlogik, die globalen Regeln und die Regeln $\forall r$-$\forall l$ und $i\forall$ haben den Aufbau

$$\frac{\Phi_1 \vdash \Psi_1 \quad ... \quad \Phi_n \vdash \Psi_n}{\Phi_0 \vdash \Psi_0} \tag{4.7}$$

und können erweitert werden zu

$$\frac{\Gamma, \langle \mathscr{J} \rangle \Phi_1 \vdash \langle \mathscr{J} \rangle \Psi_1, \Delta \quad ... \Gamma, \langle \mathscr{J} \rangle \Phi_n \vdash \langle \mathscr{J} \rangle \Psi_n, \Delta}{\Gamma, \langle \mathscr{J} \rangle \Phi_0 \vdash \langle \mathscr{J} \rangle \Psi_0, \Delta} \quad . \tag{4.8}$$

Die dynamischen Regeln haben einen symmetrischen Aufbau

$$\frac{\phi_1}{\phi_0} \tag{4.9}$$

und können erweitert werden zu

$$\frac{\Gamma \vdash \langle \mathscr{J} \rangle \phi_1, \Delta}{\Gamma \vdash \langle \mathscr{J} \rangle \phi_0, \Delta} \tag{4.10}$$

oder zu

$$\frac{\Gamma, \langle \mathscr{J} \rangle \phi_1 \vdash \Delta}{\Gamma, \langle \mathscr{J} \rangle \phi_0 \vdash \Delta} \quad . \tag{4.11}$$

Γ und Δ sind beliebige Mengen von Formeln (inklusive der leeren Menge) und $\mathscr{J}$ ist eine beliebige Menge diskreter Wertzuweisungen (inklusive der leeren Menge).

Regeln der Aussagenlogik

Mit der weiter oben angegebenen Semantik für das $\vdash$-Symbol sind die Regeln der Aussagenlogik (Propositional Rules) einfach nachzuvollziehen. Die Regeln $(\neg r)$, $(\neg l)$, $(\vee r)$, $(\wedge l)$ und $(\rightarrow r)$ sind unmittelbar klar. Die Regeln $(\vee l)$, $(\wedge r)$ und $(\rightarrow l)$ teilen den Beweis einer Sequenz in zwei Sequenzen auf. Es müssen beide Sequenzen bewiesen werden, damit die ursprüngliche Sequenz bewiesen ist. Die Regel (ax) schließt einen Beweis ab. Dies wird häufig auch zusätzlich durch das Symbol * gekennzeichnet. Die Regel (cut) wird verwendet um Fallunterscheidungen zu machen. Sollen die Fälle $x \geq 0$ und $x < 0$ unterschieden werden, dann wird dies durch $\vdash x \geq 0$ und $x \geq 0 \vdash$ ausgedrücken.

$\neg r, \neg l,$
$\vee r, \wedge l,$
$\rightarrow r, \vee l,$
$\wedge r, \rightarrow l,$
ax, cut

Dynamische Regeln

Die Regel ([?]) bedeutet: Nach allen möglichen Ausführungen des Hybriden Programms $[?\chi]$ muss ψ gelten. Gibt es gar keine Ausführung (also χ ist $false$), so ist der Wahrheitswert von ψ irrelevant. Dieser Sachverhalt lässt sich auch durch $\chi \rightarrow \psi$ ausdrücken. Die Regel ($\langle ? \rangle$) hingegen verlangt, dass es mindestens eine Ausführung gibt, so dass ψ gilt. Daher ist es notwendig, das χ und ψ gelten. [?] $\langle ? \rangle$

Die Regeln ([:=]) und ($\langle := \rangle$) sagen aus, dass eine Menge diskreter Wertzuweisungen durch Substitutionen in ϕ ersetzt werden können, wenn nach allen Ausführungen bzw. mindestens einer Ausführung der diskreten Wertzuweisungen ϕ gilt. [:=], $\langle := \rangle$

Die Regeln ($\langle ' \rangle$) und ([']) sind in d$\mathscr{L}$ die einzigen Regeln, die Differentialgleichungen behandeln. $\langle \mathscr{S}_t \rangle$ steht für die diskreten Wertzuweisungen $\langle x_1 := y_1(t), ..., x_n := y_n(t) \rangle$, wobei $y_1(t), ..., y_n(t)$ die Lösungen der Differentialgleichungen sind. Die Regeln sind einfacher zu verstehen, wenn die Bedingung χ als wahr angenommen wird [2, S. 85]. Sie vereinfachen sich damit zu $\langle ' \rangle$, [']

Regeln der Aussagenlogik

$$(\neg r)\frac{\phi \vdash}{\vdash \neg\phi} \quad (\vee r)\frac{\vdash \phi, \psi}{\vdash \phi \vee \psi} \quad (\wedge r)\frac{\vdash \phi \qquad \vdash \psi}{\vdash \phi \wedge \psi} \quad (\to r)\frac{\phi \vdash \psi}{\vdash \phi \to \psi} \quad (ax)\frac{}{\phi \vdash \phi}$$

$$(\neg l)\frac{\vdash \phi}{\neg\phi \vdash} \quad (\vee l)\frac{\phi \vdash \qquad \psi \vdash}{\phi \vee \psi \vdash} \quad (\wedge l)\frac{\phi, \psi \vdash}{\phi \wedge \psi \vdash} \quad (\to l)\frac{\vdash \phi \quad \psi \vdash}{\phi \to \psi \vdash} \quad (cut)\frac{\vdash \phi \quad \phi \vdash}{\vdash}$$

Dynamische Regeln

$$(\langle ; \rangle)\frac{\langle\alpha\rangle\langle\beta\rangle\phi}{\langle\alpha;\beta\rangle\phi} \quad (\langle^{*n}\rangle)\frac{\phi \vee \langle\alpha\rangle\langle\alpha^*\rangle}{\langle\alpha^*\rangle\phi} \quad (\langle :=\rangle)\frac{\phi^{\theta_1 \ldots \theta_n}_{x_1 \ldots x_n}}{\langle x_1 := \theta_1, .., x_n := \theta_n\rangle\phi}$$

$$([;])\frac{[\alpha][\beta]\phi}{[\alpha;\beta]\phi} \quad ([^{*n}])\frac{\phi \wedge [\alpha][\alpha^*]\phi}{[\alpha^*]\phi} \quad ([:=])\frac{\langle x_1 := \theta_1, .., x_n := \theta_n\rangle\phi}{[x_1 := \theta_1, .., x_n := \theta_n]\phi}$$

$$(\langle\cup\rangle)\frac{\langle\alpha\rangle\phi \vee \langle\beta\rangle\phi}{\langle\alpha \cup \beta\rangle\phi} \quad (\langle ?\rangle)\frac{\chi \wedge \psi}{\langle ?\chi\rangle\psi} \quad (\langle'\rangle)\frac{\exists t \geq 0((\forall\, 0 \leq \tilde{t} \leq t\langle \mathscr{S}_{\tilde{t}}\rangle\chi) \wedge \langle \mathscr{S}_t\rangle\phi)}{\langle x_1' = \theta_1, .., x_n' = \theta_n \& \chi\rangle\phi}$$

$$([\cup])\frac{[\alpha]\phi \wedge [\beta]\phi}{[\alpha \cup \beta]\phi} \quad ([?])\frac{\chi \to \psi}{[?\chi]\psi} \quad ([']) \frac{\forall t \geq 0((\forall\, 0 \leq \tilde{t} \leq t\langle \mathscr{S}_{\tilde{t}}\rangle\chi) \to \langle \mathscr{S}_t\rangle\phi)}{[x_1' = \theta_1, .., x_n' = \theta_n \& \chi]\phi}$$

Quantoren-Regeln der Prädikatenlogik

$$(\forall r)\frac{\vdash \phi(s(X_1, .., X_n))}{\vdash \forall x \phi(x)} \qquad (\exists r)\frac{\vdash \phi(X)}{\vdash \exists x \phi(x)}$$

$$(\exists l)\frac{\phi(s(X_1, .., X_n)) \vdash}{\exists x \phi(x) \vdash} \qquad (\forall l)\frac{\phi(X) \vdash}{\forall x \phi(x) \vdash}$$

$$(i\forall)\frac{\vdash QE(\forall X(\Phi(X) \vdash \Psi(X)))}{\Phi(s(X_1, .., X_n)) \vdash \Psi(s(X_1, .., X_n))} \quad (i\exists)\frac{\vdash QE(\exists X \bigwedge_i(\Phi_i \vdash \Psi_i))}{\Phi_1 \vdash \Psi_1 \quad \ldots \quad \Phi_n \vdash \Psi_n}$$

Globale Regeln

$$([\,]gen)\frac{\vdash \forall^\alpha(\phi \to \psi)}{[\alpha]\phi \vdash [\alpha]\psi} \quad (\langle\rangle gen)\frac{\vdash \forall^\alpha(\phi \to \psi)}{\langle\alpha\rangle\phi \vdash \langle\alpha\rangle\psi}$$

$$(ind)\frac{\vdash \forall^\alpha(\phi \to [\alpha]\phi)}{\phi \vdash [\alpha^*]\phi} \quad (con)\frac{\vdash \forall^\alpha \forall v > 0(\varphi(v) \to \langle\alpha\rangle\varphi(v-1))}{\exists v \varphi(v) \vdash \langle\alpha^*\rangle\exists v \leq 0 \varphi(v)}$$

Abbildung 4.3: Regeln der Differential Dynamic Logic d$\mathscr{L}$[2]

$$\frac{\forall t \geq 0 \langle \mathscr{S}_t \rangle \phi}{[x_1' = \theta_1, ..., x_n' = \theta_n]\phi} \tag{4.12}$$

und

$$\frac{\exists t \geq 0 \langle \mathscr{S}_t \rangle \phi}{\langle x_1' = \theta_1, ..., x_n' = \theta_n \rangle \phi} \quad . \tag{4.13}$$

Mit Hilfe der Regel ($\langle := \rangle$) ist zu erkennen, dass die Differentialgleichungen (Konklusion) gelöst werden ($\mathscr{S}_t$) und für den Zeitpunkt t in ϕ eingesetzt werden. Wenn nach allen Ausführungen der kontinuierlichen Entwicklung ϕ gelten soll, muss die Prämisse für alle $t \geq 0$ gelten. Soll es hingegen mindestens eine Ausführung der kontinuierlichen Entwicklung geben nach der ϕ wahr ist, muss die Prämisse für mindestens ein $t \geq 0$ gelten. Wenn zusätzliche die Bedingung χ zu berücksichtigen ist, muss diese im Zeitraum $[0, t]$ mit den eingesetzten Lösungen der Differentialgleichungen überprüft werden.

Die übrigen Regeln sind einfacher zu verstehen. Die Erklärungen können unter [2] nachgelesen werden.

Quantoren-Regeln der Prädikatenlogik

Quantorenelimination kann nur auf Formeln der Prädikatenlogik erster Ordnung angewendet werden. Dynamische Regeln sind hingegen nur auf Formeln, die keine quantifizierten Variablen enthalten, anwendbar. Mit den Quantoren-Regeln der Prädikatenlogik (First-Order Quantifier Rules) können Quantoren durch das Einführen von Skolem-Funktionen oder freien, logischen Variablen entfernt werden, so dass auf die resultierenden Sequenzen dynamische Regeln angewendet werden können. Wenn mit den dynamischen Regeln alle Modalitäten ($[...]$ und $\langle ... \rangle$) entfernt wurden, können die Skolem-Funktionen und die freien, logischen Variablen mit den Regeln ($i\forall$) und ($i\exists$) durch das erneute Einführen von Quantoren wieder entfernt werden. Die neu eingeführten Quantoren können dann mit Quantorenelimination entfernt werden.

Die Regel ($\exists r$) ersetzt die gebundene, logische Variable x durch die freie, logische Variable X. Der Quantor $\exists$ fällt einfach weg. Die Regel ($\forall l$) funktioniert nach dem gleichen Prinzip. Sie sagt aus, dass nicht für alle x die Bedingung ϕ gilt. Daher genügt es, genau ein X zu finden, für das ϕ nicht gilt. $\exists r, \forall l$

Die Regeln ($\forall r$) und ($\exists l$) ersetzen das x in ϕ durch eine beliebige Funktion s (Skolem- $\forall r, \exists l$

Funktion). Im Gegensatz zu den zwei vorherigen Regeln ist es nicht ausreichend, nur eine freie, logische Variable einzuführen, da mit s zusätzlich die Abhängigkeit zu allen freien, logischen Variablen in ϕ erhalten bleibt. Die Kenntnis dieser Abhängigkeit ist für die Regel ($i\forall$) von Bedeutung.

Die Regel ($i\forall$) führt einen $\forall$-Quantor ein und ersetzt Skolem-Funktionen durch die gebundene, logische Variable X. Es ist notwendig, dass die Skolem-Funktionen von den gleichen freien, logischen Variablen abhängig sind. Die entstehende Formel wird durch Quantorenelimination (s. Kapitel 3.4) vereinfacht. $i\forall$

Die Regel ($i\exists$) führt n offene Zweige des Tableaus zusammen. Die zunächst freie, logische Variable X darf nur in den Sequenzen $\Phi_i \vdash \Psi_i$ auftreten und nicht in der Abhängigkeit einer Skolem-Funktion stehen. Die Regel besagt, dass ein X gesucht wird, das alle Zweige erfüllt. Durch Quantorenelimination wird die Formel vereinfacht (s. Kapitel 3.4). $i\exists$

Globale Regeln

Durch globale Regeln (Global Rules) wird eine Sequenz stark verallgemeinert. Es werden nicht mehr nur die Zustände, die tatsächlich durch das Hybride Programm α erreicht werden können, sondern alle Zustände (d.h. alle möglichen Belegungen der gebundenen Zustandsvarialben in α) in den Beweis einbezogen. Eine zu starke Verallgemeinerung kann dazu führen, dass kein Beweis mehr gefunden werden kann.

