UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

Laufzeitverifikation mit einer vierwertigen Semantik für die Reguläre Lineare Temporale Logik

Runtime-Verification with a four-valued semantics for the Regular Linear Temporal Logic

Masterarbeit

im Rahmen des Studiengangs
Informatik
der Universität zu Lübeck

vorgelegt von
Christofer Krüger

ausgegeben und betreut von
Prof. Dr. Leucker

mit Unterstützung von
Torben Scheffel

Lübeck, den 17. Dezember 2014

Abstract

In dieser Arbeit wird eine vierwertige Semantik für die Reguläre Lineare Temporale Logik (RLTL) definiert. Im Vergleich zu LTL können in dieser Logik mehr Eigenschaften spezifiziert werden. Die Semantik beschreibt eine Auswertung von endlichen Worten, wobei berücksichtigt wird, dass die Worte um weitere Zeichen verlängert werden können. Darüber hinaus wird ein Verfahren erläutert, mit dem zur Laufzeitverifikation Mealy-Maschinen als Monitore generiert werden können, deren Ausgaben den von der Semantik festgelegten Werten entsprechen. Zusätzlich werden die Implementierung sowie einige Optimierungen des Prozederes analysiert.

In this thesis a four-valued semantics is defined for the Regular Linear Temporal Logic (RLTL). As compared to LTL more properties can be expressed in this logic. The semantics describes an evaluation of finite words where it needs to be considered that these words can be expanded by additional symbols. Furthermore for Runtime-Verification a process gets explained which allows the generation of Mealy-automata which output values defined by the semantics. Additionally the implementation of the procedure gets analyzed along with a couple of optimizations.

Erklärung

Ich versichere an Eides statt, die vorliegende Arbeit selbstständig und nur unter Benutzung der angegebenen Hilfsmittel angefertigt zu haben.

Lübeck, 17. Dezember 2014

Inhaltsverzeichnis

1 Einleitung

Im Kontext der Systemverifikation können unter anderem temporallogische Formeln zur Spezifikation von Anforderungen verwendet werden, die ein System einhalten muss. Die Techniken der Runtime-Verification ermöglichen es, zur Laufzeit des überwachten Systems zu überprüfen, ob dieses die entsprechenden Anforderungen erfüllt. Da die Anzahl der Systemzustände während der Ausführung stets endlich ist, muss für ein endliches Wort überprüft werden, ob es ein Modell für eine Formel ist, wobei das Wort durch weitere Aktionen des überwachten Systems um zusätzliche Zeichen verlängert werden kann. Durch die Definition von Semantiken für eine Logik kann festgelegt werden, zu welchen Wahrheitswerten ein Wort zu einer gegebenen Formel auswertet. Damit ein eindeutiges Urteil darüber möglich ist, inwiefern ein Wort, das um beliebige Zeichen verlängert werden kann, die Formel erfüllt, sind zwei Wahrheitswerte im Allgemeinen nicht ausreichend; stattdessen können Semantiken definiert werden, die zusätzlich zur Information, ob das Wort mit den bislang vorhandenen Zeichen ein Modell für eine Formel ist, ebenfalls angeben, ob diese Entscheidung endgültig ist oder noch verändert werden kann, wenn das Wort durch weitere Aktionen des Systems um zusätzliche Zeichen verlängert wird. Da somit insgesamt vier Auswertungen möglich sind, für beide Kriterien ist sowohl ein positives als auch ein negatives Urteil möglich, kommen Semantiken mit ebenso vielen Wahrheitswerten zum Einsatz. Die eigentliche Überprüfung zur Laufzeit wird von Automaten vorgenommen, die als *Monitor* bezeichnet werden. Diese werden aus einer temporallogischen Formel generiert, lesen die Zeichen des Wortes ein und geben die von der Semantik definierten Wahrheitswerte aus.

Für Formeln in der Linearen Temporalen Logik (LTL) [Pnueli, 1977] existiert bereits ein Verfahren zur Konstruktion von Automaten, deren Ausgaben den Wahrheitswerten entsprechen, die in der Definition einer vierwertigen Semantik auf endlichen Worten festgelegt sind [Leucker, 2012]. Der Vorteil dieser Methode liegt darin, dass sich eine Formel direkt ohne die Durchführung eines Erfüllbarkeitstests in eine alternierende Mealy-Maschine transformieren lässt, wobei die Zustände des Automaten Teilformeln der ursprünglichen LTL-Formel repräsentieren.

Die Ausdrucksmächtigkeit von LTL ist allerdings auf die der sternfreien ω-regulären Ausdrücken begrenzt [Vardi, 2008], sodass Eigenschaften existieren, die sich nicht als LTL-Formel ausdrücken lassen. Mit der Regulären Linearen Temporalen Logik (RLTL) [Leucker und Sánchez, 2007] wurde eine Logik entwickelt, welche die Ausdrucksmäch-

tigkeit auf die der ω-regulären Ausdrücke erhöht, indem temporallogische Formeln mit regulären Ausdrücken kombiniert werden. Bislang wird die Modellprüfung in dieser Logik nur mit unendlichen Worten durchgeführt, wobei zwei Wahrheitswerte ausreichend sind, da direkt ein eindeutiges Urteil möglich ist.

Das Ziel dieser Arbeit ist es, ein Verfahren für die Reguläre Lineare Temporale Logik zu konzipieren und implementieren, welches endliche Worte vierwertig auswertet, sodass der von LTL bekannte Ansatz, bei dem Monitore zur Laufzeitverifikation generiert werden, auch in der Logik mit der vergrößerten Ausdrucksmächtigkeit funktioniert. In dieser Arbeit wird daher eine vierwertige Semantik auf endlichen Worten entwickelt, die für eine RLTL-Formel und ein endliches Wort sowohl entscheidet, ob das Wort ein Modell für die Formel ist, als auch angibt, ob nach einer Verlängerung des Wortes die Möglichkeit besteht, dass eine andere Auswertung erfolgt. Außerdem wird ein Verfahren beschrieben, mit dem es möglich ist, aus RLTL-Formeln alternierende Mealy-Maschinen zu generieren, die dazu verwendet werden können, ein System zur Laufzeit zu überwachen, indem sie als Monitor eingesetzt werden, der die von der Semantik definierten Werte ausgibt. Des Weiteren wird das in dieser Arbeit vorgestellte Verfahren in der Automatenbibliothek *Rltl-Conv*, die an der Universität zu Lübeck am Institut für Softwaretechnik und Programmiersprachen (ISP) entwickelt wird und unter anderem bereits Monitore konstruieren kann, die LTL-Formeln vierwertig auswerten, implementiert.

Die Arbeit ist dazu wie folgt gegliedert: In Kapitel 2 werden neben der Einführung der verwendeten Notation die Syntax und Semantik der Logiken LTL und RLTL beschrieben, wobei zunächst unendlich lange Worte betrachtet werden. Zusätzlich wird ein Überblick über andere Temporallogiken gegeben. Anschließend werden Begrifflichkeiten und Einsatzgebiete der Laufzeitverifikation erläutert, woraufhin die bereits existierenden verschiedenwertigen Semantiken für LTL beschrieben und miteinander verglichen werden, wobei auch die Vorteile einer vierwertigen Semantik aufgezeigt werden. Unterschiede zwischen den Semantiken für endliche und unendliche Worte sind ebenso Inhalt des Kapitels, wie eine Abgrenzung zum Model-Checking, einer Technik, die ebenfalls zur Verifikation eingesetzt werden kann. Die vierwertige Semantik für RLTL auf endlichen Worten wird in Kapitel 3 definiert. Damit dies überhaupt möglich ist, ist es erforderlich, einen weiteren Operator für die reguläre Linearzeitlogik zu definieren. Die Begründung, warum für die Definition der Semantik die Variante von RLTL zum Einsatz kommt, in der Propositionen in der Syntax definiert sind, wird ebenfalls aufgeführt. Zum Abschluss des Kapitels folgt die Argumentation dafür, dass die Definition die Worte gemäß dem Prinzip einer vierwertigen Semantik korrekt auswertet. In Kapitel 4 werden Transitionsfunktionen definiert, mit denen sich zu einer RLTL-Formel eine alternierende Mealy-Maschine generieren lässt, deren Ausgaben die von der Semantik festgelegten Wahrheitswerte sind. Nach dem Prinzip des Formel-rewritings werden dabei für die möglichen Eingabesymbole positive

boolesche Kombinationen von Formeln berechnet, die als Nachfolgezustände verwendet werden. Zusätzlich wird gezeigt, dass das Verfahren korrekt ist und dass der Automat bezüglich der über die Anzahl der Operatoren definierte Länge der Formel aus linear vielen Zuständen besteht. Details zur Implementierung der Monitorgenerierung in der Automatenbibliothek RltlConv, die in der Programmiersprache *Scala* geschrieben wurde, sowie mögliche Optimierungen zur Reduzierung der Laufzeit des Algorithmus sind Inhalt von Kapitel 5. Darüber hinaus werden als Beispiel Automaten aufgeführt, die mittels des in dieser Arbeit beschriebenen Verfahrens aus RLTL-Formeln generiert wurden und somit als Monitor eingesetzt werden können. Zusätzlich werden Analysen durchgeführt, welche die unterschiedlichen implementierten Varianten miteinander vergleichen, sowie die Zeitdauer der Generierung von alternierenden und deterministischen Monitoren für LTL- und RLTL-Formeln gegenüberstellen. Kapitel 6 fasst die Arbeit zusammen und gibt einen Ausblick auf mögliche zukünftige Arbeiten.

2 Grundlagen

Dieses Kapitel gibt einen Überblick über verschiedene temporale Logiken, enthält die bereits existierenden Definitionen verschiedener Semantiken, auf die sich in dieser Arbeit bezogen wird, und beschreibt die grundlegenden Prinzipien der Runtime-Verification, wo derartige Logiken zur Programmverifikation verwendet werden. Dabei wird zwischen der Analyse von unendlichen und endlichen Worten distinguiert und hervorgehoben, welche Eigenschaften Semantiken, die sich auf endliche Worte beziehen, erfüllen müssen.

Abschnitt 2.1 beschreibt die in dieser Arbeit verwendete Notation. Danach werden in Abschnitt 2.2 Syntax und Semantik von LTL definiert, bevor in Abschnitt 2.3 mit RLTL eine Logik betrachtet wird, die LTL derart erweitert, dass sich zusätzliche Eigenschaften ausdrücken lassen. Auf die Charakteristika von RLTL wird besonders detailliert eingegangen, da diese auch für die Definition der vierwertigen RLTL-Semantik auf endlichen Worten in Kapitel 3 relevant sind. Zusätzlich werden in Abschnitt 2.4 kurz andere Ansätze vorgestellt, die ebenfalls Logiken definieren, welche eine höhere Ausdrucksmächtigkeit als LTL haben, teilweise aber gegenüber RLTL auch Nachteile in Bezug auf die Einsetzbarkeit in der Praxis aufweisen. Anschließend werden in Abschnitt 2.5 die Grundlagen, Begrifflichkeiten und Einsatzgebiete der Runtime-Verification beschrieben. Die Definition des Begriffs der Monitorbarkeit sowie die Definitionen der zwei-, drei- und vierwertigen Semantiken, die für LTL bereits existieren, folgen in Abschnitt 2.6. Darüber hinaus werden diese miteinander verglichen und die Begründung dafür geliefert, warum letztere für die Analyse von endlichen Worten mit Monitoren bei der online-Runtime-Verification am geeignetsten sind.

2.1 Notation

In diesem Abschnitt wird kurz auf die Notation eingegangen, die in dieser Arbeit für die Definitionen der Logiken und verschiedenwertigen Semantiken verwendet wird. Damit stets deutlich wird, ob endliche oder unendliche Worte betrachtet werden, kommen unterschiedliche Notationen zum Einsatz.
Eine Menge an Symbolen, *Alphabet* genannt, wird mit Σ bezeichnet, $w \in \Sigma^*$ ist ein *endliches Wort*, $w \in \Sigma^+$ ein endliches, nicht leeres Wort und $w \in \Sigma^\omega$ ein *unendliches Wort* über dem Alphabet Σ. Für die Länge eines Wortes wird die Notation $|w|$ verwendet, das leere Wort wird wie üblich mit ε bezeichnet. Eine *Position* in einem Wort $w = w_0, w_1, \cdots, w_n$ ist eine

natürliche Zahl. Dabei bezeichnet w_i die i-te Position im Wort, der Suffix eines Wortes ab Position i wird mit w^i beschrieben, ${}^iw = w_0, \cdots, w_{i-1}$ sei die Notation für den Präfix eines Wortes bis zur Position $i-1$. Dementsprechend ist ${}^jw^i = w_i, w_{i+1}, \cdots, w_{j-2}, w_{j-1}$ das *Teilwort* von w von Position i bis $j-1$. Falls $j \geq |w|$ oder $i = 0$ gelte ${}^iw = w^j = \varepsilon$, ebenso sei ${}^jw^i = \varepsilon$ für $j \leq i$. Im Gegensatz dazu beschreibt (w, i, j) das *Segment* des unendlichen Wortes $w \in \Sigma^\omega$ von Position i bis zu Position j; (w, i) sei das Wort ab Position i. Diese beiden Notationen bezeichnen also das gesamte Wort zusammen mit Markierungen. Eine *Sequenz* von Positionen wird mit $\langle i_0, i_1, \cdots, i_n \rangle$ bezeichnet, $\langle i_0, i_1, \cdots \rangle$ steht dabei für eine unendliche Sequenz.
Die Relationen $\cdot \models \cdot$ und $\cdot \models_{\text{RE}} \cdot$ geben an, dass ein Wort ein Modell für eine temporallogische Formel oder einen regulären Ausdruck ist. Die Sprache einer Formel φ oder eines regulären Ausdrucks α wird mit $L(\varphi)$ beziehungsweise $L(\alpha)$ bezeichnet. Zwei Formeln φ und ψ sind zueinander äquivalent, die Notation hierfür ist $\varphi \equiv \psi$, wenn sie die gleiche Sprache akzeptieren.

2.2 LTL

Als Spezifikationssprache für sequentielle und parallele Programme wurde die *Linear Temporal Logic (LTL)* [Pnueli, 1977] entwickelt, um die Verifikation von temporalen Eigenschaften zu ermöglichen. Bei nebenläufigen Programmen kann beispielsweise spezifiziert werden, dass kein Deadlock auftritt. Die Logik enthält neben den booleschen Operatoren und Propositionen eine Reihe von temporalen Operatoren, wie den *Until*-Operator [Gabbay et al., 1980]. Die Operatoren in der ursprünglichen Definition von LTL können nur zukünftige Eigenschaften beschreiben. Die Logik wurde später auch noch um Vergangenheits-Operatoren erweitert, mit denen sich somit auch Formeln angeben lassen, die Anforderungen an das vergangene Verhalten des Systems stellen [Lichtenstein et al., 1985]. Da sich beide Varianten nicht in der Mächtigkeit unterscheiden, finden für die Definition von Syntax und Semantik von LTL lediglich die Zukunfts-Operatoren Verwendung.

Definition 2.1 (Syntax für LTL)**.** *Die Syntax für LTL-Formeln ist durch folgende Grammatik definiert:*

$$\varphi ::= p \quad | \quad \neg\varphi \quad | \quad \varphi \vee \varphi \quad | \quad X\varphi \quad | \quad \varphi U \varphi \quad | \quad \varphi W \varphi$$

Definition 2.2 (Semantik für LTL auf unendlichen Worten)**.** *Seien φ und ψ LTL-Formeln und $w \in \Sigma^\omega$ ein unendliches Wort. Die Semantik für LTL ist wie folgt definiert.*

- $(w, i) \models true$ *gilt immer*

- $(w, i) \models p \quad \iff p \in w_0$
- $(w, i) \models \neg\varphi \quad \iff (w, i) \not\models \varphi$
- $(w, i) \models \varphi \vee \psi \quad \iff (w, i) \models \varphi$ *oder* $(w, i) \models \psi$
- $(w, i) \models X\varphi \quad \iff (w, i + 1) \models \varphi$
- $(w, i) \models \varphi U \psi \quad \iff \exists k : (w, k) \models \psi$ *und* $\forall l(0 \leq l < k) : (w, l) \models \varphi$
- $(w, i) \models \varphi W \psi \quad \iff \varphi U \psi$ *oder* $\forall k : (w, k) \models \varphi$

Der *Next*-Operator, für den das Symbol X verwendet wird, fordert, dass eine Eigenschaft an der nächsten Position gelten muss. Der *Until*-Operator, für den in der Formel ein U verwendet wird, verlangt, dass ψ irgendwann gelten muss und zuvor in jedem Zeitschritt φ gegolten hat. Der *Weak-Until*-Operator erlaubt zusätzlich, dass die Eigenschaft φ unendlich lange an jeder Position gelten kann. Für diesen Operator wird das Symbol W verwendet.
Es ist üblich, weitere temporale Operatoren einzuführen, die sich jedoch auch mit den übrigen Operatoren beschreiben lassen. Ein Beispiel ist der *finally*-Operator $F\varphi$, der fordert, dass die Eigenschaft φ irgendwann erfüllt sein muss. Eine äquivalente Formel ist $trueU\varphi$. Der zum finally-Operator duale *globally*-Operator, der mit einem G bezeichnet wird, verlangt dementsprechend, dass die Eigenschaft an jeder Position erfüllt sein muss. Außerdem gibt es noch die Operatoren, die zu den Until-Operatoren dual sind, den *Release*-Operator, für den ein R verwendet wird, und den *strong-Release*-Operator, der mit dem Symbol S kenntlich gemacht wird. Für diese Operatoren gelten die folgenden Äquivalenzen:

$$\neg F\varphi \equiv G\neg\varphi \qquad \neg G\varphi \equiv F\neg\varphi$$
$$\neg(\varphi U\psi) \equiv \neg\varphi R\neg\psi \qquad \neg(\varphi R\psi) \equiv \neg\varphi U\neg\psi$$
$$\neg(\varphi W\psi) \equiv \neg\varphi S\neg\psi \qquad \neg(\varphi S\psi) \equiv \neg\varphi W\neg\psi$$

2.3 RLTL

Die *Regular Linear Temporal Logik (RLTL)* [Leucker und Sánchez, 2007] ist eine Erweiterung der in dem vorherigen Abschnitt vorgestellten linearen temporalen Logik, welche deren Ausdrucksmächtigkeit vergrößert, indem zusätzlich zu den temporalen Operatoren reguläre Ausdrücke (*regular expressions, RE*) eingeführt werden.
RLTL wurde entwickelt, um Eigenschaften von unendlichen Läufen auszudrücken, damit bei reaktiven Systemen verifiziert werden kann, ob diese die Spezifikation erfüllen. Dabei baut die Logik auf LTL auf und ergänzt reguläre Ausdrücke, sodass im Gegensatz zu LTL

alle ω-regulären Ausdrücke als RLTL-Formel ausdrückbar sind. Dabei ist das Erfüllbarkeitsproblem, wie es auch bereits bei LTL der Fall ist, PSPACE-vollständig. Aufgrund der Eigenschaften von RLTL ist es möglich, sowohl reguläre und ω-reguläre Ausdrücke als auch LTL-Formeln in RLTL-Formeln zu übersetzen.
In [Sánchez und Leucker, 2010] wurde die Logik um Past-Operatoren erweitert. Zu derartigen RLTL-Formeln lassen sich alternierende 2-Wege Paritätsautomaten generieren, sodass genau die Worte, die Modell für eine RLTL-Formel sind, auch von den Automaten akzeptiert werden. Die Größe des Automaten ist linear in der Länge der Formel, da für die regulären Ausdrücke ein endlicher nichtdeterministischer Automat konstruiert wird, was mit einem linearen Overhead möglich ist und pro RLTL-Operator ein weiterer Zustand bei der Konstruktion hinzukommt.

Definition 2.3 (RLTL-Syntax)**.** *Die Syntaxen einer RLTL-Formel φ und eines regulären Ausdrucks (RE) α sind wie folgt definiert:*

- $\alpha ::= \quad \alpha + \alpha \quad | \quad \alpha ; \alpha \quad | \quad \alpha * \alpha \quad | \quad p$
- $\varphi ::= \quad \varnothing \quad | \quad \varphi \vee \varphi \quad | \quad \neg\varphi \quad | \quad \alpha ; \varphi \quad | \quad \varphi \mid \alpha \gg \varphi \quad | \quad \varphi \mid \alpha > \varphi$

Der reguläre Ausdruck kann aus Disjunktionen, Konkatenationen, dem binären Kleene-Operator sowie aus Propositionen bestehen. Dabei dient `true` als Kurzschreibweise für die Menge aller Propositionen. Die Negation ist ebenso wie die Schnittbildung nicht in der Grammatik enthalten, da eine Erweiterung um diese Operatoren dazu führen würde, dass das Erfüllbarkeitsproblem im Falle der Negation nicht elementar entscheidbar [Stockmeyer, 1974] und im Falle der Hinzunahme eines Schnittoperators EXPSPACE-vollständig wäre [Lange, 2007].
RLTL-Formeln können aus dem die leere Sprache repräsentierenden Operator $\varnothing$ und Disjunktionen sowie aus Negationen, dem Sequenz-Operator und den beiden Power-Operatoren bestehen.
Nachdem die Operatoren für RLTL beschrieben wurden, kann nun die entsprechende Semantik für unendliche Worte angegeben werden.

Definition 2.4 (RLTL-Semantik auf unendlichen Worten, [Sánchez und Leucker, 2010])**.** *Seien $w \in \Sigma^\omega$ ein unendliches Wort, $i, j \in \mathbb{N}$ Positionen in w, α und β reguläre Ausdrücke sowie φ und ψ RLTL-Formeln. Dann gelte*

- $(w, i, j) \models_{\text{RE}} p \quad \iff p \in w_0$ *und* $j = i + 1$
- $(w, i, j) \models_{\text{RE}} \alpha + \beta \quad \iff (w, i, j) \models_{\text{RE}} \alpha$ *oder* $(w, i, j) \models_{\text{RE}} \beta$
- $(w, i, j) \models_{\text{RE}} \alpha ; \beta \quad \iff \exists k : (w, i, k) \models_{\text{RE}} \alpha$ *und* $(w, k, j) \models_{\text{RE}} \beta$

- $(w, i, j) \models_{\text{RE}} \alpha * \beta \quad \Longleftrightarrow (w, i, j) \models_{\text{RE}} \beta$ *oder*
  $\exists \langle i_0, i_1, \cdots, i_m \rangle$ *sodass* $\forall k \in \{0, \cdots, m-1\}$:
  $(w, i_k, i_{k+1}) \models_{\text{RE}} \alpha$ *und* $(w, i_m, j) \models_{\text{RE}} \beta$
- $(w, i) \models \varnothing \quad$ *gilt niemals*
- $(w, i) \models \varphi \vee \psi \quad \Longleftrightarrow (w, i) \models \varphi$ *oder* $(w, i) \models \psi$
- $(w, i) \models \neg\varphi \quad \Longleftrightarrow (w, i) \not\models \varphi$
- $(w, i) \models \alpha; \varphi \quad \Longleftrightarrow \exists k : (w, i, k) \models_{\text{RE}} \alpha$ *und* $(w, k) \models \varphi$
- $(w, i) \models \varphi \mid \alpha \gg \psi \quad \Longleftrightarrow (w, i) \models \psi$ *oder*
  $\exists \langle i_0, i_1, \cdots, i_m \rangle$ *sodass* $(w, i_m) \models \psi$ *und*
  $\forall 0 \leq k < m : (w, i_k, i_{k+1}) \models_{\text{RE}} \alpha$ *und* $(w, i_k) \models \varphi$
- $(w, i) \models \varphi \mid \alpha > \psi \quad \Longleftrightarrow (w, i) \models x \mid \alpha \gg y$ *oder*
  $\exists \langle i_0, i_1, \cdots \rangle$ *sodass*
  $\forall k \geq 0 : (w, i_k, i_{k+1}) \models_{\text{RE}} \alpha$ *und* $(w, i_k) \models \varphi$

Der Sequenz-Operator $\alpha; \varphi$ fordert, dass ein endlicher Teil des Wortes den regulären Ausdruck erfüllen muss, bevor das unendlich lange Restwort durch die RLTL-Formel erfüllt wird. Die Semantik entspricht der des Next-Operators in LTL, wenn als regulärer Ausdruck `true` gewählt wird, um eine Verzögerung von einem Schritt auszudrücken. Die Power-Operatoren $\varphi \mid \alpha \gg \psi$ und $\varphi \mid \alpha > \psi$ verlangen, dass die *Pflicht* φ jeweils mit einer *Verzögerung* α gelten muss, bis der *Versuch* ψ erfüllt ist. Der schwache Power-Operator $\varphi \mid \alpha > \psi$ erlaubt zudem, dass der Pflichtteil der Formel unendlich oft nach den von der Verzögerung festgelegten Abständen erfüllt sein kann. Für den Fall, dass als regulärer Ausdruck `true` verwendet wird, was einer Verzögerung von einem Schritt entspricht, sind die Semantiken für die Power-Operatoren äquivalent zu denen der Until- und Weak-Until-Operatoren in LTL. Die Power-Operatoren sind einfach zu benutzen, ermöglichen die Darstellung anderer temporaler Operatoren und kommen auch bei Past-Formeln und Negationen ohne nicht-algebraische Fixpunktkonstruktionen aus. Gegenüber anderen temporalen Logiken und Spezifikationssprachen bietet RLTL einige weitere Vorteile, auf die in Abschnitt 2.4 eingegangen wird.
Dass die Menge der mittels LTL-Formeln beschreibbaren Sprachen eine echte Teilmenge der durch RLTL-Formeln ausdrückbaren Sprachen ist, verdeutlicht das folgende Beispiel. Die Aussage, dass an jeder zweiten Position in einem Lauf die Proposition q gelten soll, lässt sich durch die Formel $q \mid \texttt{true;true} > \varnothing$ in RLTL spezifizieren. Da $\varnothing$ niemals erfüllt ist, müssen, damit die gesamte Formel erfüllt sein kann, unendlich oft der Pflicht- und der Verzögerungsteil erfüllt sein. Dies erfordert gerade, dass q wiederholt mit einer

Verzögerung von zwei Schritten, also an jeder zweiten Position gilt. Dass sich derartige Eigenschaften nicht in LTL ausdrücken lassen, wurde in [Wolper, 1981] bewiesen.