$\forall^\alpha \phi$ bedeutet, dass alle in dem Hybriden Programm α gebundenen Zustandsvariablen mit $\forall$ quantifiziert werden. Eine Zustandsvariable x ist gebunden, wenn ihr Wert in ϕ geändert wird, entweder mit einer diskreten Wertzuweisung ($x := \theta$) oder mit einer Differentialgleichung ($x' = \theta$). In Gleichung 4.14 ist dieser Zusammenhang beispielhaft dargestellt.

$$\begin{aligned} \alpha &\equiv (a := \theta; t := 0; x' = v, v' = a, t \leq \epsilon)^* \\ \forall^\alpha \phi &\equiv \forall a, t, x, v\ \phi \end{aligned} \tag{4.14}$$

Die Regel ($[]gen$) drückt aus, dass alle durch α erreichbaren Zustände ψ erfüllen, wenn alle durch α erreichbaren Zustände ϕ erfüllen ($[\alpha]\phi$) und ϕ die Bedingung ψ in allen möglichen Zuständen impliziert ($\forall^\alpha(\phi \rightarrow \psi)$). Gleiches gilt, wenn man wie in der Regel ($\langle\rangle gen$) nur einen durch α erreichbaren Zustand betrachtet. $[]gen, \langle\rangle gen$

Die Regel (ind) besagt, dass ϕ nach keiner oder mehrfacher Ausführung des Hybride Programm α gilt, wenn ϕ initial und nach exakt einer Ausführung des Hybriden Programmes α bei allen möglichen initialen Zuständen gilt. ind

Die Regel (*con*) besagt, dass nachdem α genügend oft ausgeführt wurde die Variante $\varphi(v)$ mit $v \leq 0$ gilt, wenn $\varphi(v)$ am Anfang galt (Antezedent) und $\varphi(v)$ durch jede Ausführung von α um 1 dekrementiert wird (Prämisse). *con*

Von (*ind*) und (*con*) abgeleitete Regeln eignen sich in der Praxis um Sequenzen mit Schleifen zu beherrschen (s. Gleichung 4.15 und 4.16). Einer Herleitung dieser Regeln ist in [2, S. 86] zu finden. Die Schwierigkeit bei der Anwendung dieser Regeln ist das finden einer geeigneten Invarianten ϕ bzw. Varianten $\varphi(v)$.

$$(\text{ind}')\frac{\vdash \phi \qquad \vdash \forall^{\alpha}(\phi \rightarrow [\alpha]\phi) \qquad \vdash \forall^{\alpha}(\phi \rightarrow \psi)}{\vdash [\alpha^*]\psi} \tag{4.15}$$

$$(\text{con}')\frac{\vdash \exists v \varphi(v) \qquad \vdash \forall^{\alpha}\forall v > 0(\varphi(v) \rightarrow \langle\alpha\rangle\varphi(v-1)) \qquad \vdash \forall^{\alpha}(\exists v \leq 0 \varphi(v) \rightarrow \psi)}{\vdash \langle\alpha^*\rangle\psi} \tag{4.16}$$

Regeln für DAL

In Abbildung 4.4 sind die Regeln abgebildet, die in der Logik DAL zusätzlich zu den Regeln der Logik d$\mathscr{L}$ zur Verfügung stehen.

Quantoren-Regeln der Prädikatenlogik

Reelle Quantorenelimination kann nicht auf Formeln, die Modalitäten enthalten, angewendet werden [2, S. 168]. Beispielsweise muss in der Formel $\exists x[x' = -x; x := 2x]x \leq 5$ berücksichtigt werden, wie sich x zwischen $\exists x$ und $x \leq 5$ verändert. Um dieses Problem zu lösen verwenden die Quantoren-Regeln für DAL „side deduction“ (s. Abbildung 4.5). Anstatt die Suche nach einem Beweis mit $\Gamma \vdash \Delta, \exists x \phi$ fortzusetzen, wird die Sequenz $\Gamma \vdash \Delta, \phi$, in der x nicht quantifiziert ist, verwendet. Ohne den Quantor können nun dynamische Regeln angewendet werden. Sobald x in den Sequenzen $\Gamma_1 \vdash \Delta_1$, ..., $\Gamma_n \vdash \Delta_n$ nur noch in Formeln der Prädikatenlogik erster Ordnung auftritt, wird es in der Konjunktion der Sequenzen wieder mit $\exists$ quantifiziert und Quantorenelimination durchgeführt. $r\forall$, $r\exists$, $l\forall$, $l\exists$

Die Regeln $(r\forall)$, $(r\exists)$, $(l\forall)$ und $(l\exists)$ lassen sich aus den Quantoren-Regeln der Prädikatenlogik für d$\mathscr{L}$ ableiten.

Quantoren-Regeln der Prädikatenlogik

$$(r\forall)\frac{QE(\forall x \bigwedge_i(\Gamma_i \vdash \Delta_i))}{\Gamma \vdash \Delta, \forall x\phi} \quad (r\exists)\frac{QE(\exists x \bigwedge_i(\Gamma_i \vdash \Delta_i))}{\Gamma \vdash \Delta, \exists x\phi}$$

$$(l\forall)\frac{QE(\exists x \bigwedge_i(\Gamma_i \vdash \Delta_i))}{\Gamma, \forall x\phi \vdash \Delta} \quad (l\exists)\frac{QE(\forall x \bigwedge_i(\Gamma_i \vdash \Delta_i))}{\Gamma, \exists x\phi \vdash \Delta}$$

Dynamische Regeln

$$(\langle\exists\rangle)\frac{\exists x\langle\mathscr{J}\rangle\phi}{\langle\exists x\,\mathscr{J}\rangle\phi} \qquad (\langle:=\rangle)\frac{\chi \wedge \phi^{\theta_1 \ldots \theta_n}_{x_1 \ldots x_n}}{\langle x_1 := \theta_1 \wedge .. \wedge x_n := \theta_n \wedge \chi\rangle\phi}$$

$$([\exists])\frac{\forall x[\mathscr{J}]\phi}{[\exists x\,\mathscr{J}]\phi} \qquad ([:=])\frac{\chi \rightarrow \phi^{\theta_1 \ldots \theta_n}_{x_1 \ldots x_n}}{[x_1 := \theta_1 \wedge .. \wedge x_n := \theta_n \wedge \chi]\phi}$$

$$(\langle J\rangle)\frac{\langle\mathscr{J}_1 \cup \ldots \cup \mathscr{J}_n\rangle\phi}{\langle\mathscr{J}\rangle\phi} \quad (\langle D\rangle)\frac{\langle(\mathscr{D}_1 \cup \ldots \cup \mathscr{D}_n)*\rangle\phi}{\langle\mathscr{D}\rangle\phi}$$

$$([J])\frac{[\mathscr{J}_1 \cup \ldots \cup \mathscr{J}_n]\phi}{[\mathscr{J}]\phi} \quad ([D])\frac{[(\mathscr{D}_1 \cup \ldots \cup \mathscr{D}_n)*]\phi}{[\mathscr{D}]\phi}$$

$$(\langle DR\rangle)\frac{\vdash \langle\mathscr{D}\rangle\phi}{\vdash \langle\mathscr{E}\rangle\phi} \qquad (DS)\frac{\vdash [\mathscr{D}]\chi \quad \vdash [\mathscr{D} \wedge \chi]\phi}{\vdash [\mathscr{D}]\phi}$$

$$([DR])\frac{\vdash [\mathscr{E}]\phi}{\vdash [\mathscr{D}]\phi}$$

Globale Regeln

$$(DI)\frac{\vdash \forall^\alpha \forall y_1 \ldots \forall y_k(\chi \rightarrow F'^{\theta_1 \ldots \theta_n}_{x'_1 \ldots x'_n})}{[\exists y_1 \ldots \exists y_k \chi]F \vdash [\exists y_1 \ldots \exists y_k(x'_1 = \theta_1 \wedge \ldots \wedge x'_n = \theta_n \wedge \chi)]F}$$

$$(DV)\frac{\vdash \exists\epsilon > 0 \forall^\alpha \forall y_1 \ldots y_k(\neg F \wedge \chi \rightarrow (F' \geq \epsilon)^{\theta_1 \ldots \theta_n}_{x'_1 \ldots x'_n})}{[\exists y_1 \ldots y_k(x'_1 = \theta_1 \wedge \ldots \wedge x'_n = \theta_n \wedge \sim F)]\chi \vdash \langle\exists y_1 \ldots y_k(x'_1 = \theta_1 \wedge \ldots \wedge x'_n = \theta_n \wedge \chi)\rangle F}$$

Abbildung 4.4: Regeln der Differential-Algebraic Dynamic Logic DAL [2]

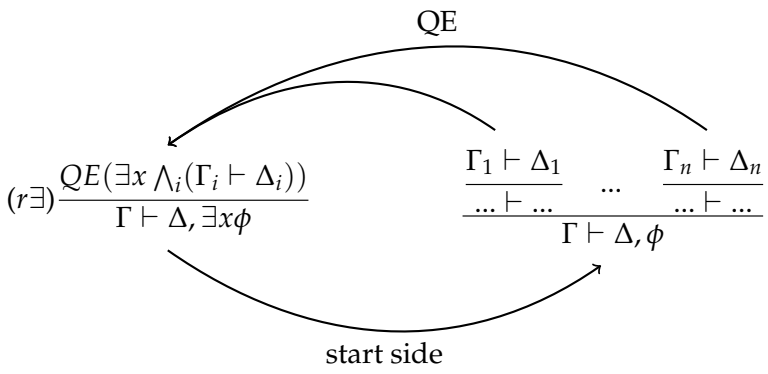

Abbildung 4.5: Side deduction [2]

Dynamische Regeln

$\mathscr{D}$ und $\mathscr{E}$ sind DA-Bedingungen für die gilt $\mathscr{D} \rightarrow \mathscr{E}$. Die Regeln ($[DR]$) kann verwendet werden, um DA-Bedingungen zu verallgemeinern. Beispielsweise kann $[x' = 5x \wedge y' = 1 - x \wedge x > 0]$ zu $[x' = 5x \wedge y' \leq 1 \wedge x > 0]$ verallgemeinert werden [2, S. 166]. Die Regel ($\langle DR \rangle$) kann verwendet werden, um nichtdeterministisches Verhalten zu entfernen. Beispielsweise kann $\langle z' = v \wedge v' \geq -b \rangle \phi$ zu dem speziellen Fall $\langle z' = v \wedge v' = -b \rangle \phi$ umgeformt werden [2, S. 166]. $[DR]$ $\langle DR \rangle$

Die Erklärungen der übrigen Regeln ist in [2] zu finden.

Globale Regeln

Die globalen Regeln der Logik DAL können mit Differentialgleichungen umgehen, ohne diese lösen zu müssen. Die Regeln sind einfacher zu verstehen, wenn die existenziell quantifizierten Variablen y_1, ..., y_k und die Bedingung χ zunächst nicht beachtet werden.

In der Regel (DI) ist F eine differentielle Invariante. F muss eine Formel der Prädikatenlogik erster Ordnung sein, die keine negativen Gleichungen ($\neq$) enthält [2, S. 164]. Anstatt die Differentialgleichungen $x_1' = \theta_1$, ..., $x_n' = \theta_n$ zu lösen, wird in der Prämisse die totale Ableitung F' berechnet [2, Definition 3.13] und alle Vorkommnisse von x_1', ..., x_n' werden durch θ_1, ..., θ_n substituiert. Die Regel sagt aus, dass F auch nach der kontinuierlichen Entwicklung $x_1' = \theta_1$, ..., $x_n' = \theta_n$ (Sukzedent der Konklusion) gilt, wenn F initial gilt (Antezedent der Konklusion) und die totale Ableitung F' für alle x_1, ..., x_n ($\forall^\alpha$) gilt (Prämisse). DI

Für die Regel (DV) darf F keine Gleichungen enthalten (d.h. nur Ungleichungen DV

sind zugelassen) und die Differentialgleichungen müssen Lipschitz-stetig sein. Der Ausdruck $F' \geq \epsilon$ bedeutet, dass alle Ungleichungen der Form $a \geq b$ in F' durch $a \geq b + \epsilon$ ersetzt werden [2, S. 181]. Gleiches gilt für die Operatoren $\leq$, $>$ und $<$. Der Operator $\sim$ ist ähnlich der Negation $\neg$. Dabei werden Ausdrücke der Form $a \leq b$ in $a \geq b$ umgeformt, Ausdrücke der Form $a < b$ allerdings in $a \geq b$. Gleiches gilt für die Operatoren $\geq$ und $>$. Die Regel drückt aus, dass es eine kontinuierliche Entwicklung $x_1' = \theta_1$, ..., $x_n' = \theta_n$ gibt, so dass F gilt (Sukzedent der Konklusion), wenn F initial noch nicht gilt (Antezedent der Konklusion) und es ein positives ϵ gibt, mit dem der Ausdruck $F' \geq \epsilon$ mit den eingesetzten Differentialgleichungen für alle x_1, ..., x_n ($\forall^\alpha$) wahr ist (Prämisse).

Mit χ wird in den globalen Regeln die Ausbreitung der Differentialgleichung eingeschränkt (evolution domain). y_1, ..., y_k sind in der DA-Bedingung quantifizierte Variablen. Diese müssen, wenn sie im Sukzedenten der Konklusion auftreten, auch in dem Antezedenten auftreten und in der Prämisse berücksichtigt werden.

4.2.4 Relative Vollständigkeit

Der Begriff der relativen Vollständigkeit wurde in Abschnitt 3.3 eingeführt.

Auch die Logiken d$\mathscr{L}$, DAL und dTL sind nur relativ vollständig. In [2] wird gezeigt, dass d$\mathscr{L}$ und dTL relativ zu FOD vollständig ist, sowie DAL zu DA-Bedingungen. FOD steht für „first order logic of differential equations“ und wird in [2] eingeführt.

Der Nachweis der relativen Vollständigkeit zeigt, dass die Herausforderung bei der Verifikation Hybrider Systeme das Finden von Invarianten bzw. differentiellen Invarianten ist [2, S. 118]. Alle anderen Dinge werden bereits vom Kalkül berücksichtigt.

4.2.5 Suchstrategien

Der Kalkül ist die Grundlage für die automatisierte Suche nach einem Beweis für eine Formel. Der in KeYmaera implementierte Algorithmus ist in Abbildung 4.2 zu sehen [2, S. 237]. Bei gegebenem Tableau, das initial die zu beweisende Formel enthält, sind in einer Schleife solange Regeln anzuwenden, bis ein Beweis gefunden ist. In jedem Schleifendurchlauf sind 3 Entscheidungen zu treffen:

1. Wahl eines offenen Zweiges B des Tableaus T (Zeile 2),
2. Wahl einer Regel M, die auf den zuvor gewählten Zweig B anwendbar ist (Zeile 3) und
3. Wahl einer Menge von Formeln F des gewählten Zweiges B in Abhängigkeit von der gewählten Regel M (Zeile 4).

```
while tableau T has open branches do
  B := selectBranch(T)
  M := selectMode(B)
  F := selectFormulas(B, M)
  if M = foreground then
    R :=  result of applying a propositional or
          dynamic or global rule to F in B
    replace branch B by R in tableau T
  else
    send key F to background decision procedure QE
    receive result R from QE
    apply a quantifier rule to T with QE–result R
  end if
end while
```

Listing 4.2: Tableau-Prozedur für d$\mathscr{L}$

Die Wahl eines Zweiges hat geringen Einfluss auf die Performance der Suche [2, S. 238], da jeder Zweig bewiesen werden muss. Lediglich bei der Regel ($i\exists$), die mehrere Zweige zusammenführt, besteht die Notwendigkeit, dass die Regel auf alle Zweige anwendbar ist. Dazu müssen Zweige, in denen die existenziell quantifizierte Variable bereits nur noch in Prädikatenlogik erster Ordnung auftritt, zurückgestellt werden, bis diese Eigenschaft bei allen Zweigen vorliegt.

Die Wahl der Formeln hat einen großen Einfluss auf die Rechenzeit. Der Grund dafür ist im Wesentlichen die von Mathematica ausgeführte Quantorenelimination [2, S. 239]. Sei Φ eine Menge von Formeln, die eine Tautologie bilden, dann ist jede Obermenge $\Psi \supseteq \Phi$ ebenfalls eine Tautologie. Mathematica benötigt für die Quantorenelimination (mit der Funktion Reduce) der Formel in Abbildung 4.6, die eine überflüssige Bedingung enthält, mehr als 24 h. Der Reduce-Aufruf ist in Abbildung 4.2.5 dargestellt. Nach Entfernung der überflüssigen Vorbedingung $2b(m-z) \geq v^2$ ist die Berechnung in weniger als 1 s abgeschlossen [2]. Überflüssige Vorbedingungen entstehen, wenn eine Sequenz durch eine Regel in mehrere Zweige aufgeteilt wird. Hierbei werden alle Vorbedingungen an jeden Zweig weitergegeben, auch wenn sie nur bei einigen benötigt werden. Durch einen manuellen Eingriff kann dieses Problem gelöst werden, indem in KeYmaera mit dem Befehl „hide_left“ die überflüssigen Formeln ausgeblendet wird. Dazu müssen die überflüssigen Formeln natürlich erkannt werden.