Ohne die Ausdrucksmächtigkeit weiter zu erhöhen, können für RLTL weitere Operatoren definiert werden. In [Sànchez und Samborski-Forlese, 2012] sind auch die Operatoren definiert, die zu dem Sequenz-Operator und den Power-Operatoren dual sind.

Definition 2.5 (Universelle Operatoren für RLTL, [Sànchez und Samborski-Forlese, 2012])**.** *Seien $w \in \Sigma^\omega$ ein unendliches Wort, i und j Positionen in w, α ein regulärer Ausdruck sowie φ und ψ RLTL-Formeln. Dann gelte*

- $(w,i) \models \alpha \cdot \varphi \quad \iff \forall k : (w,i,k) \models_{\text{RE}} \alpha \rightarrow (w,k) \models \varphi$
- $(w,i) \models \varphi \parallel \alpha \gg \psi \iff (w,i) \models \psi$ *und* $\forall \langle i_0, i_1, \cdots, i_m \rangle$
  $(w, i_k, i_{k+1}) \models_{\text{RE}} \alpha \rightarrow \exists j < k : (w,i_j) \models \varphi \vee (w,i_{k+1}) \models \psi,$
  und $\forall \langle i_0, i_1, \cdots \rangle$ *mit* $(w, i_k, i_{k+1}) \models_{\text{RE}} \alpha \wedge (w,i_k) \models \psi$
  $\rightarrow \exists m : (w,i_m) \models \varphi$
- $(w,i) \models \varphi \parallel \alpha > \psi \iff (w,i) \models \varphi \parallel \alpha \gg \psi$ *oder*
  $(w,i) \models \psi$ *und* $\forall k,j : (w,i,j) \models_{\text{RE}} \alpha^k \rightarrow (w,j) \models \psi$

Die Dualitätsgesetze für die universellen Operatoren lassen sich direkt angeben. Durch sie ist es möglich, Negationen bis auf die Ebene der Propositionen in die Formeln hineinzuziehen und somit die Negationsnormalform zu bilden.

$$\neg(\alpha;\varphi) \equiv \alpha \cdot \neg\varphi \qquad \neg(\alpha \cdot \varphi) \equiv \alpha;\neg\varphi$$
$$\neg(\varphi \mid \alpha \gg \psi) \equiv \neg\varphi \parallel \alpha > \neg\psi \qquad \neg(\varphi \mid \alpha > \psi) \equiv \neg\varphi \parallel \alpha \gg \neg\psi$$
$$\neg(\varphi \mid \alpha > \psi) \equiv \neg\varphi \parallel \alpha \gg \neg\psi \qquad \neg(\varphi \mid \alpha \gg \psi) \equiv \neg\varphi \parallel \alpha > \neg\psi$$

Für die Power-Operatoren sind zudem die folgenden Äquivalenzen definiert, sodass es, wie es beim Until-Operator in LTL der Fall ist, möglich ist, die Semantiken über Fixpunktgleichungen zu definieren.

Lemma 2.6 (Abrollung der Power-Operatoren auf unendlichen Worten)**.** *Seien φ, ψ RLTL-Formeln und $\alpha \in RE$ ein regulärer Ausdruck. Dann gilt*

$$\varphi \mid \alpha \gg \psi \equiv \psi \vee (\varphi \wedge \alpha; (\varphi \mid \alpha \gg \psi))$$
$$\varphi \mid \alpha > \psi \equiv \psi \vee (\varphi \wedge \alpha; (\varphi \mid \alpha > \psi))$$
$$\varphi \parallel \alpha \gg \psi \equiv \psi \wedge (\varphi \vee \alpha \cdot (\varphi \parallel \alpha \gg \psi))$$
$$\varphi \parallel \alpha > \psi \equiv \psi \wedge (\varphi \vee \alpha \cdot (\varphi \parallel \alpha > \psi))$$

Beweis. Die Äquivalenzen folgen direkt aus den Definitionen der Power-Operatoren. □

2.4 Andere Ansätze mit Temporallogiken

In der Literatur wurden weitere Ansätze beschrieben, Logiken so zu definieren, dass die Ausdrucksmächtigkeit von LTL erhöht wird. Einige dieser Ansätze bringen Nachteile mit sich, die dazu führen, dass ein Einsatz für praktische Anwendungszwecke nicht so einfach möglich ist, wie bei RLTL. *Erweiterte Temporale Logiken (ETL)* wie zum Beispiel in [Wolper, 1981] haben den Nachteil, dass zur Gewährleistung der vollständigen Ausdruckskraft unendlich viele Operatoren von Nöten sind. Reguläre Ausdrücke sind, wie bereits im Abschnitt über RLTL gezeigt wurde, dazu geeignet, die Ausdrucksmächtigkeit von LTL zu erhöhen. Sie ermöglichen es, die Spezifikationen auf eine relativ natürliche Art und Weise anzugeben und kommen daher in mehreren Logiken zum Einsatz. Die *Property Specification Language (PSL)* [Eisner et al., 2003] enthält ebenfalls reguläre Ausdrücke. Allerdings ist das Erfüllbarkeitsproblem bereits dann EXPSPACE-vollständig, wenn die so genannten *semi-extended regular expressions*, die zusätzlich zu der bei RLTL verwendeten Definition die Schnittbildung erlauben, verwendet werden. Die *Dynamic Linear time Temporal Logic (DLTL)* [Henriksen und Thiagarajan, 1999] beschreibt wie RLTL eine Kombination von LTL mit regulären Ausdrücken. Dabei werden durch Erweiterung der Next- und Until-Operatoren aus LTL weitere Operatoren eingeführt, die Ähnlichkeiten zu dem Sequenz-Operator und dem Power-Operator in RLTL haben. Für diese Logik wurden allerdings keine Past-Operatoren definiert, sodass es sich bei RLTL um die Logik handelt, in der sich die Eigenschaften einfacher und kompakter ausdrücken lassen. Für die *Linear Dynamic Logic (LDL)* [De Giacomo und Vardi, 2013], die sich syntaktisch an die *Propositional Dynaic Logic* [Fischer und Ladner, 1979] anlehnt, wurde mit LDL_f auch eine Variante auf endlichen Worten definiert. Hierfür wurden zwei unterschiedliche zweiwertige Semantiken definiert, die aber zueinander äquivalent sind.

2.5 Runtime Verification

In diesem Abschnitt werden die Grundlagen der Laufzeitverifikation (englisch: *Runtime-Verification* oder kurz *RV*) beschrieben. Nach einer Definition wird die Verifikationstechnik mit anderen Methoden, die zur Überprüfung von Softwaresystemen eingesetzt werden, verglichen, wobei Unterschiede und Gemeinsamkeiten aufgezeigt werden. Schließlich werden noch einige verschiedene Varianten der Runtime-Verification sowie die jeweiligen Einsatzgebiete erläutert. Die hier im Folgenden verwendete Definition für Runtime-Verification sowie die Erläuterungen der Begrifflichkeiten, die in diesem Forschungsgebiet gebräuchlich sind, basieren auf den Beschreibungen in [Leucker, 2012].
Mit *Runtime-Verification* werden die Entwicklung und Anwendung von Verifikationstechniken, die es ermöglichen, bei einem überwachten System zur Laufzeit festzustellen, ob

dieses bestimmte Korrektheitseigenschaften erfüllt oder verletzt, bezeichnet. Um dies zu ermöglichen, müssen sogenannte *Monitore* aus Spezifikationssprachen generiert werden. Ein *Lauf* eines Systems ist die möglicherweise unendliche Folge von Systemzuständen, welche auch als *Wort* aufgefasst werden kann. Eine *Ausführung* eines Systems ist ein endlicher Präfix eines Wortes, also folglich ein endliches Wort. Mithilfe des Monitors, der die Worte einliest, kann überprüft werden, ob eine Ausführung die Spezifikation einhält, was im einfachsten Fall geschieht, indem der Monitor »ja« oder »nein« ausgibt. Formal betrachtet geht es dabei um die Entscheidung, ob ein Wort in einer Sprache enthalten ist.

Es gibt verschiedene Varianten für den Einsatz von Runtime-Verification. Hinsichtlich der Läufe kann ein Monitor für den jeweiligen Anwendungsfall sowohl endliche Worte von terminierten Systemen als auch endliche aber stetig länger werdende sowie unendlich lange Worte überprüfen. Wenn mit dem Monitor zuvor abgespeicherte Ausführungen von Systemen kontrolliert werden sollen, spricht man davon, dass der Monitorvorgang *offline* stattfindet, andernfalls spricht man vom *online*-Monitoring. Dabei wird ein laufendes System überwacht, sodass der eingesetzte Monitor die kontinuierlich neu eintreffenden Informationen berücksichtigen muss. Außerdem kann man noch unterscheiden, ob der Monitor in das zu kontrollierende System integriert ist, oder außerhalb läuft (*inline*- und *outline*-Monitoring). Ein weiteres Kriterium unterscheidet, ob sich das zu überwachende System und der Monitor Ressourcen wie eine CPU teilen. In diesem Fall kann der Monitor das Verhalten des zu überwachenden Systems möglicherweise störend beeinflussen. Die Bezeichnung lautet für diesen Fall *eingreifendes* Monitoring; kommen hingegen zusätzliche Ressourcen für den Monitor zum Einsatz, spricht man vom *nicht-eingreifenden* Monitoring. Letztlich kann man noch unterscheiden, ob ein Monitor *passiv* arbeitet, das System also nur beobachtet und im Fehlerfall eine entsprechende Meldung ausgibt, oder *aktiv* ist, was bedeutet, dass die Ausgaben des Monitors Einfluss auf den Ablauf des Systems haben. Auch die Einsatzgebiete für Runtime-Verification können unterschiedlich sein. Zum einen lassen sich Safety- und Securityeigenschaften von Programmen überprüfen, zum anderen kann die Technik zur Performanzanalyse oder zur Beschaffung von Informationen über die Ausführung des Systems eingesetzt werden. Ein Monitor kann dabei diverse Eigenschaften des Systems überprüfen. Denkbar sind beispielsweise die Kontrolle von speziellen Events, dem Ein- und Ausgabeverhalten oder der Abfolge von bestimmen Zuständen des Systems.

Mittels Runtime-Verification wird immer nur die aktuelle Aktivität eines Systems überwacht. Andere denkbare Abläufe, die möglicherweise fehlerhaft sein könnten, finden keine Beachtung. Dies steht im Gegensatz zum Model-Checking, wo alle möglichen unendlichen Läufe betrachtet werden [Vardi und Wolper, 1986]. Um die Anwendbarkeit in der Praxis zu gewährleisten, können bei dieser Technik aus Effizienzgründen allerdings nur Modelle von (Teil-)Systemen geprüft werden. Darüber hinaus kann das Verhalten eines

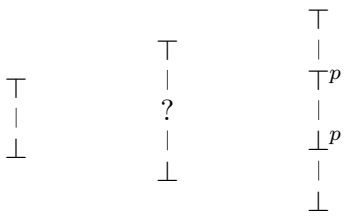

Abbildung 2.1: Die Wahrheitsverbände $\mathbb{B}_2$, $\mathbb{B}_3$ und $\mathbb{B}_4$ (von links nach rechts). Eine Verbindungslinie zwischen zwei Elementen a und b eines Verbandes bedeutet, dass $a \sqsubseteq b$ gilt und kein Element c existiert, sodass $a \sqsubseteq c \sqsubseteq b$ gilt.

Systems von der Einsatzumgebung abhängen, sodass im Vorfeld durchgeführte Tests das exakte Verhalten nicht simulieren können. In sicherheitskritischen Systemen kann es zudem sinnvoll sein, Runtime-Verification zusätzlich zu anderen Test- und Verifikationstechniken einzusetzen, um Fehlern in den zuvor durchgeführten Überprüfungen vorzubeugen.
Beim Testen wird hingegen überprüft, ob die Ergebnisse der Berechnungen eines Systems für bestimmte Eingaben den vorher festgelegten Werten entsprechen, wodurch im Allgemeinen nicht das vollständige System verifiziert werden kann. Die Überprüfung von temporallogischen Formeln kann gleichermaßen für geeignet gewählte Testeingaben durchgeführt werden. Somit kann die Laufzeitverifikation auch als eine Testmethode aufgefasst werden, bei der, anders als beim orakel-basierten Testen, wo das Orakel direkt definiert wird, der Monitor aus einer Spezifikationssprache generiert werden kann [Bauer et al., 2011]. Dabei werden weiterhin nur Teile des Systems überprüft.
Beim Model-Checking und beim Testen ist das Resultat der Überprüfungen stets ein eindeutiges, unwiderrufliches Ergebnis, sodass zur Beschreibung zwei Wahrheitswerte ausreichen. Im Folgenden wird verdeutlicht, dass es durch das Einführen zusätzlicher Monitorausgaben möglich ist, bei der Laufzeitverifikation zusätzliche Informationen zu erhalten.

2.6 Monitorbarkeit, zwei-, drei- und vierwertige Semantiken

Die Werte, zu denen die in diesem Abschnitt vorgestellten Semantiken die Worte auswerten, sind Elemente von Wahrheitsverbänden. Zur Erläuterung der Operationen auf diesen Verbänden dient die nun folgende Definition.

Definition 2.7 (Wahrheitsverband)**.** *Ein* Verband *ist eine partiell geordnete Menge $(\mathcal{S}, \sqsubseteq)$, für die für alle $x, y \in \mathcal{S}$ genau eine größte untere Schranke $x \sqcap y$,* Meet *genannt, und genau eine kleinste obere Schranke $x \sqcup y$,* Join *genannt, existierten.*

Die Verbände $\mathbb{B}_2$, $\mathbb{B}_3$ und $\mathbb{B}_4$, die in dieser Arbeit bei den zwei-, drei- und vierwertigen Semantiken Einsatz kommen, sind graphisch in Abbildung 2.1 dargestellt. Die Semantiken für LTL und RLTL, die in den Abschnitten 2.2 und 2.3 definiert wurden, beziehen sich auf unendliche Worte.
Des Weiteren existieren für LTL auch Definitionen auf endlichen Worten. Ein Überblick über unterschiedliche LTL-Semantiken, die bei der Runtime-Verification zum Einsatz kommen, liefert [Bauer et al., 2010]. Um die einzelnen Semantiken besser unterscheiden zu können, werden nun zwei Eigenschaften definiert, die den Zeitpunkt beschreiben, ab dem die endgültigen Wahrheitswerte verwendet werden.

Definition 2.8 (Impartiality)**.** *Sei $\mathcal{B} = \{\top, \bot\}$. Die Auswertung eines endlichen Wortes $u \in \Sigma^*$ ist* vorurteilsfrei (impartial)*, falls $\forall w \in \Sigma^* : [\![u \models \varphi]\!] \in \mathcal{B} \Rightarrow [\![u \models \varphi]\!] = [\![uw \models \varphi]\!]$*

Definition 2.9 (Anticipation)**.** *Die Auswertung eines endlichen Wortes $u \in \Sigma^*$ ist* antizipatorisch (anticipatory)*, falls $\forall w \in \Sigma^\omega$:*
$[\![u \models \varphi]\!] = \top \iff uw \models \varphi$ und $[\![u \models \varphi]\!] = \bot \iff uw \not\models \varphi$.

Unter welchen Umständen eine temporallogische Formel monitorbar ist, klärt folgende Definition.

Definition 2.10 (Monitorbarkeit, [Bauer et al., 2011])**.** *Seien φ eine (R)LTL-Formel, $u, v, w \in \Sigma^*$ endliche Worte und $x \in \Sigma^\omega$ ein unendliches Wort. Dann ist φ bezüglich einer Semantik, die auf unendlichen Worten definiert ist,* monitorbar*, falls $\forall u \exists v \forall x : uvx \models \varphi$ oder $uvx \not\models \varphi$. Bezüglich einer Semantik auf endlichen Worten ist φ* monitorbar*, falls $\forall u \exists v \forall w : [\![uvw \models \varphi]\!] = \top$ oder $[\![uvw \models \varphi]\!] = \bot$.*

Eine Formel ist somit nicht monitorbar, wenn ein Präfix nicht so erweitert werden kann, dass alle Fortsetzungen des Wortes die Formel entweder erfüllen oder verletzen.
In [Manna und Pnueli, 1987] klassifizieren die Autoren temporale Eigenschaften in einer Hierarchie, die in Abbildung 2.2 dargestellt ist. Zur Bestimmung der Klassen wurden vier Operatoren definiert, die Sprachen aus endlichen Sprachen konstruieren. Für unendliche Worte $w \in \Sigma^\omega$ und eine endliche Sprache Λ gilt:

- $A(\Lambda)$ enthält alle Worte w, sodass jeder Präfix von w in Λ liegt.
- $E(\Lambda)$ enthält alle Worte w, sodass ein Präfix von w in Λ liegt.
- $R(\Lambda)$ enthält alle Worte w, sodass unendlich viele Präfixe von w in Λ liegen.
- $P(\Lambda)$ enthält alle Worte w, sodass alle bis auf endlich viele Präfixe von w in Λ liegen.

Hiermit können die Klassen der temporalen Hierarchie für eine Sprache $\Pi \subseteq \Sigma^\omega$ über einem unendlichen Alphabet wie folgt mittels einer endlichen Sprache Λ charakterisiert werden:

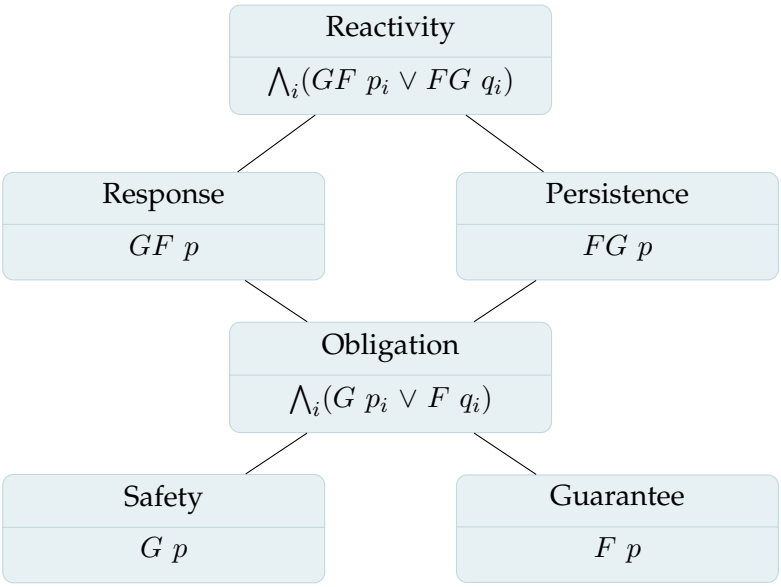

Abbildung 2.2: Graphische Darstellung der Klassen der temporalen Hierarchie aus [Manna und Pnueli, 1987] mit LTL Formeln für die jeweiligen Klassen. Eine Verbindungslinie bedeutet, dass eine Klasse in der darüber abgebildeten Klasse enthalten ist.

- Π ist eine *safety*-Sprache, wenn $\Pi = A(\Lambda)$ für ein $\Lambda \subseteq \Sigma^*$
- Π ist eine *guarantee*-Sprache, wenn $\Pi = E(\Lambda)$ für ein $\Lambda \subseteq \Sigma^*$
- Π ist eine *obligation*-Sprache, wenn $\Pi = \bigcap_i^m (A(\Lambda_i) \cup E(\Omega_i))$ für $\Lambda, \Omega \subseteq \Sigma^*$ und $m \in \mathbb{N}$
- Π ist eine *response*-Sprache, wenn $\Pi = R(\Lambda)$ für ein $\Lambda \subseteq \Sigma^*$
- Π ist eine *presistence*-Sprache, wenn $\Pi = P(\Lambda)$ für ein $\Lambda \subseteq \Sigma^*$
- Π ist eine *reactivity*-Sprache, wenn $\Pi = \bigcap_i^m (R(\Lambda_i) \cup P(\Omega_i))$ für $\Lambda, \Omega \subseteq \Sigma^*$ und $m \in \mathbb{N}$

Die einzelnen Klassen sind so aufgebaut, dass es möglich ist, die entsprechenden Eigenschaften mit einer Teilmenge der in LTL zur Verfügung stehenden Operatoren zu definieren. In der Abbildung sind für die Klassen beispielhaft LTL-Formeln angegeben. Ob spezielle temporale Formeln monitorbar sind, hängt von der für die Auswertung gewählten Semantik und der Einordnung in die temporale Hierarchie ab.
Um die verschiedenwertigen Semantiken für LTL miteinander vergleichen zu können, werden sie nacheinander definiert, wobei jeweils auf ihre spezifischen Eigenschaften eingegangen wird. Zu Beginn wird die zweiwertige Semantik für LTL auf endlichen Worten definiert, die mehrwertigen Semantiken folgen im Anschluss.

Definition 2.11 (Semantik für FLTL, [Lichtenstein et al., 1985])**.** *Sei $w \in \Sigma^+$ ein endliches Wort und $[\![\cdot \models \cdot]\!]_2 : \Sigma^+ \times LTL \to \mathbb{B}_2$. Die zweiwertige Semantik für eine LTL-Formel φ ist auf endlichen Worten wie folgt induktiv definiert:*

$$[\![w \models true]\!]_2 = \top$$

$$[\![w \models p]\!]_2 = \begin{cases} \top & \textit{falls } p \in w_0 \\ \bot & \textit{falls } p \notin w_0 \end{cases}$$

$$[\![w \models \neg\varphi]\!]_2 = \overline{[\![w \models_{\text{RE}} \varphi]\!]_2}$$

$$[\![w \models \varphi \vee \psi]\!]_2 = [\![w \models \varphi]\!]_2 \sqcup [\![w \models \psi]\!]_2$$

$$[\![w \models X\varphi]\!]_2 = \begin{cases} [\![w^1 \models \varphi]\!]_2 & \textit{falls } w^1 \neq \varepsilon \\ \bot & \textit{sonst} \end{cases}$$

$$[\![w \models \overline{X}\varphi]\!]_2 = \begin{cases} [\![w^1 \models \varphi]\!]_2 & \textit{falls } w^1 \neq \varepsilon \\ \top & \textit{sonst} \end{cases}$$

$$[\![w \models \varphi U \psi]\!]_2 = \begin{cases} \top & \textit{falls } \exists k\ 0 \leq k < |w| : [\![w^k \models \psi]\!]_2 = \top \textit{ und} \\ & \quad \forall l\ 0 \leq l < k : [\![w^l \models \varphi]\!]_2 = \top \\ \bot & \textit{sonst} \end{cases}$$

$$[\![w \models \varphi W \psi]\!]_2 = \begin{cases} \top & \textit{falls } [\![w \models \varphi U \psi]\!]_2 = \top \textit{ oder } \forall k 0 \leq k \leq |w| : [\![w \models \varphi]\!]_2 = \top \\ \bot & \textit{sonst} \end{cases}$$

Zweiwertige Semantiken sind im Allgemeinen nicht vorurteilsfrei, mit ihnen kann aber der Lauf eines bereits terminierten Systems analysiert werden. Außerdem sind sie im Falle von online-RV mit länger werdenden Worten nur dann geeignet, wenn bekannt ist, ob es sich bei der LTL-Formel um eine *safety*- oder *guarantee*-Formel handelt. Bei einer safety-Formel bedeutet eine Auswertung zu $\bot$, dass die Formel endgültig verletzt wurde, bei einer guarantee-Formel wird signalisiert, dass die zu überprüfende Eigenschaft definitiv eingehalten wurde, indem der Monitor »$\top$« ausgibt. In beiden Fällen sind nachträgliche Änderungen des Wahrheitswertes nicht mehr möglich. Dieses Prinzip wird in Tabelle 2.1 veranschaulicht. Bei den in der Hierarchie darüberliegenden Klassen kann es zu einem permanenten Wechsel zwischen beiden Ausgaben kommen, sodass die zweiwertige Semantik für derartige Formeln ungeeignet ist, da nicht erkennbar ist, ob die zuletzt erhaltene Ausgabe wirklich endgültig ist.