Die Wahl einer Regel hat geringeren Einfluss auf die Performance als die Wahl von Formeln [2]. In KeYmaera ist es möglich zwischen den Strategien *lazy*, *eager* und *IBC* zu wählen. *lazy* bedeutet, dass Mathematica erst aufgerufen wird, wenn keine andere Regel mehr angewendet werden kann. *eager* bedeutet, dass Mathematica aufgerufen

$$
\begin{aligned}
& t_2 > 0,\ \epsilon \geq t_2,\ v_2 \geq 0,\ A + 1/t_2 v_2 \geq 0,\ t_2 \geq 0, \\
& m - z_2 \geq v_2^2/(2b) + (A/b + 1)(A/2\epsilon^2 + \epsilon v_2), \\
& 2b(m - z_2) \geq v_2^2, \\
& 2b(m - z) \geq v^2, \quad \text{/* überflüssige Bedingung */} \\
& b > 0,\ A \geq 0 \\
\vdash\ & (At_2 + v_2)^2 \leq 2b(m - 1/2(At_2^2 + 2t_2 v_2 + 2z_2))
\end{aligned}
$$

Abbildung 4.6: Formel für Quantorenelimination mit überflüssiger Vorbedingung

```
Reduce[Implies[
 t2 > 0 && eps >= t2 && v2 >= 0 && A + 1/t2 v2 >= 0 && t2 >= 0 &&
 m - z2 >= v2^2/(2 b) + (A/b + 1) (A/2 eps^2 + eps v2) &&
 2 b (m - z2) >= v2^2 &&
 2 b (m - z) >= v^2 &&
 b > 0 && A >= 0, (A t2 + v2)^2 <=
 2 b (m - 1/2 (A t2^2 + 2 t2 v2 + 2 z2))], {}, Reals]
```

Abbildung 4.7: Verwendung des Reduce-Befehls in Mathematica

wird, sobald Formeln in Prädikatenlogik erster Ordnung vorliegen. *IBC* beschreibt einen Mittelweg. Aufrufe von Mathematica erfolgen mit einem Timeout, der während der Suche erhöht wird.

4.2.6 Auffälligkeiten

Die Software KeYmaera enthält einige kleinere Bugs, mit denen man umgehen kann. Wenn sich das Programm ungewöhnlich verhält, kann ein Neustart der Software helfen. Es ist dabei darauf zu achten, dass der Mathematica-Kernel („MathKernel.exe“) ebenfalls beendet wird. Sollte eine Berechnung mit Mathematica nicht klappen, wird in KeYmaera nicht angezeigt, warum dies so ist. Es kann somit sein, dass für eine einfache Formel kein Beweis gefunden wird, weil der Mathematica-Kernel mit nicht mehr benötigten Berechnungen ausgelastet ist.

Es kann auch passieren, dass der Theorembeweiser einen Beweis für eine falsche Aussage findet. Dies ist beispielsweise in dem Verifikations-Problem in Listing 4.3 der Fall. Es überprüft, ob durch die Vorbedingung $I = 1 \wedge v = 0$ die Nachbedingung $v = 1$ impliziert wird. Die Quantorenelimination von Mathematica berechnet fälschlicher Weise, dass diese Aussage wahr ist. Das Problem entsteht, da Mathematica den Bezeichner I als imaginäre Einheit verwendet, und dieser durch die Formel überschrieben wird. Entfernt man die Vorbedingung $I = 1$, so entsteht dieses Problem

nicht. Seitens KeYmaera erfolgt keine Warnung. Weitere von Mathematica verwendete Bezeichner sind unter [17] zu finden.

```
\problem{
  \[ R v, I \] (
      I=1 & v=0
    ->
      v=1
  )
}
```

Listing 4.3: Verifikations-Problem, das zu einem falschen Beweis führt

Ein weiteres Problem ist, das durch das Lösen von Differentialgleichungen Fehler der Gleitkommaarithmetik auftreten können. Löst man mit KeYmaera die Differentialgleichung des in Listing 4.4 gezeigten Programms, wird Gleichung 4.17 als Lösung zurückgegeben. Dieses Problem tritt nicht auf, wenn $0,1$ als $1/10$ angegeben wird.

```
\problem{
    \[ R x\] (
        \[ { x' = 0.1 } \] true
    )
}
```

Listing 4.4: Verifikations-Problem, das zu einem falschen Beweis führt

$$x = 0,100000000000000005...t + x \tag{4.17}$$

5 Kollisionsvermeidung

Dieses Kapitel enthält Anwendungsfälle, die zeigen, wie die Logiken d$\mathscr{L}$ und DAL für die Verifikation einfacher Hybrider Systeme eingesetzt werden können. Als Beispiele dienen ein Notbremsassistent und ein intelligenter Tempomat für Kraftfahrzeuge. Am Ende wird ein Ansatz gezeigt, mit dem Modelle für Quadrocopter erstellt werden können.

5.1 Notbremsassistent

Der in diesem Abschnitt betrachtete Notbremsassistent ist eine Vorrichtung für Fahrzeuge, die beim Erreichen eines kritischen Abstandes zu einem Hindernis eine automatische Notbremsung einleiten. Das Modell entspricht dem in [2] gezeigten Hybriden System zur Kollisionsvermeidung von Zügen im European Train Control System (ETCS).

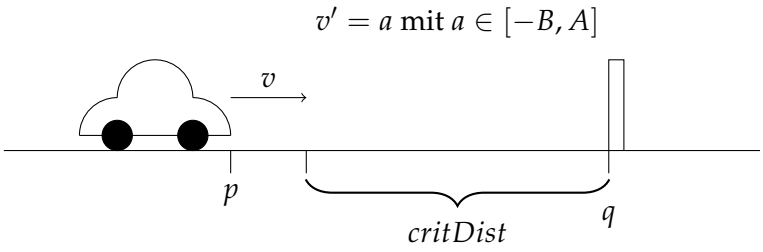

Abbildung 5.1: Notbremsassistent

$$
\begin{aligned}
NBA \equiv\ & (ctrl; drive)^* \\
ctrl \equiv\ & if\ (q - p > critDist)\ then \\
& \quad a := *;\ ?(-B \leq a \leq A) \\
& else \\
& \quad a := -B \\
& fi \\
drive \equiv\ & t := 0; \{p' = v, v' = a, t' = 1, t \leq \epsilon, v \geq 0\}
\end{aligned}
$$

Abbildung 5.2: Hybrides Programm für einen Notbremsassistenen

5.1.1 Modell

Um das Modell einfach zu halten, wird angenommen, dass sich das Fahrzeug im 1-dimensionalen Raum bewegt. Aus Abbildung 5.1 wird die Bedeutung der im Modell verwendeten Variablen ersichtlich. Für Position, Geschwindigkeit und Beschleunigung des Fahrzeuges werden die Variablen p, v und a verwendet. Die Bewegung wird mit den Differentialgleichungen

$$p' = v \tag{5.1}$$

$$v' = a \tag{5.2}$$

modelliert. B ist die maximale Bremskraft und A die maximale Beschleunigung des Fahrzeuges. Die Position des Hindernisses ist mit q angegeben. Es wird davon ausgegangen, dass initial $p \leq q$ gilt. Wenn der Abstand des Fahrzeuges zum Hindernis kleiner oder gleich *critDist* ist, dann soll der Notbremsassistent eine Vollbremsung einleiten. *critDist* muss so gewählt werden, dass eine Kollision des Fahrzeuges mit dem Hindernis ausgeschlossen ist. Das Hybride Programm des Notbremsassistenten ist in Abbildung 5.2 zu sehen. Das Hauptprogramm NBA gibt an, dass die Unterprogramme *ctrl* und *drive* sequentiell und beliebig oft hintereinander ausgeführt werden. In *ctrl* wird der Abstand des Fahrzeuges zum Hindernis überprüft. Ist der Abstand größer als *critDist*, wird die Beschleunigung a auf einen beliebigen Wert aus dem Interval $[-B, A]$ gesetzt. Anderfalls wird a auf $-B$ gesetzt, so dass das Fahrzeug eine Vollbremsung macht. Das Unterprogramm *drive* beschreibt mit den Differentialgleichungen $p' = v$ und $v' = a$ die kontinuierliche Zustandsänderung des Fahrzeuges. Die Variable ϵ steht für die maximale Reaktionszeit. Mit Hilfe der Variablen t wird die kontinuierliche Ausführung der Differentialgleichungen nach spätestens ϵ Zeiteinheiten abgebrochen und der Notbremsassistent kann der Variablen a einen neuen Wert zuweisen.

5.1.2 Herleitung des kritischen Abstandes

Der kritische Abstand *critDist* zwischen Fahrzeug und Hindernis zum sicheren Bremsen ist die Summe aus Reaktionsweg s_R und Bremsweg s_B (Gleichung 5.3).

$$critDist = s_R + s_B \tag{5.3}$$

Der Reaktionsweg s_R kann durch zweifaches Integrieren der vom Controller gewählten Beschleunigung $\tilde{a}$ ermittelt werden (s. Gleichung 5.4 bis 5.6). Er ist abhängig von der aktuellen Geschwindigkeit v_0, der Beschleunigung $\tilde{a}$ und der Reaktionszeit τ des Controllers.

$$s_R''(\tilde{a}) = \tilde{a} \tag{5.4}$$

$$s_R'(v_0, \tilde{a}, \tau) = v_0 + \tilde{a}\tau \tag{5.5}$$

$$s_R(v_0, \tilde{a}, \tau) = v_0\tau + 0.5\tilde{a}\tau^2 \tag{5.6}$$

Da in diesem Modell zum Zeitpunkt der Berechnung des kritischen Abstandes nur v_0 bekannt ist (die Beschleunigung $\tilde{a}$ wird erst danach auf einen zufälligen Wert gesetzt und die Reaktionszeit liegt zwar im Intervall $[0, \epsilon]$ ist aber nicht exakt bekannt) werden $\tilde{a}$ und τ so gewählt, dass $s_R(v_0, \tilde{a}, \tau)$ maximal wird. Dies ist der Fall, wenn für $\tilde{a}$ und τ die jeweils maximalen Werte A und ϵ eingesetzt werden. Die Gleichung für den Reaktionsweg vereinfacht sich damit zu Gleichung 5.7

$$s_R(v_0) = v_0\epsilon + 0.5A\epsilon^2 \tag{5.7}$$

Der Bremsweg s_B ist abhängig von der nach der Reaktionszeit erreichten Geschwindigkeit $\tilde{v}$ (s. Gleichung 5.8), der Bremsdauer d_B und die bei der Vollbremsung ausgeübten Bremskraft $-B$ (s. Gleichung 5.9). s_B kann durch zweifaches Integrieren von $-B$ ermittelt werden.

$$\tilde{v}(v) = s_R'(v_0, A, \epsilon) = v + A\epsilon \tag{5.8}$$

$$s_B(\tilde{v}, d_B) = \tilde{v}d_B - 0.5Bd_B^2 \tag{5.9}$$

Die Bremsdauer d_B ist die Zeit, die das Fahrzeug benötigt, um von der Geschwindigkeit $\tilde{v}$ zum Stillstand zu kommen. Sie kann ermittelt werden, indem die erste Ableitung der Gleichung 5.9 gleich 0 gesetzt wird (s. Gleichung 5.10 und 5.11).

$$s'_B(\tilde{v}, d_B) = \tilde{v} - Bd_B \stackrel{!}{=} 0 \tag{5.10}$$

$$d_B = \frac{\tilde{v}}{B} \tag{5.11}$$

Die Gleichungen 5.8 und 5.11 in Gleichung 5.9 eingesetzt führen zu Gleichung 5.12.

$$s_B(v_0) = \frac{(v_0 + A\epsilon)^2}{2B} \tag{5.12}$$

Die Gleichungen 5.12 und 5.6 in 5.3 eingesetzt führen zu der gesuchten Gleichung 5.13.

$$critDist(v) = v_0\epsilon + 0.5A\epsilon^2 + \frac{(v_0 + A\epsilon)^2}{2B} \tag{5.13}$$

5.1.3 Verifikation

Eine Kollision des Fahrzeuges mit dem Hindernis hat stattgefunden, wenn $p > q$ gilt. Wenn nach allen möglichen Ausführungen des Hybriden Programms NBA $p \leq q$ gilt, ist eine Kollision somit ausgeschlossen. Zur Verifikation der Kollisionsfreiheit ist die Formel

$$\phi \rightarrow [NBA]p \leq q \tag{5.14}$$

auszuwerten. In Gleichung 5.15 ist die zugehörige Vorbedingung ϕ angegeben. Es wird gefordert, dass die maximale Bremskraft B positiv, die maximale Beschleunigung A nicht negativ, die maximale Reaktionszeit ϵ positiv und die Position des Fahrzeuges p kleinergleich der Position des Hindernisses q ist. Zudem ist die initial erlaubte Geschwindigkeit abhängig von dem initialen Abstand des Fahrzeuges zum Hindernis und der maximalen Bremskraft. Diese Bedingung ($v^2 \leq 2B(q-p)$) lässt sich automatisiert mit KeYmaera ermitteln (s. [2][S. 90]). Sie geht auch aus Gleichung 5.13 hervor, wenn ϵ auf 0 gesetzt wird.

$$\phi \equiv B > 0 \wedge A \geq 0 \wedge \epsilon > 0 \wedge p \leq q \wedge v^2 \leq 2B(q-p) \tag{5.15}$$

Mit Hilfe des Theorembeweisers KeYmaera kann die Formel 5.14 vollständig automatisiert beweisen werden. Der Beweis dauert ca. 10 Sekunden. Der Quelltext zur

Verifikation der Formel 5.14 mit KeYmaera ist im Anhang A.1 zu finden.

5.1.4 Bemerkungen

Das zuvor erstellte und verifizierte Modell besitzt einige Freiheiten für die Implementierung eines Notbremsassistenten. Offen bleibt, wie die Beschleunigung gesetzt wird, wenn der Abstand zum Hindernis noch ausreichend groß ist. Damit eine Kollision ausgeschlossen ist, wird lediglich gefordert, dass sie im Bereich $[-B, A]$ liegt.

Das folgende Beispiel zeigt allerdings, dass das Modell ggf. unerwünschte Eigenschaften hat. Diese treten in Erscheinung, wenn versucht wird, das Fahrzeug mit einer konstanten Bremskraft exakt vor einem Hindernis zum Stehen zu bringen. Mögliche Anwendungen wären das Halten vor einer roten Ampel oder das Heranfahren an eine Laderampe. In Abbildung 5.3 ist erkennbar, wie sich ein Fahrzeug verhält, das sich 35 m vor einen Hindernis befindet, sich mit einer Geschwindigkeit von $9m/s$ ($\widehat{=}32,4km/h$) auf dieses zubewegt und mit einer Beschleunigung von $-1,16m/s^2$ abbremst. Ohne Eingriff des Notbremsassistenten würde das Fahrzeug exakt vor dem Hindernis zum stehen kommen (gepunktete Linie). Mit dem Notbremsassistenten (mit der Konfiguration $A = 4m/s^2$, $B = 4m/s^2$ und $\epsilon = 1s$) würde das Fahrzeug nach 1 Sekunde eine Vollbremsung machen. Der Grund dafür ist, dass bei der Berechnung des kritischen Abstandes (Gleichung 5.13) angenommen wird, dass das Fahrzeug mit A beschleunigt. Die Abschätzung der Beschleunigung ist damit zu restriktiv. Wählt man nach 2 Sekunden wieder eine Beschleunigung, die das Fahrzeug exakt vor dem Hindernis zum Stehen bringt, greift der Notbremsassistent nach 6 Sekunden erneut ein und bringt das Fahrzeug ca. 7 m vor dem Hindernis zum Stehen.