Dreiwertige Semantiken umgehen die Probleme, die bezüglich der Vorurteilsfreiheit auftreten, und führen einen weiteren Wahrheitswert ein, der ausdrückt, dass eine endgültige Auswertung mit den bisher vorliegenden Zeichen nicht möglich ist. In diesem Fall gibt

Typ	Formel	Semantik	Auswertung $\{a\}$	$\{a\}$	$\{a,b\}$	$\{a\}$	$\{b\}$	$\{\}$
Safety	Ga	zweiwertig	$\top$	$\top$	$\top$	$\top$	$\bot$	$\bot$
		dreiwertig	$?$	$?$	$?$	$?$	$\bot$	$\bot$
		vierwertig	$\top^p$	$\top^p$	$\top^p$	$\top^p$	$\bot$	$\bot$
Guarantee	Fb	zweiwertig	$\bot$	$\bot$	$\top$	$\top$	$\top$	$\top$
		dreiwertig	$?$	$?$	$\top$	$\top$	$\top$	$\top$
		vierwertig	$\bot^p$	$\bot^p$	$\top$	$\top$	$\top$	$\top$
Obligation	$Ga \wedge Fb$	zweiwertig	$\bot$	$\bot$	$\top$	$\top$	$\bot$	$\bot$
		dreiwertig	$?$	$?$	$?$	$?$	$\bot$	$\bot$
		vierwertig	$\bot^p$	$\bot^p$	$\top^p$	$\top^p$	$\bot$	$\bot$
Response	$G(a \rightarrow Fb)$	zweiwertig	$\bot$	$\bot$	$\top$	$\bot$	$\top$	$\top$
		dreiwertig	$?$	$?$	$?$	$?$	$?$	$?$
		vierwertig	$\bot^p$	$\bot^p$	$\top^p$	$\bot^p$	$\top^p$	$\top^p$

Tabelle 2.1: Verschiedene Auswertungsstrategien für unterschiedliche Klassen der temporalen Hierarchie für ein beispielhaft gegebenes Wort.

der Monitor »?« aus, um zu signalisieren, dass er unschlüssig ist. LTL_3 ist eine dreiwertige Semantik für LTL, mit der die Monitorbarkeit nicht mehr auf safety- und guarantee-Eigenschaften beschränkt ist. Dies zeigt die Auswertung der Obligation-Formel in Tabelle 2.1. Hier ergibt die Auswertung zunächst »?«, sobald der endgültige Wahrheitswert jedoch feststeht, wird gemäß dieser Semantik »$\bot$« ausgegeben. Bei der zweiwertigen Semantik kann aus der Ausgabe »$\bot$« hingegen nicht geschlossen werden, ob die Monitorausgabe noch verändert werden kann. Des Weiteren existiert das Problem der Alternation zwischen den beiden Wahrheitswerten $\top$ und $\bot$ bei der dreiwertigen Auswertung nicht mehr, da diese Werte die endgültigen Wahrheitswerte sind, die nach Definition auch bei allen Verlängerungen des Wortes nicht mehr verändert werden. Die dreiwertige Semantik für LTL ist dazu wie folgt definiert:

Definition 2.12 (Semantik für LTL_3, [Bauer et al., 2007])**.** *Seien $w \in \Sigma^*$ ein endliches Wort, $\varphi \in$ LTL eine LTL-Formel und $[\![\cdot \models \cdot]\!]_3 : \Sigma^* \times \text{LTL} \rightarrow \mathbb{B}_3$. Dann sei die dreiwertige Semantik wie folgt definiert:*

$$[\![w \models \varphi]\!]_3 = \begin{cases} \top & \textit{falls } \forall u \in \Sigma^\omega : wu \models \varphi \\ \bot & \textit{falls } \forall u \in \Sigma^\omega : wu \not\models \varphi \\ ? & \textit{sonst} \end{cases}$$

Die Semantik ist offenbar sowohl vorurteilsfrei als auch antizipatorisch. Für die zweite Eigenschaft muss bestimmt werden, ob eine Formel erfüllbar oder unerfüllbar ist. Dieses

Problem, das durch einen Leerheitstest auf einem alternierenden Büchi-Automaten gelöst werden kann, ist PSPACE-vollständig. Solange sich der Monitor nicht auf einen endgültigen Wahrheitswert festgelegt hat, wird stets »?« ausgegeben. Hierdurch erhält man allerdings keinerlei Informationen darüber, ob die Formel für das aktuell vorhandene Wort erfüllt ist, oder nicht. Die nicht monitorbare Persistenzformel in Tabelle 2.1 verdeutlicht diesen Sachverhalt. Die Ausgabe ist hier für sämtliche endlichen Eingabeworte immer »?«. Um aus den Ausgaben des Monitors weitere Informationen entnehmen zu können, ist es möglich, zusätzlich zwei Wahrheitswerte zu verwenden, die eine Auswertung beschreiben, die nicht endgültig ist. Dieses Prinzip verwendet die vierwertige Semantik für LTL, die wie folgt definiert ist:

Definition 2.13 (Semantik für FLTL$_4$, [Leucker, 2012])**.** *Seien $w \in \Sigma^+$ ein endliches Wort und $[\![\cdot \models \cdot]\!]_4 : \Sigma^+ \times LTL \to \mathbb{B}_4$, ε bezeichne wie üblich das leere Wort. Die vierwertige Semantiken für den starken und den schwachen Next-Operator sind wie folgt definiert:*

$$[\![w \models X\varphi]\!]_4 = \begin{cases} [\![w^1 \models \varphi]\!]_2 & \textit{falls } w^1 \neq \varepsilon \\ \bot^p & \textit{sonst} \end{cases}$$

$$[\![w \models \overline{X}\varphi]\!]_4 = \begin{cases} [\![w^1 \models \varphi]\!]_2 & \textit{falls } w^1 \neq \varepsilon \\ \top^p & \textit{sonst} \end{cases}$$

Atomare Propositionen, boolesche Kombinationen sowie boolesche Konstanten sind genauso definiert, wie bei der zweiwertigen Semantik in Definition 2.11. Der Until-Operator kann über die Äquivalenz $\varphi U \psi \equiv \psi \vee (\varphi \wedge X(\varphi U \psi))$ abgerollt werden, für den schwachen Until-Operator gilt entsprechend $\varphi W \psi \equiv \psi \vee (\varphi \wedge \overline{X}(\varphi W \psi))$.

Die vierwertige Semantik FLTL$_4$ wurde dazu konzipiert, für alle nichtleeren Präfixe des zu analysierenden Wortes eine Monitorausgabe zu generieren. Da das Wort bei der online-Runtime-Verification Schritt für Schritt erweitert wird, sollte der Monitor dieses auch Schritt für Schritt auswerten. Die Semantik wertet das Wort dabei immer so aus, als ob es an der aktuellen Position zu Ende wäre. Die vier möglichen Wahrheitswerte des $\mathbb{B}_4$ unterscheiden dabei zum einen, ob das Wort bis zur aktuellen Position ein Modell für die Formel ist; zum anderen beschreiben sie, ob diese Auswertung endgültig ist, oder ob sich das Resultat in der Zukunft noch verändern kann. Die endgültigen Wahrheitswerte sind wie bei den zuvor beschriebenen Semantiken $\top$ und $\bot$; *possibly true* ($\top^p$) und *possibly false* ($\bot^p$) geben an, dass die Formel zur Zeit erfüllt beziehungsweise nicht erfüllt ist, eine andere Auswertung aber ebenfalls noch möglich ist.
Die Definition eines dualen Next-Operators [Lichtenstein et al., 1985], der auch als schwacher Next-Operator bezeichnet wird, ermöglicht es, eine Unterscheidung in der Auswertung vorzunehmen, wenn das zu überprüfende Wort gerade soweit vorliegt, dass sich

der Next-Operator auf den noch fehlenden Teil des Wortes auswirkt. Der starke Next-Operator verlangt zunächst, dass ein weiteres Symbol vorhanden ist, sodass der Monitor bei der zweiwertigen Semantik »$\bot$« und bei der vierwertigen Semantik »$\bot^p$« ausgibt. Der schwache Next-Operator ist so definiert, dass das Wort an der nächsten Position die entsprechende Eigenschaft erfüllen muss, sofern die nächste Position existiert, sodass die Monitorausgaben im zweiwertigen Fall »$\top$« und im vierwertigen Fall »$\top^p$« sind. Die vierwertige Auswertung ist ebenfalls beispielhaft in Tabelle 2.1 veranschaulicht. Während die dreiwertige Semantik bei den jeweiligen Formeln nur die Ausgabe »?« produziert, ist die vierwertige Semantik mit den Werten $\top^p$ und $\bot^p$ deutlich präziser. Dies gilt insbesondere für die Response-Formel $G(a \rightarrow Fb)$, die für ein endliches Wort niemals zu $\top$ oder $\bot$ ausgewertet werden kann, da nach jedem beliebigen Präfix sowohl eine Erfüllung als auch eine Verletzung der Formel mit einer unendlich langen Fortsetzung des Wortes möglich sind.

Da bei der Überwachung von Systemen stets nur Präfixe von möglicherweise unendlichen Worten vorliegen und diese durch Berechnungen des Systems erst mit der Zeit verlängert werden, eignet sich eine vierwertige Semantik besonders gut zur Analyse dieser Worte. Die Eigenschaft der Vorurteilsfreiheit, ohne die eine korrekte Auswertung nicht möglich wäre, ist erfüllt. Antizipatorisch ist die Semantik allerdings nicht, so gilt nach Definition $[\![a \models X(a \wedge \neg a)]\!]_4 = \bot^p$, obwohl die LTL-Formel unerfüllbar ist. Erst nachdem das zweite Eingabesymbol verarbeitet ist, erfolgt eine Auswertung zu $\bot$. Da kein Erfüllbarkeitstest involviert ist, ist die Monitorgenerierung einfach und effizient realisierbar. Aus einer LTL-Formel lässt sich direkt eine alternierende Mealy-Maschine generieren [Leucker, 2012], welche als Monitor im Kontext der online-Runtime-Verification verwendet werden kann. Im folgenden Kapitel wird eine vierwertige Semantik für RLTL vorgestellt, die Gleiches für die in der Ausdrucksmächtigkeit erweiterte Logik ermöglicht.

3 Vierwertige Semantik

In diesem Kapitel wird in Anschluss an die Beschreibung von notwendigen Anpassungen der regulären Linearzeitlogik die vierwertige Semantik für RLTL auf endlichen Worten definiert. Analog zu FLTL_4 trägt die neue Semantik den Namen FRLTL_4. Außerdem wird begründet, dass die über die Semantikfunktionen bestimmten Wahrheitswerte korrekt beschreiben, ob eine Formel durch ein Wort endgültig erfüllt oder unerfüllt, beziehungsweise mit den bislang vorliegenden Zeichen des Wortes, das noch um weitere Symbole verlängert wird, erfüllt oder unerfüllt ist, wobei sich die entsprechende Auswertung noch ändern kann.
Damit die Semantikfunktionen für FLTL_4 und FRLTL_4 bei LTL-Formeln und dazu äquivalenten RLTL-Formeln bei Betrachtung von endlichen Worten den gleichen Wahrheitswert liefern, muss die Definition von RLTL, die im vorherigen Kapitel vorgestellt wurde, überarbeitet werden. In Abschnitt 3.1 werden die auftretenden Probleme an Beispielen verdeutlicht. Außerdem wird zur Lösung der Probleme unter anderem ein weiterer schwacher Next-Operator definiert, zu dem ebenfalls ein dualer Operator existiert. Der Weiteren wird argumentiert, weswegen sich die Variante von RLTL, in der Propositionen auch außerhalb von regulären Ausdrücken verwendet werden können, besonders eignet. Danach erfolgt in Abschnitt 3.2 die Definition der vierwertigen Semantik für RLTL auf endlichen Worten. Die Semantik für RLTL-Formeln wird dabei zum Zwecke der Übersichtlichkeit aufbauend auf der separat notierten Definition der Semantik für reguläre Ausdrücke angegeben. Die Begründung weswegen die definierte Semantik die Worte korrekt auswertet folgt zum Abschluss des Kapitels in Abschnitt 3.3.

3.1 Erweiterungen von RLTL

Bevor die vierwertige Semantik für RLTL auf endlichen Worten formal definiert wird, werden zunächst die Anforderungen, die die Semantik zwingend erfüllen muss, rekapituliert. Da RLTL eine Erweiterung von LTL ist und es zu jeder LTL-Formel eine äquivalente RLTL-Formel gibt, muss folgende Gleichung für die Semantiken erfüllt sein, wobei $\tau : \text{LTL} \to \text{RLTL}$ über die Aussage $w \in L(\varphi) \iff w \in L(\tau(\varphi))$ spezifiziert werden kann.

$$\forall \varphi \in \text{LTL} : [\![w \models \varphi]\!]_4^{\text{LTL}} = [\![w \models \tau(\varphi)]\!]_4^{\text{RLTL}}$$

Wie bei der vierwertigen Semantik für LTL ist die Semantik, die in dieser Arbeit definiert wird, also vorurteilsfrei aber nicht antizipatorisch. Dass sich diese Anforderungen jedoch gegenseitig ausschließen, wenn ungeeignete Umwandlungsfunktionen verwendet werden, machen die folgenden Überlegungen deutlich, in denen die LTL-Formel p betrachtet wird, die nur aus einer Proposition besteht. Für derartige Formeln wurden in verschiedenen Arbeiten diverse Übersetzungen in RLTL-Formeln definiert.

1. $\tau_1(p) = \widehat{p}$
2. $\tau_2(p) = p; \neg\varnothing$
3. $\tau_3(p) = p$

Der zusätzliche Operator $\widehat{\cdot}$ aus [Leucker und Sánchez, 2007] ist in der später definierten Version von RLTL [Sánchez und Leucker, 2010] nicht mehr vorhanden. Die Semantik von $\widehat{\alpha}$ besagt, dass jedes unendliche Wort, das einen Präfix hat, der α erfüllt, Modell für die entsprechende Formel ist. Gleiches lässt sich ebenfalls durch die RLTL-Formel $\alpha; \neg\varnothing$ ausdrücken. Die dritte Variante, die ebenfalls aus [Sánchez und Leucker, 2010] stammt, erfordert, dass die Grammatik für RLTL-Formeln zusätzlich die Ableitung zu Propositionen zulässt.
Die zweite Variante ist diejenige, die in der Automatenbibliothek RltlConv ursprünglich eingesetzt wurde. Bei der Betrachtung von unendlichen Worten ist dies auch ohne Weiteres möglich, bei endlichen Worten treten allerdings Probleme auf. Sei $\varphi_1 = a$ eine LTL-Formel. Offenbar gilt $[\![a \models a]\!]_4^{\text{LTL}} = \top$. Dies impliziert, dass auch $[\![a \models \tau_2(a)]\!]_4^{\text{RLTL}} = \top$ gelten muss. Also folgt

$$[\![a \models \tau_2(a)]\!]_4^{\text{RLTL}} = [\![a \models \texttt{a}; \neg\varnothing]\!]_4^{\text{RLTL}} \stackrel{!}{=} \top$$

Als nächstes Beispiel sei die LTL-Formel $\varphi_2 = Xtrue$ mit $\tau_2(\varphi_2) = \texttt{true}; \neg\varnothing$ gegeben. Nach der Semantik aus Definition 2.13 gilt $[\![a \models Xtrue]\!]_4^{\text{LTL}} = \bot^p$. Es gilt folglich

$$[\![a \models \tau_2(Xtrue)]\!]_4^{\text{RLTL}} = [\![a \models \texttt{true}; \neg\varnothing]\!]_4^{\text{RLTL}} \stackrel{!}{=} \bot^p$$

Hieraus folgt für die in RLTL strukturell identischen Formeln auf endlichen Worten eine semantische Ungleichheit, obwohl die Auswertung für das gleiche Wort identische Werte liefern sollte. Es gilt

$$[\![a \models \texttt{a}; \neg\varnothing]\!]_4^{\text{RLTL}} \neq [\![a \models \texttt{true}; \neg\varnothing]\!]_4^{\text{RLTL}}$$

Um dieses Problem zu umgehen, kann man es in Betracht ziehen, die Übersetzungsfunktion τ_2 so zu modifizieren, dass die LTL-Formel $true$ ebenso übersetzt wird, wie die Pro-

positionen. Dann gelten $\tau_2'(true) = \texttt{true};\neg\varnothing$ sowie $\tau_2'(p) = \texttt{p};\neg\varnothing$ und $\tau_2'(Xtrue) = \texttt{true};(\texttt{true};\neg\varnothing)$. Dies liefert

$$[\![a \models \texttt{a};\neg\varnothing]\!]_4^{\text{RLTL}} \stackrel{!}{=} \top, \text{ da } [\![a \models a]\!]_4^{\text{LTL}} = \top$$
$$[\![a \models \texttt{true};\neg\varnothing]\!]_4^{\text{RLTL}} \stackrel{!}{=} \top, \text{ da } [\![a \models true]\!]_4^{\text{LTL}} = \top$$
$$[\![a \models \texttt{true};(\texttt{true};\neg\varnothing)]\!]_4^{\text{RLTL}} \stackrel{!}{=} \bot^p, \text{ da } [\![a \models Xtrue]\!]_4^{\text{LTL}} = \bot^p$$

In diesem Fall erfordert die vierwertige Semantik für RLTL, dass die Auswertung des regulären Ausdrucks bei dem Sequenz-Operator von der Teilformel im hinteren Teil abhängt. Dies führt direkt zu der Problematik, dass unklar ist, wie eine Semantik, die nicht antizipatorisch ist, mit Tautologien umgehen soll.

$$[\![a \models \texttt{a};\neg\varnothing]\!]_4^{\text{RLTL}} \stackrel{!}{=} \top$$
$$[\![a \models \texttt{a};\underbrace{\neg(\neg\varnothing \wedge \varnothing)}_{\neg\varnothing}]\!]_4^{\text{RLTL}} \stackrel{?}{=} \top$$
$$[\![a \models \texttt{a};\underbrace{((a;\neg\varnothing) \vee \neg(a;\neg\varnothing))}_{\neg\varnothing}]\!]_4^{\text{RLTL}} \stackrel{?}{=} \bot^p$$

In diesem Beispiel wird das Wort für die zweite Formel zu $\top$ ausgewertet, da die Tautologie in der nur aus Konstanten und elementaren booleschen Operatoren bestehenden Formel unmittelbar zu erkennen ist und keine weiteren Sequenz-Operatoren enthalten sind, die verlangen, dass weitere Zeichen gelesen werden. In der dritten Gleichung hingegen sind in der Tautologie weitere Sequenz-Operatoren enthalten. Wenn man dem Prinzip folgt, dass diese zunächst ausgewertet werden müssen, was nach dem ersten gelesenen Zeichen bei einer vierwertigen Semantik noch nicht der Fall ist, ergibt dies den Wahrheitswert $\bot^p$. Gleichwohl es möglich wäre, in der Semantik zu definieren, in welchen Fällen eine Auswertung zu $\top$ erfolgt, scheint ein solcher Ansatz nicht sonderlich intuitiv zu sein. Die Einführung des $\hat{\cdot}$-Operators wäre möglich, allerdings sind Personen, die RLTL einsetzen, mit Propositionen, die in einer Vielzahl an anderen Logiken enthalten sind, aller Voraussicht nach bereits vertraut, sodass in dieser Arbeit die um Propositionen erweiterte Version von RLTL eingesetzt wird. Die Automatenbibliothek RltlConv wurde parallel zu dieser Arbeit um die entsprechenden Funktionen erweitert.

In Abschnitt 2.3 wurden unter anderem die Abrollungen der Power-Operatoren beschrieben. Dies ist unter der Annahme geschehen, dass unendliche Worte betrachtet werden. Dass die Äquivalenzen aus Lemma 2.6 nicht für endliche Worte gelten, wird an folgendem Beispiel verdeutlicht. Sei $\varphi = a \mid \texttt{a;b} > c$ eine RLTL-Formel, die aus einem Weak-Power-Operator besteht. Die Pflicht- und Versuch-Teilformeln bestehen lediglich aus Propositio-

nen. Die vierwertige Semantik auf endlichen Worten wurde noch nicht formal definiert. Daher werden in den folgenden Überlegungen die Wahrheitswerte betrachtet, die sich intuitiv ergeben sollten. Sei $w = ab$ das endliche Wort. Dann sind sowohl der Pflicht- als auch der Verzögerungsteil der Formel erfüllt. Die Formel ist aber nicht komplett erfüllt, da hierzu der Versuchsteil erfüllt sein müsste. Andererseits kann φ auch noch verletzt werden, zum Beispiel wenn an der nächsten Position des Wortes $\{\}$ gilt. Die logische Auswertung ist somit $\top^p$. Die auf unendlichen Worten zu φ äquivalente Formel ist $\varphi' = c \vee (a \wedge ((\mathtt{a;b}); \varphi))$. Im Beispielwort gilt an der ersten Position nicht c, sondern a. Damit bleibt, nachdem mit a das erste Zeichen gelesen wurde, als zu erfüllender Teil der Formel die Teilformel $\mathtt{b}; \varphi$ übrig. Da der Sequenz-Operator semantisch dem starken Next-Operator in LTL entspricht und keine Zeichen für den Teil der Formel vorhanden sind, der nach der Verzögerung gelten soll, wird, da ab den regulären Ausdruck erfüllt, die Formel zu $\bot^p$ ausgewertet.
Dass sich dieses Problem nicht mit dem dualen Sequenz-Operator lösen lässt, ist durch das nachfolgende Beispiel ersichtlich. Sei nun für die Formel $\varphi = a \mid \mathtt{a;b} > c$ das Wort $w = aa$ gegeben. Die Formel ist aufgrund der Nichteinhaltung der in der Verzögerung spezifizierten Propositionen unwiderruflich verletzt, die Ausgabe muss demzufolge $\bot$ sein. Betrachtet man stattdessen $\widetilde{\varphi} = c \vee (a \wedge ((\mathtt{a;b}) \cdot \varphi))$ als mögliche Abrollung, erfolgt eine Auswertung zu $\top$, da diese Formel für alle unendlichen Fortsetzungen von w erfüllt ist, weil die Verzögerung vor dem dualen Sequenz-Operator nicht erfüllt ist. Zusammengefasst gilt also

$$[\![ab \models a \mid \mathtt{a;b} > c]\!] = \top^p \qquad [\![aa \models a \mid \mathtt{a;b} > c]\!] = \bot$$
$$[\![ab \models c \vee (a \wedge (\mathtt{a;b}); \varphi)]\!] = \bot^p \qquad [\![aa \models c \vee (a \wedge (\mathtt{a;b}) \cdot \varphi)]\!] = \top$$

Während die vierwertige Semantik für LTL mit einem Next-Operator und einem Weak-Next-Operator, welche dual zueinander sind, auskommt, muss die Unterscheidung für RLTL verfeinert werden. In dieser Logik werden insgesamt vier Next-Operatoren benötigt, da es möglich sein muss, sowohl zwischen stark und schwach als auch zwischen existentiell und universell zu differenzieren. In LTL beschreiben die Next-Operatoren eine Verzögerung von einem Schritt. Daher gibt es auch nur eine einzige Möglichkeit zur Erfüllung, weswegen eine existentielle und eine universelle Auswertung zum selben Ergebnis führen. Eine Unterscheidung, ob Eigenschaften bei Erfüllungen von regulären Ausdrücken existentiell oder universell gelten müssen, wurde außer bei der Spezifikation mit Temporallogiken bereits bei den quantifizierten regulären Ausdrücken angewendet [Olender und Osterweil, 1990].
Die folgende Definition der Next-Operatoren dient nun als Erweiterung der in Definition 2.4 festgelegten Semantik für unendliche Worte.

Definition 3.1 (Next-Operatoren für RLTL)**.** *Seien $\alpha \in RE$ ein regulärer Ausdruck und $\varphi \in$ RLTL eine RLTL-Formel. Dann gilt für ein unendliches Wort $w \in \Sigma^\omega$*

- $(w,i) \models \alpha \,;_\Sigma\, \varphi \quad \iff \exists k : (w,i,k) \models_{\text{RE}} \alpha$ *und* $(w,k) \models \varphi$
- $(w,i) \models \alpha \,;_\Pi\, \varphi \quad \iff \forall k : (w,i,k) \models_{\text{RE}} \alpha \longrightarrow (w,k) \models \varphi$

Dabei bezeichnet $;_\Sigma$ den *starken existentiellen Next-Operator* und $;_\Pi$ den *starken universellen Next-Operator*. Es gelten die folgenden Dualitätsgesetze, aus denen sich die Definitionen der beiden schwachen Next-Operatoren ableiten lassen.

$$\neg(\alpha \,;_\Sigma\, \varphi) \equiv \alpha \overline{;_\Pi} \neg\varphi \qquad \neg(\alpha \overline{;_\Pi} \varphi) = \alpha \,;_\Sigma\, \neg\varphi$$
$$\neg(\alpha \,;_\Pi\, \varphi) \equiv \alpha \overline{;_\Sigma} \neg\varphi \qquad \neg(\alpha \overline{;_\Sigma} \varphi) = \alpha \,;_\Pi\, \neg\varphi$$

Der Operator $\overline{;_\Sigma}$ ist der *schwache existentielle Next-Operator*, $\overline{;_\Pi}$ ist analog der *schwache universelle Next-Operator*. Diese Notation dient vor allem der Verdeutlichung der unterschiedlichen Konzepte, insbesondere ist festzuhalten, dass $;_\Sigma$ identisch zum Sequenz-Operator aus Definition 2.4 ist und $\overline{;_\Pi}$ lediglich eine neue Schreibweise für den universellen Sequenz-Operator aus Definition 2.5 ist. Die semantischen Unterschiede zwischen den beiden starken und schwachen Next-Operatoren, die sich bei der Auswertung von endlichen Worten ergeben, werden durch die Definition im folgenden Abschnitt deutlich.