Dieses Problem tritt insbesondere dann auf, wenn ϵ oder A zu groß gewählt werden.

5.1.5 Verbesserungen

Um dem in Abschnitt 5.1.4 genannten Problem entgegenzuwirken führe ich hier eine Möglichkeit auf, wie die Beschleunigung vom Controller „besser“ ausgewählt werden kann.

Die Beschleunigung des Fahrzeuges a wird auf den Bereich $[-B, Min(\tilde{A}, A)]$ beschränkt. $\tilde{A}$ ist abhängig von der aktuellen Position und Geschwindigkeit des Fahrzeuges und wird so gewählt, dass das Fahrzeug exakt vor dem Hindernis zum Stehen kommt, wenn es für ϵ Zeiteinheiten mit $\tilde{A}$ und danach bis zum Stillstand mit $-B$ beschleunigt wird. Mit dieser dynamisch angepassten Beschränkung ist es nicht mehr notwendig den kritischen Abstand zu berechnen und die Entscheidung zu treffen ob eine Vollbremsung einzuleiten ist, da die Wahl von a bereits für die geforderte Sicherheit sorgt. Die Berechnung von $\tilde{A}$ ist in Gleichung 5.16 gegeben.

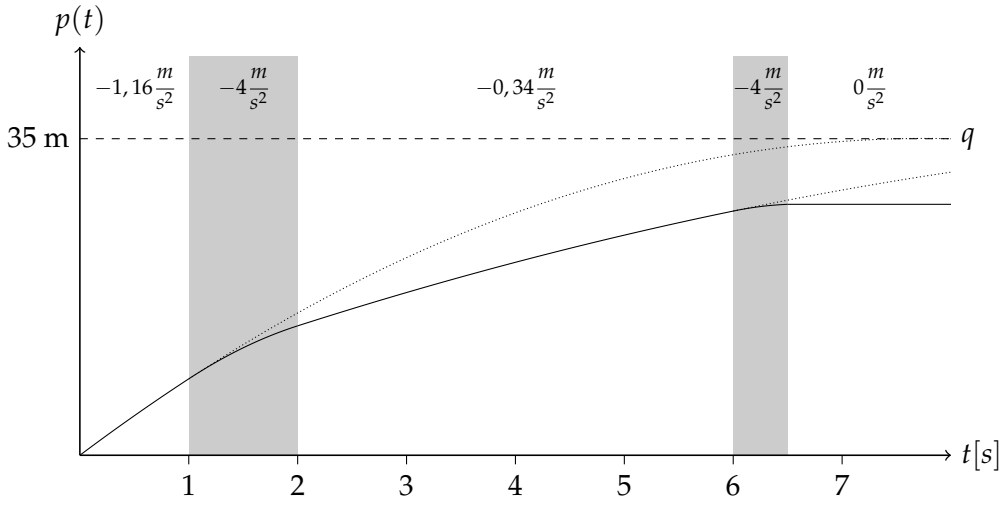

Abbildung 5.3: Unerwünschtes Verhalten des Bremsassistenten

$$\tilde{A} = -\frac{B}{2} - \frac{v}{\epsilon} + \frac{1}{2}\sqrt{B^2 + \frac{8B(q-p)}{\epsilon^2} - \frac{4Bv}{\epsilon}} \tag{5.16}$$

Das Hybride Programm für den optimierten Notbremsassistenten und die d$\mathscr{L}$-Formel für die Verifikation sind in Abbildung 5.4 gezeigt. Die Vorbedingung ist die Gleiche wie für das Hybride Programm NBA und in Gleichung 5.15 gegeben. Die vollautomatisierte Verifikation mit KeYmaera musste ich aufgrund der langen Rechenzeit von mehreren Stunden vorzeitig abbrechen. Auch manuell war es mir nicht möglich, einen Beweis zu finden.

Die Kurve in Abbildung 5.5 zeigt, wie sich der optimierte Notbremsassistent beim Stoppen vor einem Hindernis verhält. Es wurden die gleichen Parameter wie in Abschnitt 5.1.4 verwendet. Die gepunktete Linie zeigt, wie ein Fahrzeug mit einer konstanten Bremskraft von $-1,16m/s^2$ exakt vor dem Hindernis zum Stehen kommt. Die durchgezogene Linie zeigt die Bewegung eines Fahrzeuges, wenn der Controller in jedem Zyklus die maximal mögliche Beschleunigung wählt. Im ersten Zyklus kann das Fahrzeug noch mit maximaler Kraft beschleunigt werden. Im zweiten Zyklus muss die Beschleunigung auf den Bereich $[-4, -3.1]$ beschränkt werden. Wenn der Controller den größten Wert wählt, muss in den folgenden Zyklen bis zum Stillstand des Fahrzeuges mit der maximaler Bremskraft $-4m/s^2$ verzögert werden.

$$
\begin{aligned}
NBA_{opt} \equiv\ & (ctrl; drive)^* \\
ctrl \equiv\ & a := *;\ ?(-B \leq a \leq Min(A, -\tfrac{B}{2} - \tfrac{v}{\epsilon} + \tfrac{1}{2}\sqrt{B^2 + \tfrac{8B(q-p)}{\epsilon^2} - \tfrac{4Bv}{\epsilon}})) \\
drive \equiv\ & t := 0; \{p' = v, v' = a, t' = 1, t \leq \epsilon, v \geq 0\}
\end{aligned}
$$

$$\phi \rightarrow [NBA_{opt}]p \leq q$$

Abbildung 5.4: Hybrides Programm für den optimierten Notbremsassistenten und d$\mathscr{L}$-Formel (s. Listing A.2)

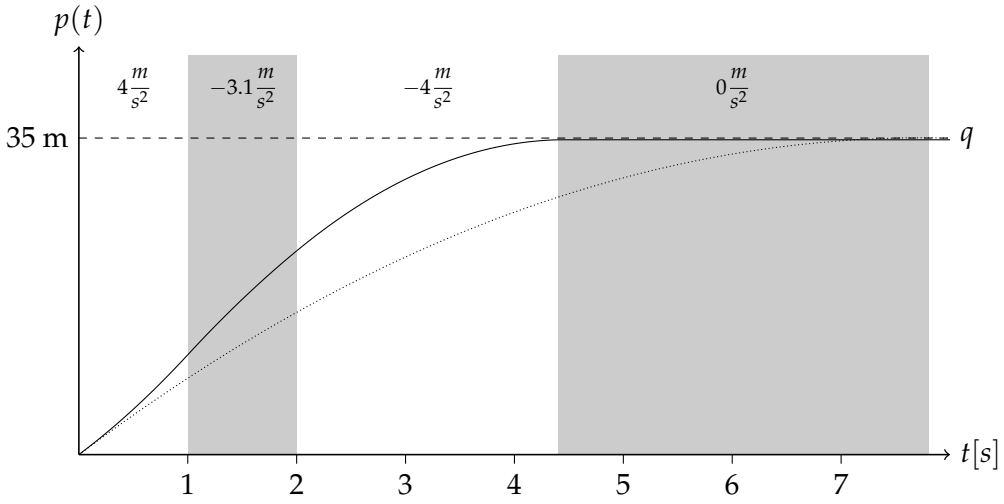

Abbildung 5.5: Optimierter Notbremsassistent

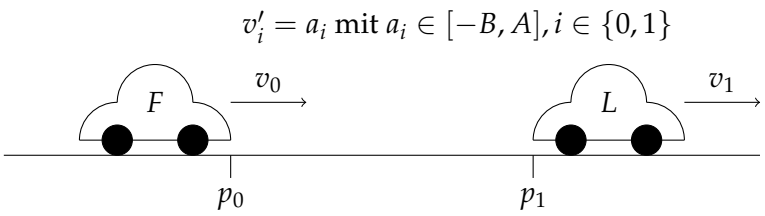

Abbildung 5.6: Intelligenter Tempomat

5.2 Intelligenter Tempomat

Der intelligente Tempomat regelt die Geschwindigkeit eines Fahrzeuges F so, dass unter keinen Umständen eine Kollision mit einem vorausfahrenden Fahrzeug L passieren kann. Das Modell ist eine Erweiterung des Notbremsassistenten aus Abschnitt 5.1 und entspricht dem in [18] gezeigten Model „Local lane control (llc)“ bzw. „Intelligent Cruise Control (ICC) in d$\mathscr{L}$“ aus [19].

5.2.1 Modell

Wie bei dem Notbremsassistenten handelt es sich um ein 1-dimensionales Modell. Aus Abbildung 5.6 wird die Bedeutung der im Modell verwendeten Variablen ersichtlich. Für Position, Geschwindigkeit und Beschleunigung der Fahrzeuge werden die Variablen p_i, v_i und a_i mit $i \in \{0, 1\}$ verwendet. Die Bewegung wird wie in den Gleichungen 5.1 und 5.2 modelliert. B ist die maximale Bremskraft und A die maximale Beschleunigung der Fahrzeuge. In dem Beispiel davon ausgegangen, dass initial $p_0 \leq p_1$ gilt. Das Hybride Programm des intelligenten Tempomates ist in Abbildung 5.7 zu sehen. Das Hauptprogramm iT gibt an, dass die Unterprogramme *ctrl* und *drive* sequentiell und beliebig oft hintereinander ausgeführt werden. In dem Unterprogramm *ctrl* werden wiederum die Unterprogramme $ctrl_0$ und $ctrl_1$ ausgeführt, wobei die Reihenfolge für dieses Modell irrelevant ist. Das Unterprogramm $ctrl_1$ setzt die Beschleunigung des Fahrzeuges L auf einen beliebigen Wert im Bereich $[-B, A]$. Das Unterprogramm $ctrl_0$ unterscheidet drei Fälle. Im ersten Fall wird eine Vollbremsung durchgeführt. Dieser Fall ist ohne Einschränkung erlaubt. Hierbei wird angenommen, dass die Bremskraft im Bereich von $[-B, -b]$ liegt, d.h. die Bremskraft des Fahrzeuges L kann bei einer Vollbremsung stärker sein, als die des Fahrzeuges F. Im zweiten Fall wird die Sicherheitsbedingung $Safe_\epsilon$ geprüft. Ist die Sicherheitsbedingung erfüllt, kann die Beschleunigung des Fahrzeuges F auf einen beliebigen Wert im Bereich $[-B, A]$ gesetzt werden. Im dritten Fall wird die Beschleunigung des Fahrzeuges F auf 0 gesetzt, wenn die Geschwindigkeit bereits 0 ist, so dass das Fahrzeug stehen

$$
\begin{aligned}
iT \equiv\ & (ctrl;\ drive)^* \\
ctrl \equiv\ & ctrl_0;\ ctrl_1 \\
ctrl_1 \equiv\ & (a_1 := *;\ ?(-B \leq a_1 \leq A)) \\
ctrl_0 \equiv\ & (a_0 := *;\ ?(-B \leq a_0 \leq -b)) \\
& \cup (?Safe_\epsilon;\ a_0 := *;\ ?(-B \leq a_0 \leq A)) \\
& \cup (?(v_0 = 0);\ a_0 := 0) \\
drive \equiv\ & t := 0; \{p_0' = v_0, v_0' = a_0, p_1' = v_1, v_1' = a_1, t' = 1, t \leq \epsilon, v_0 \geq 0, v_1 \geq 0\}
\end{aligned}
$$

Abbildung 5.7: Hybrides Programm für den intelligenten Tempomat

bleiben kann. Das Unterprogramm *drive* beschreibt mit Differentialgleichungen die kontinuierliche Zustandsänderung beider Fahrzeuge. Diese ist zeitlich beschränkt auf ϵ Zeiteinheiten und wird nur für nichtnegative Geschwindigkeit v_0 und v_1 ausgeführt.

5.2.2 Herleitung der Sicherheitsbedingung

Mit der Sicherheitsbedingung soll geprüft werden, ob das Fahrzeug für bis zu ϵ Zeiteinheiten eine beliebige Beschleunigung im Bereich $[-B, A]$ wählen darf, ohne dass eine Kollision mit dem vorausfahrenden Fahrzeug unausweichlich wird. Formell ist eine Kollision ausgeschlossen, wenn gilt

$$\forall t \geq 0 : p_0(t) \leq p_1(t) \quad . \tag{5.17}$$

Die Verifikation im Abschnitt 5.2.3 wird zeigen, dass es ausreicht zu prüfen, dass $\hat{p}_0 \leq \hat{p}_1$ gilt. $\hat{p}_0$ ist dabei die Position, die das Fahrzeug F erreicht, nachdem es für ϵ Zeiteinheiten mit A beschleunigt und danach mit $-b$ zum Stillstand abgebremst wurde. $\hat{p}_1$ ist die Position, die das Fahrzeug L ereicht, nachdem es mit $-B$ zum Stillstand abgebremst wurde. $Safe_\epsilon$ ergibt sich somit zu

$$Safe_\epsilon \equiv \hat{p}_0 \leq \hat{p}_1 \quad . \tag{5.18}$$

Die Berechnung für $\hat{p}_0$ und $\hat{p}_1$ sind in den Gleichungen 5.19 und 5.20 gegeben. Diese lassen sich mit Hilfe der Gleichung 5.13 herleiten. Für $\hat{p}_0$ muss B durch b ersetzt werden und für $\hat{p}_1$ muss ϵ durch 0 ersetzt werden, da eine sofortige Vollbremsung angenommen wird. Zusätzlich müssen noch die aktuellen Positionen p_0 bzw. p_1 dazu addiert werden.

$$\hat{p}_0 = p_0 + v_0\epsilon + 0.5A\epsilon^2 + \frac{(v_0 + A\epsilon)^2}{2b} \tag{5.19}$$

$$\hat{p}_1 = p_1 + \frac{v_1^2}{2B} \tag{5.20}$$

Die Gleichungen 5.19 und 5.20 in 5.18 eingesetzt führen zu der gesuchten Gleichung 5.21.

$$Safe_\epsilon \equiv p_0 + v_0\epsilon + 0.5A\epsilon^2 + \frac{(v_0 + A\epsilon)^2}{2b} \leq p_1 + \frac{v_1^2}{2B} \tag{5.21}$$

5.2.3 Verifikation

Zur Verifikation der Kollisionsfreiheit ist die Formel

$$\phi \rightarrow [iT] p_0 \leq p_1 \tag{5.22}$$

auszuwerten (vergl. Formel 5.14). Diese Formel lässt sich mit dem Theorembeweiser KeYmaera nicht mehr vollständig automatisiert beweisen. In [18] ist angegeben, wie sich diese Formel mit Benutzerinteraktion beweisen lässt. Unter [20] ist der vollständige Beweis in der Datei „llc.key.proof“ zu finden. Dieser enthält allerdings 656 Benutzerinteraktionen und wird daher nicht näher erläutert.

5.2.4 Bemerkungen

Die kontinuierliche Entwicklung in dem Hybriden Programm iT enthält die Bedingungen $t \leq \epsilon$, $v_0 \geq 0$ und $v_1 \geq 0$. Mit der ersten Bedingung soll erreicht werden, dass bis zu ϵ Zeiteinheiten vergehen dürfen, bis der Controller eine neue Entscheidung treffen muss. Allerdings kann es zuvor passieren, dass das Fahrzeug F oder L zum Stehen kommt und die Bedingung $v_0 \geq 0$ bzw. $v_1 \geq 0$ nicht mehr gültig ist. Dies würde bedeuten, dass der Controller schon vor Ablauf der Zeit ϵ eine neue Entscheidung zu treffen hätte. ϵ beschränkt damit nicht mehr den maximal zugelassenen zeitlichen Abstand zwischen zwei Entscheidungen und kann damit nicht mehr verwendet werden, um Hardwareanforderungen eines realen Systems zu bestimmen. Dieses Problem kann gelöst werden, indem das Unterprogramm *drive* in

$$
\begin{aligned}
drive_1 \equiv & t := 0; \{ \\
& (p_0' = v_0, v_0' = a_0, p_1' = v_1, v_1' = a_1, t' = 1, t \leq \epsilon, v_0 \geq 0, v_1 \geq 0) \vee \\
& (p_0' = v_0, v_0' = a_0, t' = 1, t \leq \epsilon, v_0 \geq 0) \vee \\
& (p_1' = v_1, v_1' = a_1, t' = 1, t \leq \epsilon, v_1 \geq 0) \vee \\
& (t' = 1, t \leq \epsilon) \}
\end{aligned}
\tag{5.23}
$$

abgeändert wird. Durch diese Änderung kann das Programm mit der kontinuierlichen Entwicklung fortfahren, auch wenn bereits ein Fahrzeug zum Stehen gekommen ist. Es ist leicht zu sehen, dass die letztgenannte Bedingung der Disjunktion weggelassen werden kann, da nur die Variable t verändert wird, die für den Rest des Programms nicht von Bedeutung ist. Nachzuweisen ist aber, ob weitere Bedingungen weggelassen werden dürfen.

Eine weitere Auffälligkeit ist, dass beide Fahrzeuge synchron getaktet sind. Beiden wird zum gleichen Zeitpunkt eine neue Beschleunigung zugewiesen. Dieses Problem kann gelöst werden, indem die Beschleunigung des Fahrzeuges L mit Differential-Ungleichungen beschrieben wird (s. Gleichung 5.24).

$$
...\{..., p_1' = v_1, -B \leq v_1' \leq A, ...\}... \tag{5.24}
$$

Da in dem Hybriden Programm iT der Wert von a_1 bei der Berechnung von $Safe_\epsilon$ nicht berücksichtigt wird, tritt dieses Problem hier nicht auf. Allerdings sollte es berücksichtigt werden, wenn Änderungen an der Berechnung von $Safe_\epsilon$ vorgenommen werden.

5.3 Modellierung von Quadrocoptern

Die Dynamik der bisher betrachteten Systeme ist durch sehr einfache Differentialgleichungen beschrieben. Reale Systeme sind meist deutlich komplexer. In diesem Abschnitt zeige ich Ansätze, mit denen es gelingen kann, Quadrocopter zu modellieren.

5.3.1 Kollisionsvermeidung mit beliebiger Geschwindigkeitsbeschränkung

In den Modellen für den Notbremsassisten und den intelligenten Tempomaten wird vorrausgesetzt, dass sich Fahrzeuge nur vorwärts bewegen. Dies kann für Fahrzeuge, die sich grundsätzlich nur in eine Richtung bewegen, eine annehmbare Vereinfachung

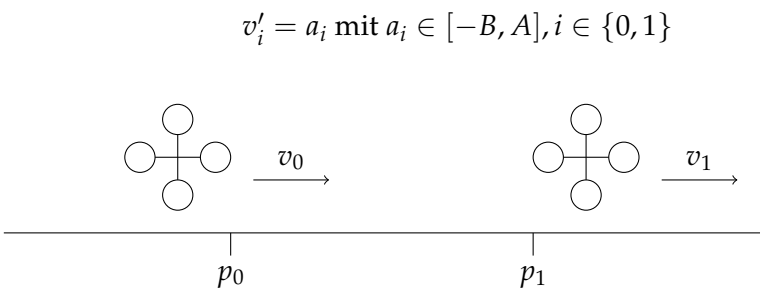

Abbildung 5.8: Intelligenter Tempomat für Quadrocopter

sein. Für Quadrocopter müssen die Modelle aber auch mit negative Geschwindigkeiten umgehen können.

Im Folgenden zeige ich, wie das Modell des intelligenten Tempomaten um beliebige Geschwindigkeitsbeschränkungen erweitert werden kann. Zur Vereinfachung werden nach wie vor nur 1-dimensionale Modelle betrachtet. Demnach bewegen sich die Fahrzeuge wie in Abbildung 5.8 auf einer Linie. Weiterhin wird die in Gleichung 5.1 und 5.2 angegebene Bewegungsgleichung verwendet.

Modelle ohne Geschwindigkeitsbeschränkung

Modelle ohne Beschränkung der Geschwindigkeit können nicht kollisionsfrei sein, wenn beide Fahrzeuge die gleiche maximale Bremskraft haben. Der Notbremsassistent und der intelligente Tempomat aus den vorherigen Abschnitten haben daher eine untere Geschwindigkeitsbeschränkung von 0.

Unter der Annahme, dass beide Fahrzeuge zu einem beliebigen Zeitpunkt $\tilde{t}$ ihre Beschleunigung auf $-B$ setzen, ist die Geschwindigkeitsdifferenz d_v zwischen den beiden Fahrzeugen ab diesem Zeitpunkt konstant (s. Gleichung 5.25 bis 5.28). d_v ist nur von den zum Zeitpunkt $\tilde{t}$ gefahrenen Geschwindigkeiten $v_{0,initial}$ und $v_{1,initial}$ abhängig. Wenn d_v negativ ist, wird der Abstand zwischen den Fahrzeugen geringer und eine Kollision kann von dem Fahrzeug F nicht mehr abgewendet werden. Existiert allerdings für beide Fahrzeuge eine untere Geschwindigkeitsbeschränkung, wird die Geschwindigkeitsdifferenz zu 0, wenn beide Fahrzeuge diese erreicht haben. In diesem Fall wäre eine Kollision nicht mehr möglich, da beide Fahrzeuge mit der gleichen Geschwindigkeit fahren.

$$v_0' = v_1' = -B \tag{5.25}$$
$$v_0(t) = v_{0,initial} - Bt \tag{5.26}$$
$$v_1(t) = v_{1,initial} - Bt \tag{5.27}$$
$$d_v = v_1(t) - v_0(t) = v_{1,initial} - v_{0,initial} \tag{5.28}$$

Modelle mit Geschwindigkeitsbeschränkung

Um eine beliebige untere Geschwindigkeitsbeschränkung einzuführen, muss die Gleichung 5.10 in Gleichung 5.29 abgeändert werden.

$$\hat{v}_{V_{min}}(t) = \tilde{v} - B\tilde{t} \stackrel{!}{=} V_{min} \tag{5.29}$$
$$\tilde{t}_{V_{min}} = \frac{\tilde{v} - V_{min}}{B} \tag{5.30}$$

Der Bremsweg ergibt sich dann zu

$$s_{B,V_{min}}(v) = \frac{1}{2B}(2(v + A\epsilon)(v + A\epsilon - V_{min}) - (v + A\epsilon - V_{min})^2). \tag{5.31}$$

Die Sicherheitsbedingung für den intelligenten Tempomaten ergibt sich aus Gleichung 5.21 zu 5.32.

$$\begin{aligned} Safe_\epsilon \equiv & p_0 + v_0\epsilon + 0.5A\epsilon^2 + \\ & \frac{1}{2B}(2(v + A\epsilon)(v + A\epsilon - V_{min}) - (v + A\epsilon - V_{min})^2) \\ & \leq p_1 + \frac{1}{2B}(2v(v - V_{min}) - (v - V_{min})^2) \end{aligned} \tag{5.32}$$

Die Vorbedingung für den intelligenten Tempomaten mit beliebiger Geschwindigkeitsbeschränkung ist in Gleichung 5.33 angegeben und entspricht im Wesentlichen der Gleichung 5.15. Lediglich die letztgenannte Bedingung muss um die minimale Geschwindigkeit V_{min} erweitert werden.

$$\phi \equiv B > 0 \wedge A \geq 0 \wedge \epsilon > 0 \wedge p \leq q \wedge v^2 - V_{min}^2 \leq 2B(q - p) \tag{5.33}$$

5.3.2 Kollisionsvermeidung im n-dimensionalen Raum

Um Kollisionen zu vermeiden darf ein Hybrides System nicht in einen Zustand gelangen, durch den eine Kollision unausweichlich wird. Der Controller des Hybrides Systems muss diese Zustände kennen, um jederzeit die richtigen Entscheidungen treffen zu können.

Eine Kollision hat stattgefunden, wenn die Schnittmenge der Körper zweier Fahrzeuge nicht leer ist. Seien P und Q die Körper zweier unterschiedlicher Fahrzeuge, die sich in Abhängigkeit von der Zeit verändern, kann Kollisionsfreiheit formell durch

$$\forall t \in \mathbb{R} : P(t) \cap Q(t) = \emptyset \tag{5.34}$$

ausgedrückt werden. Werden die Körper der Fahrzeuge durch Kugeln näherungsweise beschrieben, kann man die Gleichung zu

$$\forall t \in \mathbb{R} : \|p(t) - q(t)\|_2 \geq C \tag{5.35}$$

vereinfachen. $p(t)$ und $q(t)$ sind Punkte im n-dimensionalen Raum, die sich im Zentrum der Kugeln befinden und C ist die Summe der Radien der beiden Kugeln.

Für den Notbremsassistenten aus Abschnitt 5.1 wurden Annahmen getroffen, die zu einer deutlichen Vereinfachung führen. Das Modell ist nur 1-dimensional, p und q werden so gesetzt, dass C zu 0 wird und initial gilt $p \leq q$. Dies vereinfacht die Gleichung 5.35 zu

$$\forall t \in \mathbb{R} : q(t) - p(t) \geq 0 \quad . \tag{5.36}$$

Dies bedeutet, dass eine Kollision stattgefunden hat, wenn $p(t)$ im Intervall $[q(t), \infty]$ liegt. Zudem ist die Vollbremsung die einzige Strategie um eine Kollision zu vermeiden. Wenn das Fahrzeug nach einer Vollbremsung zum Zeitpunkt $\tilde{t}$ zum Stillstand gekommen ist, dann genügt es, genau diesen Zeitpunkt zu untersuchen. Dies vereinfacht die Gleichung zu

$$q(\tilde{t}) - p(\tilde{t}) \geq 0 \quad . \tag{5.37}$$

Weiterhin ist $q(t)$ konstant und $p(t)$ lässt sich aufgrund der verwendeten Differentialgleichungen leicht bestimmen.

Im n-dimensionalen Raum (mit $n > 1$) können diese Vereinfachungen nicht mehr angewendet werden. Neben der Vollbremsung zur Kollisionsvermeidung besteht auch die Möglichkeit des Ausweichens. Ein Zeitpunkt, an dem das Fahrzeug zum Stillstand kommt, ist somit nicht mehr zwingend vorhanden.

Ein möglicher Ansatz, um im n-dimensionalen Raum zu erkennen, ob eine Kollision erfolgreich vermieden wurde, ist die Untersuchung der Abstandsänderung zwischen Fahrzeugen. Sei

$$d_{p,q}(t) = \|p(t) - q(t)\|_2^2 \tag{5.38}$$

der quadratische euklidische Abstand zwischen zwei Fahrzeugen, dann ist die Änderung des quadratischen Abstandes durch

$$d'_{p,q}(t) = 2 \sum_{i=0}^{n-1} (p_i(t) - q_i(t))(p'_i(t) - q'_i(t)) \tag{5.39}$$

gegeben. Wenn $d'_{p,q}(t) < 0$ gilt, wird der Abstand zwischen den Fahrzeugen geringer. Gilt $d'_{p,q}(t) > 0$, so wird er größer. Setzt man $d'_{p,q}(t)$ gleich 0, kann man den Zeitpunkt bestimmen, an dem der Abstand zwischen den beiden Fahrzeuge am geringsten ist. Mit der zweiten Ableitung

$$d''_{p,q}(t) = 2 \sum_{i=0}^{n-1} (p'_i(t) - q'_i(t))^2 + (p_i(t) - q_i(t))(p''_i(t) - q''_i(t)) \tag{5.40}$$

kann mit $d''_{p,q}(t) > 0$ sichergestellt werden, dass es sich um ein Minimum handelt.

5.3.3 Bewegungsgleichung eines Quadrocopters

Die Bewegungsgleichung eines Quadrocopters ist in den Gleichungen 5.41 bis 5.45 gegeben [21, Kap. 10]. Dieses System hat 4-fach integrales Verhalten. F_1 bis F_4 sind die an den 4 Rotoren anliegenden Kräfte, die umgerechnet werden in Gesamtauftrieb u_1, Roll-Winkeländerung u_2, Pitch-Winkeländerung u_3 und Yaw-Winkeländerung u_4 (s. Abbildung 5.9). Mit ω_x, ω_y und ω_z werden die Winkelgeschwindigkeiten um jede Achse beschrieben. ω'_x, ω'_y und ω'_z sind dementsprechend die Beschleunigungen der Winkelgeschwindigkeiten. ψ, θ und ϕ sind Roll-, Pitch- und Yaw-Winkel, v_x, v_y und v_z sind die Geschwindigkeiten in alle Richtungen, p_x, p_y und p_z beschreibt die Position des Quadrocopters, J_x, J_y und J_z sind die Trägheitsmomente um jede Achse, l ist der Abstand zwischen Rotormittelpunkt und Zentrum des Quadrocopters und c

ist eine Proportionalitätskonstante für die Drehung um die z-Achse. Noch komplexer wird das System, würde die Trägheit der einzelnen Rotoren und die Eigenschaften des Motor-Reglers berücksichtigt werden.

$$\begin{aligned} p'_x &= v_x \\ p'_y &= v_y \\ p'_z &= v_z \end{aligned} \tag{5.41}$$

$$\begin{aligned} v'_x &= -\frac{1}{m}u_1 + (sin(\phi)sin(\psi) + cos(\phi)sin(\theta)cos(\psi)) \\ v'_y &= -\frac{1}{m}u_1 + (sin(\phi)cos(\psi) + cos(\phi)sin(\theta)sin(\psi)) \\ v'_z &= -\frac{1}{m}u_1 cos(\phi)cos(\theta) + g \end{aligned} \tag{5.42}$$

$$\begin{aligned} \psi' &= \omega_x \\ \theta' &= \omega_y \\ \phi' &= \omega_z \end{aligned} \tag{5.43}$$

$$\begin{aligned} \omega'_x &= \omega_y\omega_z\frac{J_y - J_z}{J_x} + \frac{l}{J_x}u_2 \\ \omega'_y &= \omega_x\omega_z\frac{J_z - J_x}{J_y} + \frac{l}{J_y}u_3 \\ \omega'_z &= \omega_x\omega_y\frac{J_x - J_y}{J_z} + \frac{lc}{J_z}u_4 \end{aligned} \tag{5.44}$$

$$\begin{aligned} u_1 &= F_1 + F_2 + F_3 + F_4 \\ u_2 &= F_4 - F_2 \\ u_3 &= F_1 - F_3 \\ u_4 &= F_1 + F_3 - F_2 - F_4 \end{aligned} \tag{5.45}$$