3.2 Definition der vierwertigen Semantik für RLTL

In diesem Abschnitt wird nun formal die vierwertige Semantik für RLTL auf endlichen Worten definiert. Wie bereits erläutert, bestimmt die Semantikfunktion, ob ein Wort ein Modell für die gegebene Formel ist und trifft dabei auch eine Aussage darüber, ob diese Entscheidung endgültig ist, oder ob ein um zusätzliche Zeichen verlängertes Wort ein anderes Resultat liefern kann. Die Funktionen $[\![\cdot \models_{\text{RE}} \cdot]\!]$ und $[\![\cdot \models \cdot]\!]$ berechnen dabei die Semantik für reguläre Ausdrücke und RLTL-Formeln. Nach den jeweiligen Definitionen werden zunächst kurz die Grundgedanken der entsprechenden Semantikfunktionen für die einzelnen Operatoren beschrieben. Warum die Definitionen die gestellten Anforderungen einhalten und Worte im Sinne einer vierwertigen Semantik, wie sie von LTL-Formeln bekannt ist, korrekt auswerten, wird im Anschluss an die beiden Definitionen in Abschnitt 3.3 begründet.

Definition 3.2 (Semantikfunktion für reguläre Ausdrücke)**.** *Seien $w \in \Sigma^+$ mit $w \neq \varepsilon$ ein endliches nicht leeres Wort, α, β reguläre Ausdrücke und p eine atomare Proposition. Dann ist die Semantikfunktion für reguläre Ausdrücke $[\![\cdot \models_{\text{RE}} \cdot]\!] : \Sigma^+ \times RE \to \mathbb{B}_4$ wie folgt definiert:*

$$[\![w \models_{\mathrm{RE}} \mathrm{p}]\!] = \begin{cases} \top & \mathrm{p} \in w_0 \wedge |w| = 1 \\ \bot & \textit{sonst} \end{cases}$$

$$\mathfrak{F}([\![w \models_{\mathrm{RE}} \alpha]\!]) = \begin{cases} \top & [\![w \models_{\mathrm{RE}} \alpha]\!] = \top \\ \bot & \textit{sonst} \end{cases}$$

$$\mathfrak{B}([\![w \models_{\mathrm{RE}} \alpha]\!]) = \begin{cases} \top & [\![w \models_{\mathrm{RE}} \alpha]\!] = \bot \\ \bot^p & \textit{sonst} \end{cases}$$

$$[\![w \models_{\mathrm{RE}} \alpha + \beta]\!] = [\![w \models_{\mathrm{RE}} \alpha]\!] \sqcup [\![w \models_{\mathrm{RE}} \beta]\!]$$

$$[\![w \models_{\mathrm{RE}} \alpha ; \beta]\!] = (\bot^p \sqcap [\![w \models_{\mathrm{RE}} \alpha]\!]) \sqcup \bigsqcup_{0<i<|w|} \left(\mathfrak{F}([\![^i w \models_{\mathrm{RE}} \alpha]\!]) \sqcap [\![w^i \models_{\mathrm{RE}} \beta]\!]\right)$$

$$[\![w \models_{\mathrm{RE}} \alpha * \beta]\!] = \bigsqcup_{0\leq i<|w|} \Bigg(\left([\![w^i \models_{\mathrm{RE}} \beta]\!] \sqcup \left(\bot^p \sqcap [\![w^i \models_{\mathrm{RE}} \alpha]\!]\right)\right) \sqcap \bigsqcup_{Y \in \mathcal{Y}(^i w)} \left(\bigsqcap_{\langle y \rangle \in Y} \mathfrak{F}([\![^{y_1} w^{y_0} \models_{\mathrm{RE}} \alpha]\!]) \right) \Bigg)$$

Dabei gibt die Funktion $\mathcal{Y}$, die in der Definition der Semantik vom Kleene-Operator verwendet wird, die Menge aller möglichen Aufteilungen eines endlichen Wortes in beliebig viele nichtleere Teilworte zurück, wobei diese Unterteilungen als Menge von Start- und Endposition festlegenden Tupeln repräsentiert werden. Formal gilt

$$\mathcal{Y}(w) := \Big\{ \{\langle x_0, x_1\rangle, \langle x_1, x_2\rangle, \cdots, \langle x_{n-2}, x_{n-1}\rangle, \langle x_{n-1}, x_n\rangle\} \\ \Big| \; x_0 = 0 \wedge x_n = |w| \wedge \forall i \in \{0, \cdots, n-1\} : x_i < x_{i+1} \Big\}$$

Für Propositionen wird zum einem sichergestellt, dass die Proposition an der betrachteten Stelle des Wortes gilt, zum anderen wird überprüft, ob das zu entsprechende Teilwort nur aus einer Position besteht. Nur wenn beides der Fall ist, ist das Wort ein Modell für die Proposition. Die Funktion $\mathfrak{F}$ bildet auf die beiden endgültigen Wahrheitswerte $\top$ und $\bot$ ab. Anders formuliert bedeutet dies, dass die Auswertung zweiwertig erfolgt. Damit die Funktion $\mathfrak{B}$ eine Auswertung zu $\top$ bewirkt, darf das Wort gerade kein Modell für den regulären Ausdruck sein. Diese Funktion wird erst in der Definition der vierwertigen Semantik für die RLTL-Operatoren benötigt. Dort wird auch näher auf den Nutzen dieser Funktion eingegangen. Bei der Disjunktion werden beide Teilausdrücke separat evaluiert. Durch die Verknüpfung mit dem Join-Operator des $\mathbb{B}_4$ wird der größere Wahrheitswert ausgewählt. Bei der Konkatenation wird unterschieden, ob alle Zeichen des bislang vor-

liegenden Wortes den Teilausdruck α erfüllen oder ob eine Aufteilung existiert, sodass der vordere Teil α endgültig erfüllt und der hintere Teil des Wortes den Ausdruck β erfüllt. Die Auswertung des Kleene-Operator erfolgt, indem nach Sequenzen gesucht wird, in denen wiederholt α gilt. Der Rest des Wortes muss anschließend noch β erfüllen. Die Funktion $\mathfrak{F}$, die bei den beiden zuletzt genannten Operatoren zum Einsatz kommt, wird benötigt, um sicherzustellen, dass die Semantik einen Teilausdruck α nicht fälschlicherweise zu $\bot^p$ auswertet, wenn ein Teil des Eingabewortes bereits zur Auswertung von β herangezogen wurde. Hierzu werden als Beispiel der reguläre Ausdruck `(a;b);c` und das Eingabewort ac betrachtet. Offenbar gilt $[\![ac \models_{\text{RE}} \texttt{(a;b);c}]\!] = \bot$, da das Wort den regulären Ausdruck nicht erfüllt, da an der zweiten Position des Wortes kein b vorhanden ist, und diesen auch mit allen denkbaren Verlängerungen nicht erfüllen wird. Aufgrund des Aufrufs der Funktion $\mathfrak{F}$ wird bei der Auswertung verhindert, dass bei einer Aufteilung des Wortes an der ersten Position ein inkorrekter Wahrheitswert berechnet wird. Das folgende Beispiel vergleicht die in der Definition angegebene Semantikfunktion mit der Variante, in der der Wahrheitswert, der bei der Auswertung des regulären Ausdrucks ermittelt wird, ohne Aufruf der Funktion $\mathfrak{F}$ übernommen wird. In der oberen Zeile ergibt sich aufgrund der Auswertung von $[\![a \models_{\text{RE}} \texttt{a;b}]\!]$ zu $\bot^p$ und der Tatsache, dass `c` den hinteren Teil erfüllt, sodass für diesen Teil $\top$ berechnet wird, der Wahrheitswert $\bot^p$. Die Funktion $\mathfrak{F}$ bildet diesen Wert, wie es in der unteren Zeile ausführlich beschreiben ist, auf $\bot$ ab, sodass die Auswertung des gesamten regulären Ausdrucks korrekterweise $\bot$ ist.

$$
\begin{aligned}
[\![ac \models_{\text{RE}} \texttt{(a;b);c}]\!] &= \quad [\![a \models_{\text{RE}} \texttt{a;b}]\!] \sqcap [\![c \models_{\text{RE}} \texttt{c}]\!] = \quad \bot^p \sqcap \top \qquad\qquad = \bot^p \\
[\![ac \models_{\text{RE}} \texttt{(a;b);c}]\!] &= \mathfrak{F}([\![a \models_{\text{RE}} \texttt{a;b}]\!]) \sqcap [\![c \models_{\text{RE}} \texttt{c}]\!] = \mathfrak{F}(\bot^p) \sqcap \top = \bot \sqcap \top \; = \bot
\end{aligned}
$$

Für den Fall, dass die gesamte Eingabe für α verwendet wird, berechnet die Semantikfunktion in beiden Varianten $\bot$. Dieser Fall ist im Beispiel aus Gründen der Übersichtlichkeit nicht aufgeführt.
Unter Zuhilfenahme der Semantikfunktion für reguläre Ausdrücke ist es nun möglich, die Definition der Semantikfunktion für RLTL-Formeln anzugeben.

Definition 3.3 (FRLTL$_4$, die vierwertige Semantik für RLTL auf endlichen Worten)**.** *Seien φ, ψ RLTL-Formeln, $\texttt{p}$ eine Proposition, α ein regulärer Ausdruck und $w \in \Sigma^+$, ein endliches, nichtleeres Wort. Die vierwertige Semantik ist über die Funktion $[\![\cdot \models \cdot]\!] : \Sigma^+ \times RLTL \to \mathbb{B}_4$ wie folgt definiert:*

$$
\begin{aligned}
[\![w \models \varnothing]\!] &= \bot \\
[\![w \models \texttt{p}]\!] &= \begin{cases} \top & \texttt{p} \in w_0 \\ \bot & \textit{sonst} \end{cases}
\end{aligned}
$$

$$
\begin{aligned}
[\![w \models \varphi \vee \psi]\!] &= [\![w \models \varphi]\!] \sqcup [\![w \models \psi]\!] \\
[\![w \models \varphi \wedge \psi]\!] &= [\![w \models \varphi]\!] \sqcap [\![w \models \psi]\!] \\
[\![w \models \neg\varphi]\!] &= \overline{[\![w \models \varphi]\!]} \\
[\![w \models \alpha ;_\Sigma \varphi]\!] &= (\bot^p \sqcap [\![w \models_{\mathrm{RE}} \alpha]\!]) \sqcup \bigsqcup_{0<i<|w|} \left(\mathfrak{F}([\![^i w \models_{\mathrm{RE}} \alpha]\!]) \sqcap [\![w^i \models \varphi]\!]\right) \\
[\![w \models \alpha ;_\Pi \varphi]\!] &= \mathfrak{B}([\![w \models_{\mathrm{RE}} \alpha]\!]) \sqcap \bigsqcap_{0<i<|w|} \left(\mathfrak{F}([\![^i w \models_{\mathrm{RE}} \alpha]\!]) \to [\![w^i \models \varphi]\!]\right) \\
[\![w \models \alpha \overline{;_\Pi} \varphi]\!] &= [\![w \models \neg(\alpha ;_\Sigma \neg\varphi)]\!] \\
[\![w \models \alpha \overline{;_\Sigma} \varphi]\!] &= [\![w \models \neg(\alpha ;_\Pi \neg\varphi)]\!] \\
[\![w \models \varphi \mid \alpha \gg \psi]\!] &= [\![w \models \psi \vee (\varphi \wedge (\alpha ;_\Sigma \varphi \mid \alpha \gg \psi))]\!] \\
[\![w \models \varphi \mid \alpha > \psi]\!] &= [\![w \models \psi \vee (\varphi \wedge (\alpha \overline{;_\Sigma} \varphi \mid \alpha > \psi))]\!] \\
[\![w \models \varphi \parallel \alpha \gg \psi]\!] &= [\![w \models \psi \wedge (\varphi \vee (\alpha ;_\Pi \varphi \parallel \alpha \gg \psi))]\!] \\
[\![w \models \varphi \parallel \alpha > \psi]\!] &= [\![w \models \psi \wedge (\varphi \vee (\alpha \overline{;_\Pi} \varphi \parallel \alpha > \psi))]\!]
\end{aligned}
$$

Da $\varnothing$ für kein Wort erfüllt ist, wertet die Semantik eine derartige Formel zu $\bot$ aus. Im Falle von Propositionen wird je nachdem, ob die Proposition an der ersten Stelle des Wortes vorhanden ist, $\top$ oder $\bot$ ausgegeben. Bei Disjunktionen und Konjunktionen werden die einzelnen Teilformeln ausgewertet; aus den daraus resultierenden Wahrheitswerten wird über die Verbandsoperationen $\sqcup$ und $\sqcap$ der Wahrheitswert für die gesamte Formel bestimmt. Bei der Negation einer Formel wird der Wahrheitswert komplementiert. Die Funktionen für beiden starken Next-Operatoren suchen nach Präfixen des Wortes, die den regulären Ausdruck erfüllen und berechnen die Semantikfunktion für den Rest des Wortes und der RLTL-Teilformel der Operators. Die schwachen Next-Operatoren können über die Dualitätsgesetze mit Hilfe der starken Next-Operatoren beschrieben werden. Die Semantikfunktionen der vier Power-Operatoren sind über die Abrollungen der Formeln beschrieben, wobei für jeden Power-Operator ein anderer Next-Operator verwendet wird.