Im Abschnitt 6.1 werde ich zeigen, dass KeYmaera Erreichbarkeitsanalysen mit Systemen im 3-Dimensionalen Raum mit bis zu 5-fach integralem Verhalten in kurzer Zeit bewerkstelligen kann, wenn die Differentialgleichungen einfach sind und keine

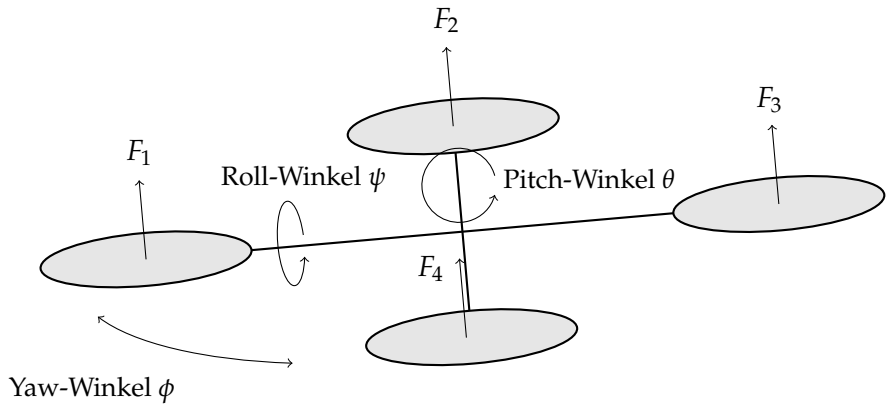

Abbildung 5.9: Bewegungen eines Quadrocopters Pitch, Roll, Yaw

Schleifen verwendet werden. Eine Erreichbarkeitsanalyse mit dem deutlich komplexeren dynamischen Verhalten eines Quadrocopters führte allerdings zu keinem Erfolg. Für eine Analyse ist die deutliche Vereinfachung des Modells erforderlich (s. Abschnitt 5.3.4). Zudem ist die Vereinfachung sinnvoll, um Strategien zur Kollisionsvermeidung zu entwickeln.

5.3.4 Ansatz zur Vereinfachung der Bewegungsgleichung

Zur leichteren Lesbarkeit beschränke ich mich hier auf ein 1-dimensionales System. Eine Erweiterung auf mehrere Dimensionen wäre problemlos möglich.

Sei v_{soll} die Stellgröße des vereinfachten Modells eines Quadrocopters, v_0 die aktuelle Geschwindigkeit und D eine positive Konstante, so kann mit zwei einfachen Differentialgleichungen (s. Gleichung 5.46 und Abbildung 5.10) ein Bereich definiert werden, der zum Zeitpunkt $t = 0$ die aktuelle Geschwindigkeit v_0 und ab einem späteren Zeitpunkt $t \geq \tilde{t}$ die gewünschte Geschwindigkeit v_{soll} einschließt. Kann nachgewiesen werden, dass für die reale Geschwindigkeit $v(t)$ des Quadrocopters Gleichung 5.47 gilt, also $v(t)$ von $v_t(t)$ und $v_b(t)$ eingeschlossen wird, so ist das Modell eine gültige Verallgemeinerung.

$$\begin{aligned} v_t'(t) &= v_{soll} + D - v_t(t) \quad mit\ v_t(0) = v_0 + D \\ v_b'(t) &= v_{soll} - D - v_b(t) \quad mit\ v_b(0) = v_0 - D \end{aligned} \tag{5.46}$$

$$\forall t \geq 0 : v_b(t) \leq v(t) \leq v_t(t) \tag{5.47}$$

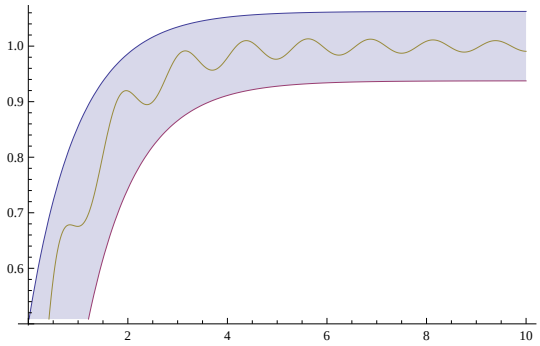

Abbildung 5.10: Einschließung des realen Verhaltens der Geschwindigkeit eines Quadrocopters durch zwei einfache Differentialgleichungen

Mit Hilfe einer DA-Bedingung lässt sich dieses vereinfachte Modell mit DAL modellieren (s. Abbildung 5.11).

Eine vollautomatisierte Erreichbarkeitsanalyse mit diesem Modell führte auch nach mehreren Stunden zu keinem Ergebnis. Auch das Entfernen der Schleife ($*$-Operator) brachte keinen Erfolg.

Die Lösung der Differentialgleichungen für v_t und v_b mit den gegebenen Anfangswerten ist in Gleichung 5.48 und 5.49 angegeben. Da Terme in den Logiken d$\mathscr{L}$, DAL und dTL per Definition keine Exponentialfunktionen enthalten können, die nicht als endliches Produkt umgeschrieben werden können [2, S. 40], kann die Lösung bei der Suche nach einem Beweis oder bei der Formulierung von Sicherheitsbedingungen nicht verwendet werden.

$$v_t(t) = e^{-t}(v_0 - v_{soll} + e^t v_{soll} + De^t) \tag{5.48}$$

$$v_b(t) = e^{-t}(v_0 + v_{soll} + e^t v_{soll} - De^t) \tag{5.49}$$

$$
\begin{aligned}
QDRCPTR \equiv\ & (ctrl; fly)^* \\
ctrl \equiv\ & v_{soll} := *; ?(V_{min} \leq v_{soll} \leq V_{max}) \\
fly \equiv\ & t := 0; \\
& v_t := v + D; \\
& v_b := v - D; \\
& \{\ \ \exists \tilde{v}(\ p'_x = \tilde{v} \quad \wedge \quad v_b \leq \tilde{v} \leq v_t\ ) \quad \wedge \\
& \quad v'_t = v_{soll} + D - v_t \quad \wedge \\
& \quad v'_b = v_{soll} - D - v_b \quad \wedge \\
& \quad v' = v_{soll} - v \quad \wedge \\
& \quad t' = 1 \quad \wedge \quad t \leq \epsilon \quad \}
\end{aligned}
$$

Abbildung 5.11: Hybrides Programm des vereinfachten Modells eines Quadrocopters (1-dimensional)

6 Vergleich vollautomatisierter Erreichbarkeits- und Sicherheitsanalysen

In diesem Kapitel werden Erreichbarkeits- und Sicherheitsanalysen unterschiedlicher Komplexität miteinander verglichen. Es wird gezeigt, welche Auswirkung die Komplexität auf das Laufzeitverhalten des Theorembeweisers KeYmaera hat. Die Modelle sind daher so gewählt, dass eine vollautomatisierte Verifikation möglich ist.

6.1 Erreichbarkeitsanalysen unterschiedlicher Komplexität

Mit einer Erreichbarkeitsanalyse wird überprüft, ob ein Hybrides System bestimmte Zustände erreichen kann. In dem hier betrachteten Fall wird überprüft, ob ein Fahrzeug im n-dimensionalen Raum mit m-fach integralem Verhalten jeden beliebigen Punkt im Raum anfahren kann. Die Dynamik des Fahrzeuges wird mit den in 6.1 gegebenen Differentialgleichungen beschrieben. Die Komplexität des Systems wird durch Wahl der Parameter n und m verändert.

$$\begin{array}{cccc} x'_{0,0} = x_{0,1} & x_{1,0} = x'_{1,1} & \cdots & x_{n-1,0} = x'_{n-1,1} \\ x'_{0,1} = x_{0,2} & x_{1,1} = x'_{1,2} & \cdots & x_{n-1,1} = x'_{n-1,2} \\ \vdots & \vdots & \ddots & \vdots \\ x'_{0,m-1} = x_{0,m} & x_{1,m-1} = x'_{1,m} & \cdots & x_{n-1,m-1} = x'_{n-1,m} \end{array} \tag{6.1}$$

Das Hybride Programm ist in Abbildung 6.1 zu sehen. Stellgrößen sind die Variablen $x_{0,m}$, $x_{1,m}$, ..., $x_{n,m}$.

$$\begin{aligned} EAN \equiv\ & (ctrl; drive) \\ ctrl \equiv\ & x_0 m := *; x_1 m := *; ... x_n m := * \\ drive \equiv\ & \{x'_{0,0} = x_{0,1}, x'_{1,0} = x_{1,1}, ..., x'_{n-1,0} = x_{n-1,1}, \\ & x'_{0,1} = x_{0,2}, x'_{1,1} = x_{1,2}, ..., x'_{n-1,1} = x_{n-1,2}, \\ & ... \\ & x'_{0,m-1} = x_{0,m}, x'_{1,m-1} = x_{1,m}, ..., x'_{n-1,m-1} = x_{n-1,m}\} \end{aligned}$$

Abbildung 6.1: Hybrides Programm für die Erreichbarkeintsanalyse

n	m	Nodes	Time [s]	Arithmetic Solver Time [s]	Arithmetic Memory [Mb]
1	1	7	0,500	0,123 (24,6 %)	36,066
1	2	7	0,692	0,230 (33,2 %)	33,475
1	3	7	3,085	2,580 (83,6 %)	40,086
1	4	7	6,584	6,141 (93,3 %)	41,354
1	5	7	6,621	6,149 (92,9 %)	41,340
2	1	11	0,699	0,241 (34,5 %)	36,214
2	2	12	6,646	6,153 (92,6 %)	42,360
2	3	12	6,771	6,167 (91,1 %)	46,591
2	4	12	6,868	6,250 (91,0 %)	51,199
2	5	12	6,952	6,241 (89,8 %)	80,106
3	1	17	2,991	2,377 (79,5 %)	40,942
3	2	17	6,813	6,238 (91,6 %)	66,163
3	3	17	6,947	6,225 (89,6 %)	48,254
3	4	17	7,957	6,286 (79,0 %)	49,867
3	5	17	7,151	6,298 (88,1 %)	80,058

Tabelle 6.1: Ergebnisse der Erreichbarkeitsanalysen mit IBC

Für die Erreichbarkeitsanalysen wurde die Formel 6.2 mit KeYmaera verifiziert. Die Variablen $t_0, t_1, ..., t_{n-1}$ beschreiben einen beliebigen Punkt im n-dimensionalen Raum. In Tabelle 6.1 ist der zeitliche Aufwand der Berechnungen zu sehen, den KeYmaera zum Finden eines Beweises benötigt hat. Als „First-Order"-Strategie wurde IBC gewählt. In Listing A.6 ist der Quelltext mit der Parameterwahl $n = 2$ und $m = 3$ abgebildet.

$$\langle EAN \rangle (x_{0,0} = t_0 \wedge x_{1,0} = t_1 \wedge \cdots \wedge x_{n,0} = t_n) \tag{6.2}$$

Aus den Ergebnissen wird ersichtlich, dass KeYmaera die Erreichbarkeitsanalyse auch bei steigender Komplexität in kurzer Zeit lösen kann. Jeder Beweis war in weniger als 8 Sekunden gefunden.

Allerdings reicht eine leichte Modifikation des Hybriden Programms EAN aus, damit ein Beweis nicht mehr vollautomatisiert gefunden wird. Führt man $(ctrl; drive)$ in einer Schleife aus, so bricht KeYmaera die Suche nach kurzer Zeit ab. Selbst mit $n = 1$ und $m = 0$ (s. Formel 6.3) und unabhängig von der gewählten „First Order"-Strategie (*lazy*, *eager*, *IBC*) konnte KeYmaera keinen Beweis vollautomatisiert finden. Manuell gefundene Beweise für $n = 1, m = 0$ und $n = 1, m = 1$ sind auf der beigelegten CD zu finden.

$$
\begin{aligned}
SAN \equiv\ & (ctrl; drive) \\
ctrl \equiv\ & x_0 m := *; x_1 m := *; ... x_n m := *; \\
& (?Safe_{SAN}) \\
drive \equiv\ & t := 0; \\
& \{x'_{0,0} = x_{0,1}, x'_{1,0} = x_{1,1}, ..., x'_{n-1,0} = x_{n-1,1}, \\
& x'_{0,1} = x_{0,2}, x'_{1,1} = x_{1,2}, ..., x'_{n-1,1} = x_{n-1,2}, \\
& ... \\
& x'_{0,m-1} = x_{0,m}, x'_{1,m-1} = x_{1,m}, ..., x'_{n-1,m-1} = x_{n-1,m}, \\
& t' = 1, t \leq eps\}
\end{aligned}
$$

Abbildung 6.2: Hybrides Programm für die Sicherheitsanalyse

$$\langle (x_{0,0} := *)* \rangle (x_{0,0} = t_0) \tag{6.3}$$

6.2 Sicherheitsanalysen unterschiedlicher Komplexität

Mit der Sicherheitsanalyse wird überprüft, ob ein Hybrides System bestimmte Zustände auf gar keinen Fall erreichen kann. In dem hier betrachteten Fall wird überprüft, ob ein Fahrzeug im n-dimensionalen Raum mit m-fach integralem Verhalten immer einen bestimmten Mindestabstand zu einem beliebigen, nicht bewegten Punkt im Raum einhält. Die Dynamik des Fahrzeuges wird wie bei den Erreichbarkeitsanalysen mit dem in 6.1 gegebenen Differentialgleichungen beschrieben. Das Hybride Programm ist in Abbildung 6.2 zu sehen. Stellgrößen sind wie bei den Erreichbarkeitsanalysen die Variablen $x_{0,m}$, $x_{1,m}$, ..., $x_{n,m}$. Nachdem in *ctrl* die Stellgrößen auf zufällige Werte gesetzt wurden, wird geprüft, ob diese Wahl sicher ist. Wenn dies nicht der Fall ist, wird das Programm abgebrochen. Es wird nicht versucht das Problem durch Bremsen oder Ausweichen zu lösen, da es hierzu notwendig wäre, gültige Sicherheitsbedingungen und Strategien zur Kollisionsvermeidung zu entwerfen.

Die Ausführung von *drive* ist auf ϵ Zeiteinheiten beschränkt, da $Safe_{SAN}$ (s. Formel 6.4) von dieser Eigenschaft gebrauch macht.

Wie bei den Erreichbarkeitsanalysen wird das Hybride Programm SAN ebenfalls ohne Schleife ausgeführt. Andernfalls war es nicht möglich mit KeYmaera vollautomatisiert einen Beweis zu finden.

Die Sicherheitsbedingung $Safe_{SAN}$ drückt aus, dass während des *drive*-Zyklus (im Zeitraum 0 bis ϵ) der Abstand zwischen Fahrzeug und Hindernis immer größer als C sein muss. $\vec{\mathscr{S}}_\tau$ ist die Lösung des Differentialgleichungssystems 6.1 mit eingesetztem τ. $\vec{q}$ beschreibt die Position des Hindernisses im n-dimensionalen Raum.

n	m	Nodes	Time [s]	Arithmetic Solver Time [s]	Arithmetic Memory [MB]
1	1	16	1,481	0,895 (60,4 %)	36,265
1	2	21	24,961	24,337 (97,5 %)	72,165
1	3	-	-	-	[1]
2	1	20	3,322	2,652 (79,8 %)	48,644
2	2	-	-	-	> 1.4 GB [2]
3	1	24	3,830	3,136 (81,9 %)	42,901
3	2	-	-	-	> 1.4 GB [2]

Tabelle 6.2: Ergebnisse der Sicherheitsanalysen mit IBC

$$Safe_{SAN} \equiv \forall \tau : 0 \leq \tau \leq \epsilon \rightarrow \|\vec{\mathscr{P}}_\tau - \vec{q}\|_2^2 > C^2 \tag{6.4}$$

Für die Sicherheitsanalyse habe ich die Formel 6.5 mit KeYmaera verifiziert.
In Listing A.7 ist der Quelltext mit der Parameterwahl $n = 2$ und $m = 2$ abgebildet.
Die Ergebnisse der Analysen sind in Tabelle 6.2 abgebildet.

$$\|\vec{x} - \vec{q}\|_2^2 > C^2 \rightarrow [SAN]\|\vec{x} - \vec{q}\|_2^2 > C^2 \tag{6.5}$$

6.3 Bemerkungen

Die hier gezeigten Modelle sind sehr einfach gehalten, da eine vollautomatisierte Verifikation sonst nicht möglich gewesen wäre. Die Ergebnisse zeigen, dass der größere Teil der Rechenzeit in Mathematica entstanden ist (im Schnitt ca. 78 %). Durch die Ergebnisse der Sicherheitsanalysen wird deutliche, dass der Suchalgorithmus von KeYmaera sehr schnell an seine Grenzen stößt.

Die von mir für KeYmaera verwendeten Einstellungen und Daten zu dem von mir verwendeten PC sind im Anhang B zu finden. Nach jeder durchgeführten Verifikation habe ich KeYmaera neu gestartet, da das Programm sonst auf gepufferte Daten zurückgreifen könnte.

[1] KeYmaera konnte nach über 3 h keinen Beweis finden.
[2] KeYmaera konnte nach über 1 h keinen Beweis finden. Der Speicherbedarf des Prozesses MathKernel.exe stieg auf über 1,4 GB so dass ich die Berechnung abgebrochen habe.

7 QdL - Verifikation mit n Fahrzeugen

Quantified Differential Dynamic Logic (QdL) ermöglicht in einem Modell eine beliebige Anzahl von Fahrzeugen zu modellieren [22]. Dies ist in den bisher beschriebenen Logiken d$\mathscr{L}$, DAL und dTL nicht möglich. In dem Hybriden Programm für den intelligenten Tempomaten (s. Abbildung 5.7) musste beispielsweise das dynamische Verhalten für beide Fahrzeuge angegeben werden, obwohl es sich um die gleichen Differentialgleichungen handelt. Dieses Problem wird mit der Logik QdL gelöst.

In QdL wird der Begriff „sorts“ (*Sorte*) eingeführt. Eine *Sorte* ist eine Menge zunächst nicht näher definierter Objekte. Zur Verifikation des intelligenten Tempomaten aus Abschnitt 5.2 mit n Fahrzeugen wird eine *Sorte* für Objekte erstellt, die ein Auto repräsentiert. Das dynamische Verhalten muss somit nicht mehr für jedes einzelne Fahrzeug angegeben werden, sondern könnte mit dem $\forall$-Quantor für eine *Sorte* festgelegt werden, so dass alle Fahrzeuge dieser *Sorte* den gleichen Differentialgleichungen folgen würden.

In Listing 7.1 ist ein Beispiel einer QdL-Formel zu sehen. Diese stammt aus den Beispiel-Projekten der Software KeYmaera (Version 2.1). In den Zeilen 1 bis 3 wird die *Sorte C* deklariert. Für Position, Geschwindigkeit und Beschleunigung der Fahrzeuge (der *Sorte C*) werden in den Zeilen 6 bis 8 Funktionen deklariert, die als Parameter eine Variable vom Typ C erwarten. Um die Beschleunigung des Fahrzeuges i zu setzen, kann der Befehl $a(i) := B$ verwendet werden. i muss dabei vom Typ C und B vom Typ R sein.

Die Funktion e in Zeile 5 wird im Programm verwendet, um Fahrzeuge zu erzeugen und zu löschen. Das Erzeugen und Löschen von Fahrzeugen kann verwendet werden, um das Auffahren eines Fahrzeuges auf oder das Abfahren von einer Straße zu modellieren. Gilt $e(i) = 1$, so existiert das Fahrzeug i, gilt $e(i) = 0$, so existiert es nicht. Mit $e(i) := 1$ kann das Fahrzeug i erzeugt werden und mit $e(i) := 0$ kann es gelöscht werden.

Die Variable n vom Typ C in Zeile 9 wird als Hilfsvariable verwendet, um ein beliebiges Fahrzeug auswählen zu können.

Die Vorbedingung in Zeile 12 bis 16 drückt aus, das für alle Fahrzeuge i und j, die existieren und verschieden sind (Zeile 14) gilt, dass das Fahrzeug mit der geringeren Position $x(C)$ maximal die Geschwindigkeit $v(C)$ und Beschleunigung $a(C)$ des

Fahrzeuges mit der größeren Position hat.

In Zeile 18 bis 20 wird mit $n := *$ ein beliebiges Fahrzeug ausgewält. Existiert dieses noch nicht ($?(e(n) = 0);$) so wird es erstellt ($e(n) := 1;$). Mit Zeile 21 wird festgelegt, dass das neu erzeugte Fahrzeug n dementsprechend in der Reihe platziert sein muss, dass die Vorbedingung immer noch gilt.

In Zeile 22 bis 24 wird das dynamische Verhalten aller Fahrzeuge beschrieben. Die Zeilen 25 bis 28 enthalten die Nachbedingung, die verlangt, dass alle existierenden Fahrzeuge auf unterschiedlichen Positionen stehen, es also in keiner Ausführung des Hybriden Programms zu einer Kollision gekommen ist.

QdL steht seit der Version 2.1 in KeYmaera zur Verfügung. Diese Version wurde allerdings erst zum Ende meiner Arbeit veröffentlicht. Eine vollständige Definition für QdL ist in [22] zu finden.

```
1  \sorts {
2    C;
3  }
4  \functions {
5    R e(C);
6    R x(C);
7    R v(C);
8    R a(C);
9    C n;
10 }
11 \problem {
12    \forall C i.
13      \forall C j.
14        (    e(i) = 1 & e(j) = 1 & !i = j
15         ->   x(i) <  x(j) & v(i) <= v(j) & a(i) <= a(j)
16            | x(i) >  x(j) & v(i) >= v(j) & a(i) >= a(j))
17 -> \[
18      (n := * ;
19        (?(e(n) = 0) ;
20          (e(n) := 1 ;
21            (?(\forall C i. (e(i) = 1 -> (x(i) < x(n) & (v(i) <= v(n) & a(i)
                  <= a(n)) | x(i) > x(n) & (v(i) >= v(n) & a(i) >= a(n))))) ;
22              \forall C i. {
23              x(i)' = v(i),
24              v(i)' = a(i)}))))*
25      \] \forall C i.
26           \forall C j.
27             (    e(i) = 1 & e(j) = 1 & !i = j
28              -> !x(i) = x(j))
29 }
```

Listing 7.1: Beispiel einer QdL-Formel

8 Weitere Beispiele

8.1 Betragsfunktion

Die Betragsfunktion (s. Gleichung 8.1) lässt sich in KeYmaera als Vorbedingung (s. Gleichung 8.2) definieren. Um zu testen, ob die Betragsfunktion korrekt funktioniert, habe ich versucht mit KeYmaera zu beweisen, dass „$Betrag(-2) = 2$“ gilt (s. Listing A.4). KeYmaera konnte einen Beweis nicht automatisiert finden. Der Beweis kann aber manuell wie in Abbildung 8.1 gezeigt mit KeYmaera durchgeführt werden.

Der Trick ist, dass in der Vorbedingung die quantisierte Variable x durch -2 ersetzt wird. Anschließend ist der Beweis sehr einfach (auch automatisiert) zu finden.

$$Betrag(x) = \begin{cases} x, & x \geq 0 \\ -x, & x < 0 \end{cases} \tag{8.1}$$

$$\forall R\, x : (((x < 0) \rightarrow Betrag(x) = -x) \wedge ((x \geq 0) \rightarrow Betrag(x) = x)) \tag{8.2}$$

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{*}{(Betrag(-2) = --2), (-2 \geq 0 \rightarrow Betrag(-2) = -2) \vdash Betrag(-2) = 2}\, ax}{(true \rightarrow Betrag(-2) = --2), (-2 \geq 0 \rightarrow Betrag(-2) = -2) \vdash Betrag(-2) = 2}\, simplify}{(-2 < 0 \rightarrow Betrag(-2) = --2), (-2 \geq 0 \rightarrow Betrag(-2) = -2) \vdash Betrag(-2) = 2}\, simplify}{\forall Rx : ((x < 0 \rightarrow Betrag(x) = -x) \wedge (x \geq 0 \rightarrow Betrag(x) = x)) \vdash Betrag(-2) = 2}\, \wedge l}{\vdash \forall Rx : ((x < 0 \rightarrow Betrag(x) = -x) \wedge (x \geq 0 \rightarrow Betrag(x) = x)) \rightarrow Betrag(-2) = 2}\, \rightarrow r$$

Abbildung 8.1: Beweis für $Betrag(-2) = 2$

$$
\begin{aligned}
PID \equiv\ & (ctrl; dynamic)^* \\
ctrl \equiv\ & err_0 := v_{soll} - v \\
& a := K_P * err_0 + \\
& \qquad K_I * eps * (err_0 + err_1)/2 + \\
& \qquad K_D * (err_0 - 2 * err_1 + err_2)/eps; \\
& err_2 := err_1; \\
& err_1 := err_0 \\
dynamic \equiv\ & t := 0; \{v' = a, t' = 1, t \leq \epsilon, v \geq 0\}
\end{aligned}
$$

Abbildung 8.2: digitaler PID-Regler

$$
\begin{aligned}
PID_{gt} \equiv\ & t_{global} := 0; \\
& (ctrl; dynamic)^* \\
ctrl \equiv\ & err_0 := v_{soll} - v \\
& a := K_P * err_0 + \\
& \qquad K_I * eps * (err_0 + err_1)/2 + \\
& \qquad K_D * (err_0 - 2 * err_1 + err_2)/eps; \\
& err_2 := err_1; \\
& err_1 := err_0 \\
dynamic \equiv\ & t := 0; \{v' = a, t' = 1, t'_{global} = 1, t \leq \epsilon, v \geq 0\}
\end{aligned}
$$

Abbildung 8.3: digitaler PID-Regler mit globaler Zeitvariable

8.2 Stabilität eines digitalen PID-Reglers

Im Folgenden wird gezeigt, wie Formeln zur Analyse von digitalen PID-Reglern aufgestellt werden können. Zur Stabilitätsanalyse muss gewährleistet werden, dass die Reglerabweichung nach einer gewissen Zeit unterhalb einer bestimmten Grenze liegt und dann auch dauerhaft unter dieser Grenze bleibt. In Abbildung 8.2 ist das Hybride Programm eines digitalen PID-Reglers zu sehen. Um das Verhalten des Reglers nach einer bestimmten Zeit zu untersuchen, kann eine globale Zeitvariable verwendet werden, die bei jeder kontinuierlichen Zustandsänderung mitläuft. Ein dafür abgewandeltes Hybrides Programm ist in Abbildung 8.3 zu sehen.

Zur Verifikation der Stabilität kann nun die Formel 8.3 verwendet werden.

$$
\phi \rightarrow [PID_{gt}](t_{global} > T \rightarrow |err_0| \leq E) \tag{8.3}
$$

Zum Analysieren des Überschwingverhaltens eines PID-Reglers wird keine globale Zeitvariable benötigt. Ziel ist es zu zeigen, dass die Stellgröße dauerhaft unterhalb einer bestimmten Grenze bleibt. Zur Verifikation kann die Formel 8.4 verwendet wer-

den. In diesem Beispiel wird zur Vereinfachung angenommen, dass der Soll-Wert v_{soll} größer als der aktuelle Wert v ist.

$$\phi \rightarrow [PID]v \leq v_{max} \tag{8.4}$$

$$\phi \equiv v_{soll} > v \wedge v_{max} = v + 1.1(v_{soll} - v) \tag{8.5}$$

In Listing A.5 ist der Quelltext dieses Beispiels mit weiteren Vereinfachungen angegeben. Es handelt sich hierbei um einen Regler, der zur Berechnung der Stellgröße nur noch den proportionalen Anteil verwendet. Dieses Listing kann manuelle mit 30 Schritten bewiesen werden. Als Schleifen-Invariante ist $v \leq v_{max}$ zu wählen. Der Beweis ist auf der beigelegten CD zu finden. Beweise für komplexere Beispiele konnte ich mit KeYmaera nicht finden.

9 Zusammenfassung und Ausblick

Abschließend werden zuerst die Ergebnisse dieser Arbeit und dann in einem Ausblick die während der Arbeit aufgefallen Probleme zusammengefasst.

9.1 Zusammenfassung

Ein Ziel dieser Arbeit war zu zeigen, wie die Verifikation Hybrider Systeme mit der Software KeYmaera durchgeführt werden kann. Hierzu wurde eine Einführung in die Hybriden Systeme gegeben und gezeigt, wie sie mit Hybriden Programmen beschrieben werden können. Zur Beschreibung Hybrider Programme stehen in KeYmaera die Logiken d$\mathscr{L}$, DAL und dTL zur Verfügung. Die Syntax dieser Logiken und die Semantik der einzelnen Operatoren wurden eingeführt und mit kurzen Beispielen verständlich gemacht. Ebenso würde erläutert, wie eine Quelldatei aufgebaut ist und welche Besonderheiten bei der Syntax zu berücksichtigen sind. Für die Verifikation eines Hybriden Programms wird ein Sequenzenkalkül verwendet. Die wichtigsten Regeln dieses Kalküls wurden aufgeführt und deren Bedeutung erläutert. Dadurch wurde verständlich, wie die Software mit Modaloperatoren, Quantoren und nicht lösbaren Differentialgleichungen umgeht. Ebenso wurde die Suchstrategie KeYmaeras erläutert.

Ein weiteres Ziel dieser Arbeit war zu prüfen, ob durch den Einsatz von KeYmaera die Korrektheit praxisrelevanter Systeme nachgewiesen werden kann. Dazu wurden zunächst die vereinfachten Modelle eines Notbremsassistenten und eines intelligenten Tempomaten analysiert. Es wurde gezeigt, dass das Modell des Notbremsassistenten eine unerwartete Eigenschaft besitzt und eine Verbesserung vorgeschlagen. Ebenso wurde auf die für das Modell des intelligenten Tempomaten gemachten Annahmen hingewiesen, die so in der Realität nicht zutreffen. Bereits bei den vereinfachten Modellen war die vollständig automatisierte Verifikation nur bedingt möglich. Dadurch war häufig der manuelle Eingriff durch den Anwender notwendig.

Bei dem Versuch, die Kollisionsfreiheit zwischen Quadrocoptern nachzuweisen, stellten sich mehrere Probleme heraus. Zum einen ist es deutlich schwieriger Sicherheitsbedingungen für ein mehrdimensionales Modell aufzustellen, zum anderen ist die Bewegungsgleichung eines Quadrocopters so komplex, dass starke Vereinfachungen

notwendig sind. Zur Lösung dieser Probleme habe ich verschiedene Ansätze gezeigt.

9.2 Ausblick

Bei der Durchführung dieser Arbeit sind einige Aspekte aufgefallen, an denen weiter geforscht werden sollte, um die Verifikation von Hybriden Systemen zu optimieren.

Ein Großteil der Rechenzeit bei einer Verifikation wird für Quantorenelimination verwendet. Dies legt nahe, die Quantorenelimination zu optimieren oder sie weniger aber dafür gezielter einzusetzen. Hierzu müsste erkannt werden, ob ein Aufruf der Quantorenelimination sinnvoll ist und welche Formeln als Parameter übergeben werden sollen.

Ein alternativer Ansatz zur Reduzierung der Rechenzeit wäre die Optimierung der Suchstrategie, die von KeYmaera verwendet wird. Dies könnte erreicht werden, wenn die Strategie an das gegebene Problem angepasst wird. Ein möglicher Ansatz wäre, Strategien für bestimmte Klassen von Hybriden Systeme zu entwickeln und in einem Beweisschritt nicht nur eine Regel, sondern eine fest definierte Folge von Regeln anzuwenden, die für die entsprechende Klasse von Hybriden System als erfolgversprechend eingestuft wird.

Literaturverzeichnis

[1] Weber, D.: Regelungstechnik - Wirkungsweise und Einsatz elektronischer Regler, expert verlag (1993), S. 12-13

[2] Platzer, A.: Logical Analysis of Hybrid Systems - Proving Theorem for Complex Dynamics, Springer-Verlag (2010)

[3] Platzer, A.: KeYmaera: A Hybrid Theorem Prover for Hybrid Systems, http://symbolaris.com/info/KeYmaera.html, April 2012

[5] Müller, O., Stauner, T.: Modelling and Verification using Linear Hybrid Automata - a CaseStudy, Mathematical Modelling of Systems 1996, Vol. 1, No. 1, pp. 000-111

[6] Platzer, A.: Guide for KeYmaera Hybrid Systems Verification Tool, http://symbolaris.com/info/KeYmaera-guide.html, April 2012

[7] Gentzen, G.: Untersuchungen über das logische Schließen, Mathematische Zeitschrift, 39. Band, 1935

[8] Ehrig, H., Mahr, B., Cornelius, F., Große-Rhode, M., Zeitz, P.: Mathematisch-Strukturelle Grundlagen der Informatik (2. Auflage), Springer-Verlag (2001)

[9] Hofmann, M.: Logik für Informatiker (Skript zur Vorlesung), http://www.tcs.ifi.lmu.de/lehre/ss-2012/lds/lds-material/logik-fuer-informatiker-sose-2008, Ludwig-Maximilians-Universität München, Juli 2012

[10] Eibegger, M.: logic rulez - Einführung in die mathematische Logik, http://www.logic-rulez.net/logik/korrekt_vollstaendig.php, September 2012

[11] Noll, T.: Semantics and Verification of Software, Lecture 11: Axiomatic Semantics of WHILE III (Correctness of Hoare Logic), http://www-i2.informatik.rwth-aachen.de/i2/fileadmin/user_upload/documents/SV-

SW11/l11_handout.pdf, RWTH Aachen University, September 2012

[12] Chong, S., Morrisett, G., Myers, A., Necula, G., Rugina, R.: CS 4110 - Programming Languages and Logics, Lecture 9: Relative Completeness, http://www.cs.cornell.edu/Courses/cs4110/2010fa/lectures/lecture09.pdf, Cornell University, Department of Computer Science, September 2012

[13] Günther, F.: Algorithmische Aspekte der Quantorenelimination auf den reellen Zahlen, https://www3.mathematik.tu-darmstadt.de/evs/e/32.html?evsver=927&evsdir=965&evsfile=Guenther.pdf, Technische Universität Darmstadt, August 2012

[14] Dolzmann, A., Sturm, T.: REDLOG Computer Algebra Meets Computer Logic, Universität Passau, Fakultät für Mathematik und Informatik, September 1996

[15] Brown, C.: QEPCAD B: A program for computing with semi-algebraic sets using CADs, United States Naval Academy, Department of Computer Science, Dezember 2003

[16] Hörmander, L.: The Analysis of Linear Partial Differential Operators II - Differential Operator with Contant Coefficients, Springer-Verlag (2005)

[17] Wolfram Research, Inc: http://functions.wolfram.com/Constants/ Mai 2012

[18] Loos, S., Platzer, A., Nistor, L.: Adaptive Cruise Control: Hybrid, Distributed, and Now Formally Verified, Carnegie Mellon University Pittsburgh, School of Computer Science, Juni 2011

[19] Aréchiga, N., Loos, S., Platzer, A., Krogh, B.: Using Theorem Provers to Guarantee Closed-Loop System Properties, Department of Electrical and Computer Engineering, Department of Computer Science, Carnegie Mellon University Pittsburgh

[20] Platzer, A.: Adaptive Cruise Control: Hybrid, Distributed, and Now Formally Verified, http://www.ls.cs.cmu.edu/dccs/, August 2012

[21] Wendel, J.: Integrierte Navigationssysteme - Sensordatenfusion, GPS und Inertiale Navigation, 2. überarbeitete Auflage, Oldenbourg Verlag München (2011)

[22] Platzer, A.: Quantified Differential Dynamic Logic for Distributed Hybrid Systems, Carnegie Mellon University Pittsburgh, Computer Science Department, Springer-Verlag (2010)

[23] Beckert, B., Hähnle, R., Schmitt, P.: Verification of Object-Oriented Software - The KeY Approach, Springer-Verlag (2007)

Abbildungsverzeichnis

Tabellenverzeichnis

A Quelltexte

```
\functions{
  R B;
  R A;
  R eps;
  R q;
}
\problem{
  \[ R p, v, a, critDist, t \] (
    v^2<=2*B*(q-p) & B>0 & A>=0 & eps>0 & p<=q
    ->
    \[(
        critDist := v*eps + 0.5*A*eps^2 + (v+A*eps)^2/(2*B);

        if (q-p > critDist) then              /* ctrl */
          a := *; ?(-B<=a & a<=A)
        else
          a := -B
        fi;

        t:=0;                                  /* drive */
        {p'=v, v'=a, t'=1, t<=eps, v>=0}
      )*
    \] p<=q
  )
}
```

Listing A.1: Notbremsassistent

```
\functions{
  R B;
  R A;
  R eps;
  R q;
}
\problem{
  \[ R p, v, a, t \] (
   v^2<=2*B*(q-p) & B>0 & A>=0 & eps>0 & p<=q
    ->
    \[(
        if (B^2*eps^2-8*B*p+8*B*q-4*B*eps*v < 0) then
            a := -B
        else
            a := *;
            ?(-B<=a & a<=(-B*eps-2*v)/(2*eps)+1/2*((B^2*eps^2-8*B*p+8*B*q-4*
                B*eps*v)/(eps^2))^(1/2) & a<=A)
        fi;
        t:=0;       /* drive */
        {p'=v, v'=a, t'=1, t<=eps, v>=0}
      )*
    \] p<=q
  )
}
```

Listing A.2: Optimierter Notbremsassistent

```
\functions{
  R eps;
  R V_min;
  R V_max;
  R A_min;
  R A_max;
}
\problem{
  \[ R p_0, v_0, a_0, p_1, v_1, a_1, t, tt \] (
    V_min<=0 & V_max>0 & A_min<0 & A_max>0 & eps>0 &
    V_min<=v_0 & v_0<=V_max & V_min<=v_1 & v_1<=V_max &
    p_0<=p_1 & v_0=0 & v_1=0
    ->
    \[
      (
        a_1 := *; ?(A_min<=a_1 & a_1<=A_max);

        tt := (v_0 + A_max * eps - V_min) / (-A_min) + eps;
        if (p_1 + v_1 * tt + 0.5 * A_min * tt^2 >=
            p_0 + v_0 * eps + 0.5 * A_max * eps^2 +
            (v_0 + A_max * eps)*(tt - eps) +
            0.5 * A_min * (tt - eps)^2) then
          a_0 := *; ?(A_min<=a_0 & a_0<=A_max)
        else
          a_0 := A_min
        fi;

        t := 0;
        { p_0'=v_0, v_0'=a_0,
          p_1'=v_1, v_1'=a_1,
          t'=1,
          V_min<=v_0 & v_0<=V_max &
          V_min<=v_1 & v_1<=V_max &
          t<=eps
        }
      )*
    \] (p_0<=p_1)
  )
}
```

Listing A.3: Intelligenter Tempomat

```
\functions{
  R Betrag(R);
}
\problem{
  (\forall R x; (((x<0) -> Betrag(x) = -x) & ((x>=0) -> Betrag(x) = x)))
  ->
  Betrag(-2) = 2
}
```

Listing A.4: Formel für den Beweis von $Betrag(-2) = 2$

```
\functions{
  R eps;
  R v_soll;
  R K_P;
  R v_max;
}

\problem{
  \[ R a, t, v \] (
      eps=1 & K_P>0 & K_P<=1 &
      v_soll > v & v_max=v+(v_soll-v)*1.1
    ->
      \[
        (
          a :=  K_P * (v_soll-v);
          t := 0;
          {
            v'=a,
            t'=1,
            t<=eps
          }
        )*
      \] v <= v_max
  )
}
```

Listing A.5: Überschwingverhalten eines digitialen Reglers

```
\functions{
    R t_0;
    R t_1;
}
\problem{
    \[ R    x_00, x_01, x_02, x_03,
            x_10, x_11, x_12, x_13  \] (
        \<
            x_03 := *;
            x_13 := *;
            {
                x_00'=x_01, x_10'=x_11,
                x_01'=x_02, x_11'=x_12,
                x_02'=x_03, x_12'=x_13
            }
        \>
        ( x_00=t_0 & x_10=t_1 )
    )
}
```

Listing A.6: Erreichbarkeitsanalyse im 2-dimensionalen Raum für ein System mit 3-fach integralem Verhalten

```
\functions{
    R q_0;
    R q_1;
    R C;
    R eps;
}
\problem{
    \[ R  x_00, x_01, x_02,
          x_10, x_11, x_12,
          t \] (
        C>0 & eps>0 &
        ((x_00-q_0)^2 + (x_10-q_1)^2 > C^2)
        ->
        \[
              x_02 := *;
              x_12 := *;

              (?(\forall R tau; ((0<=tau & tau<=eps) ->
                (((x_00 + x_01*tau + 0.5*x_02*tau^2 - q_0)^2 +
                  (x_10 + x_11*tau + 0.5*x_12*tau^2 - q_1)^2 > C^2)))));

              t := 0;
              {
                  x_00'=x_01, x_01'=x_02,
                  x_10'=x_11, x_11'=x_12,
                  t'=1, t<=eps
              }
        \]
        ((x_00-q_0)^2 + (x_10-q_1)^2 > C^2)
    )
}
```

Listing A.7: Sicherheitsanalyse im 2-dimensionalen Raum für ein System mit 2-fach integralem Verhalten

B Systembeschreibung

Für alle in dieser Arbeit durchgeführten Verifikationen habe ich einen Laptop mit den in der Tabelle B.1 gezeigten Eigenschaften und Software-Versionen verwendet.

Betriebssystem	Windows 7 Home Premium 64 Bit
Prozessor	AMD A8-3500M mit 1.5 - 2.4 GHz
Arbeitsspeicher	4 GB
KeYmaera-Version	2.0
Mathematica-Version	8.0.4.0

Tabelle B.1: Eigenschaften des für die Verifikationen verwendeten Systems

B.1 KeYmaera Konfiguration

Die in den Abbildungen B.1 bis B.4 gezeigten Einstellungen habe ich, wenn nicht anders angegeben, für alle in dieser Arbeit gezeigten Verifikationen mit KeYmaera verwendet.

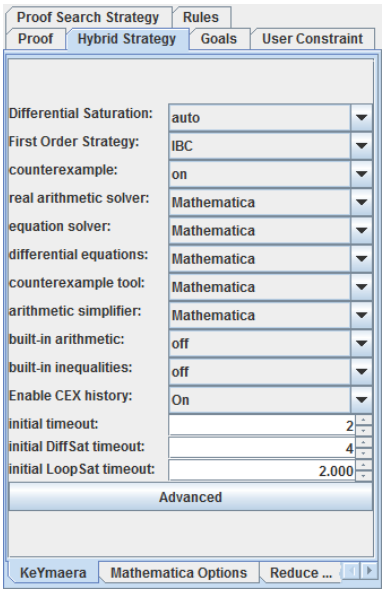

Abbildung B.1: Hybrid Strategy - KeYmaera

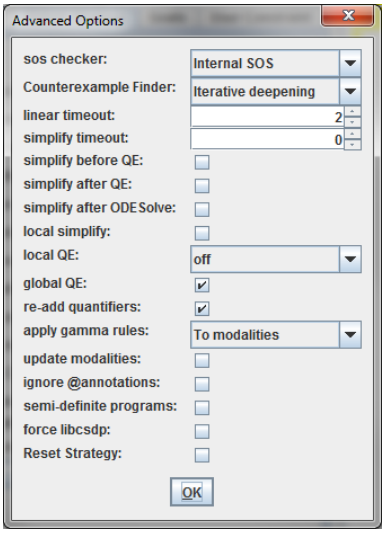

Abbildung B.2: Hybrid Strategy - KeYmaera - Advanced

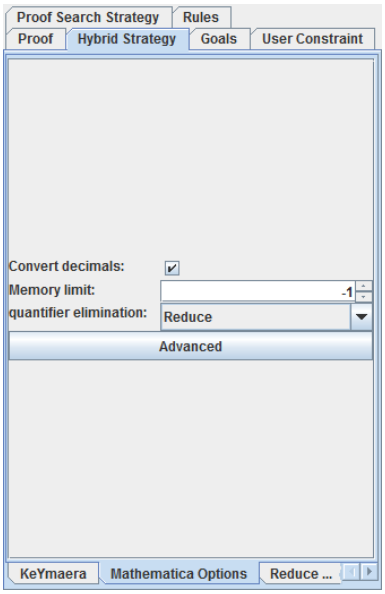

Abbildung B.3: Hybrid Strategy - Mathematica Options

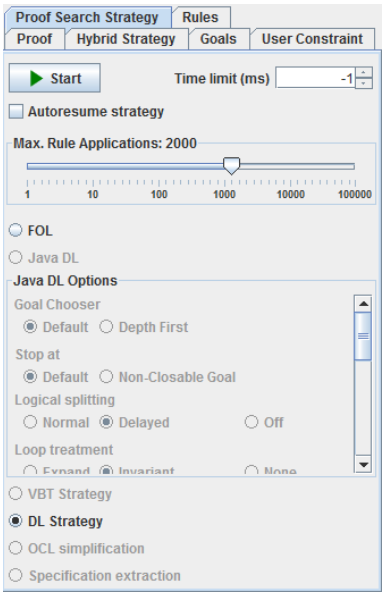

Abbildung B.4: Proof Search Strategy

C CD