3.3 Erläuterungen zur Semantik

Nach der formalen Definition der Auswertungsfunktionen, wird in diesem Abschnitt begründet, warum diese dem Sinn einer vierwertigen Semantik entsprechen, was bedeutet, dass sie korrekt angeben, ob ein Wort eine Formel erfüllt, und die Endgültigkeit dieser Entscheidung im Hinblick auf eine mögliche Verlängerung des Wortes ebenfalls dem Rückgabewert zu entnehmen ist.
Dass der Wahrheitswert bei der RLTL-Formel $\varnothing$ endgültig ist, folgt direkt aus der Definition für RLTL. Wenn eine Proposition in RLTL erfüllt ist, ist der Rest des Wortes nicht mehr von Interesse. Ist sie hingegen nicht erfüllt, kann sie auch durch eine Verlängerung

des Wortes nicht mehr erfüllt werden. Da das Ergebnis demzufolge bereits nach der ersten Position im Wort unwiderruflich feststeht, werden in der Semantik die endgültigen Wahrheitswerte $\top$ und $\bot$ verwendet. Wenn die betrachtete Formel eine Disjunktion oder eine Konjunktion ist, werden die Resultate der Auswertungen der Teilformeln über die Operationen des $\mathbb{B}_4$ verknüpft, bei der Negation wird der Wahrheitswert komplementiert. Die Korrektheit der Semantikfunktion für diese Operatoren hängt also ausschließlich von der Richtigkeit der Auswertung der Teilformeln ab.

Bevor argumentiert wird, warum die Semantik der Next-Operatoren die Anforderungen erfüllt, wird noch einmal detaillierter auf die Semantik der regulären Ausdrücke eingegangen, welche stets in Kombination mit den Next-Operatoren auftreten. Bei Propositionen werden die endgültigen Wahrheitswerte verwendet, da sich die Propositionen ausschließlich auf die erste Position des betrachteten Wortes beziehen, sodass Verlängerungen des Wortes keine Auswirkungen auf die Auswertung dieses Zeichens haben. Zusätzlich wird gefordert, dass das Wort nur aus einem Zeichen besteht. Falls es mehr Zeichen sind, ist die Proposition unwiderruflich nicht erfüllt. Eine mögliche, zu einem späteren Zeitpunkt erfolgende, Verlängerung des Eingabewortes um weitere Zeichen verursacht keine Probleme bei der Auswertung der Proposition, da diese Zeichen noch andere Teile des regulären Ausdrucks oder der RLTL-Formel erfüllen müssen.

Die Korrektheit der Disjunktion in regulären Ausdrücken folgt wiederum aus der Auswertung der beiden Teilausdrücke. Die Definition der Konkatenation ist strukturell genauso aufgebaut, wie die Definition des existentiellen starken Next-Operators, weswegen sich die Begründungen, die im Folgenden für den starken Next-Operator verwendet werden, direkt auf diesen Operator übertragen lassen. Für den Kleene-Operator sind diverse Fälle zu unterscheiden. Zunächst gibt es die Möglichkeit, dass zu Beginn des Wortes einige Male α gilt und danach in einem zweiten Teil des Wortes noch weitere Zeichen folgen. Um zu verhindern, dass diese Zeichen für den ersten Teil des Wortes verwendet werden, wenn die weiteren Zeichen bereits für den hinteren Teil verwendet werden, erfolgt die Auswertung der Teilworte, die jeweils ein Modell für α sind, zweiwertig, indem die Funktion $\mathfrak{F}$ verwendet wird. Diesen Fall deckt die zweite Zeile in der Definition ab. Dabei bestimmt $\mathcal{Y}$ die Länge der einzelnen Teilworte, in denen α gilt. Anschließend kann der Rest des Wortes den Ausdruck β komplett oder zum Teil erfüllen, was in den Wahrheitswerten $\top$ und $\bot^p$ resultiert. Diesen Fall deckt die Auswertung $[\![w^i \models_{\text{RE}} \beta]\!]$ ab. Es ist ebenfalls möglich, dass noch einmal α gilt. Diesmal ist, da sämtliche Zeichen des Wortes verwendet werden, auch eine teilweise Auswertung des Ausdrucks α möglich, sodass die Funktion $\mathfrak{F}$ hier nicht zum Einsatz kommt. Da in diesem Fall β nicht erfüllt wurde, kann der gesamte Ausdruck nicht komplett erfüllt sein. Daher ist der größte Wahrheitswert, der vorkommen kann, $\bot^p$. In der Formel wird daher $\bot^p \sqcap [\![w^i \models_{\text{RE}} \alpha]\!]$ berechnet. Die beiden zuletzt beschriebenen Auswertungen werden mit dem Join-Operator verknüpft,

damit bei einer Auswertung des β-Teils zu $\top$ gleiches für die Auswertung des gesamten Kleene-Operators gilt.
Die Semantik ist so konzipiert, dass bereits reguläre Ausdrücke zu $\top$ ausgewertet werden können. Da RLTL-Formeln aber nicht allein aus derartigen Ausdrücken bestehen können, dürfen sie auch trotz der Erfüllung des regulären Ausdrucks als Gesamtformel nicht erfüllt sein, wenn die zugrunde liegende Semantik nicht antizipatorisch ist und die Restformel keine Tautologie ist. Die Semantikfunktionen für die RLTL-Operatoren müssen daher die Auswertung des regulären Ausdrucks im Kontext der RLTL-Teilformeln betrachten. Da sich die Power-Operatoren abrollen und somit über Disjunktionen, Konjunktionen und die Next-Operatoren ausdrücken lassen und reguläre Ausdrücke somit ausschließlich vor einem der vier Next-Operatoren vorkommen, werden für die folgenden Begründungen nur die Next-Operatoren betrachtet. Da die schwachen Next-Operatoren über die Dualitätsgesetze definiert sind, ist es zudem ausreichend, die Begründungen für die beiden starken Operatoren anzugeben.
Sowohl für den existentiellen starken als auch für den universellen starken Next-Operator muss der Sonderfall betrachtet werden, dass das gesamte Wort den regulären Ausdruck α komplett oder nur zum Teil erfüllt, sodass jeweils keine Zeichen für die RLTL-Teilformel φ vorhanden sind. In diesem Fall muss die Ausgabe der Semantikfunktion $\bot^p$ sein, da die Semantik für beide Operatoren verlangt, dass φ auf jeden Fall erfüllt sein muss. Gleichzeitig ist diese Auswertung nicht endgültig, da eine Erfüllung von φ durch ein verlängertes Wort die Auswertung $\top$ ergeben würde.
Bei dem existentiellen starken Next-Operator wird der Sonderfall, dass kein Suffix des Wortes die Teilformel φ erfüllt, durch die Berechnung von $\bot^p \sqcap [\![w \models_{\mathrm{RE}} \alpha]\!]$ abgedeckt. Falls der reguläre Ausdruck nicht erfüllt ist und somit auch nach einer Verlängerung des Wortes nicht mehr erfüllt werden kann und die Semantikfunktion für diesen Ausdruck daher $\bot$ berechnet, ändert sich dieser Wahrheitswert durch die in der Definition enthaltenen Meet-Operation genauso wenig, als wenn der reguläre Ausdruck noch erfüllbar wäre, wenn das Wort verlängert wird, und die Semantikfunktion $\bot^p$ zurückgeben würde. Wenn das Eingabewort den regulären Ausdruck hingegen komplett erfüllt und dieser daher zu $\top$ auswertet, wird über die Meet-Operation der Wahrheitswert $\bot^p$ berechnet, was genau der Anforderung entspricht.
Wenn das Wort stattdessen aufgeteilt wird, sodass ein Präfix ein Modell für den regulären Ausdruck α ist und der hintere Teil des Wortes den Teilausdruck φ erfüllt, muss beachtet werden, dass es nicht mehr möglich ist, dass eine Verlängerung des Wortes die Auswertung des regulären Ausdrucks verändert. Der einzige Fall, der diesbezüglich kritisch ist, ist eine Auswertung des regulären Ausdrucks zu $\bot^p$. Durch die Funktion $\mathfrak{F}$ wird dieser Wert jedoch in solch einem Fall zu $\bot$ verändert. Bei einer Auswertung des regulären Ausdrucks zu $\top$ ist es aufgrund der Forderung, dass Propositionen nur dann zu $\top$ auswerten,

wenn das hierzu betrachtete Teilwort ein Zeichen lang ist, im Allgemeinen nicht möglich, weitere Zeichen für die Auswertung des Ausdrucks zu verwenden, ohne einen von $\bot$ verschiedenen Wahrheitswert zu erhalten. Da es nach der Definition des existentiellen Next-Operators eine Position im Wort geben muss, sodass der vordere Teil den regulären Ausdruck und der hintere Teil die RLTL-Formel erfüllt, kann in der vierwertigen Semantik der Join über die Wahrheitswerte, die sich bei allen möglichen Aufteilungen des Wortes aus der Auswertung der Teilformeln ergeben, berechnet werden, sodass der größte vorkommende Wahrheitswert die Auswertung des Next-Operators bestimmt. Aufgrund der Berechnung $\mathfrak{F}([\![^i w \models_{\text{RE}} \alpha]\!]) \sqcap [\![w^i \models \varphi]\!]$ muss der reguläre Ausdruck vollständig erfüllt sein, damit sich ein von $\bot$ verschiedener Wahrheitswert ergeben kann. Nur in diesem Fall hat der über $[\![w^i \models \varphi]\!]$ berechnete Wahrheitswert Einfluss auf die Auswertung des existentiellen Next-Operators.

Der universelle starke Next-Operator $\alpha ;_{\Pi} \varphi$ verlangt, dass nach allen Teilen des Wortes, die den Ausdruck α erfüllen, auch φ gilt. Es existieren verschiedene Möglichkeiten, wie eine derartige Formel teilweise oder vollständig erfüllt sein kann. Zum einen gibt es den Fall, dass keine Zeichen für φ verwendet werden, was in der Definition durch $\mathfrak{B}([\![w \models_{\text{RE}} \alpha]\!])$ abgedeckt wird. Falls α dann nicht erfüllt wird, ist die Ausgabe dieser Funktion $\top$, da keine weiteren Anforderungen an die Teilformel φ gestellt werden. Durch die Meet-Operation, welche die Ergebnisse der einzelnen Fälle miteinander verrechnet, ist dieser Wahrheitswert nicht automatisch die Ausgabe des Next-Operators. Wenn das Wort ein Modell für den regulären Ausdruck ist, gibt die Funktion $\mathfrak{B}$ den Wahrheitswert $\bot^p$ zurück, da die Gesamtformel ohne eine Erfüllung des hinteren Teilausdrucks nicht erfüllt ist, weitere Eingabezeichen aber im Allgemeinen zu einer endgültigen Verletzung oder Erfüllung der Formel führen können. Wenn die Auswertung des regulären Ausdrucks $\bot^p$ liefert, gilt die gleiche Argumentation. Da der Operator eine universelle Semantik hat und bei allen Unterteilungen des Wortes, bei denen der reguläre Ausdruck erfüllt ist, die Erfüllung von φ fordert, wird der Wahrheitswert des universellen starken Next-Operators über die Meet-Operation aus den Teilauswertungen errechnet. Das Ergebnis ist dabei bezüglich der Ordnung des Verbandes $\mathbb{B}_4$ der kleinste Wahrheitswert, der bei den einzelnen Unterteilungen auftritt. Wenn einmal der Fall auftritt, dass nach einem erfüllten regulären Ausdruck die weiteren Zeichen des Wortes φ definitiv verletzen, ist die Auswertung des Next-Operators daher direkt $\bot$. Durch $\mathfrak{F}([\![^i w \models_{\text{RE}} \alpha]\!]) \rightarrow [\![w^i \models \varphi]\!]$ wird $[\![w^i \models \varphi]\!]$ nur dann ausgewertet, wenn der reguläre Ausdruck erfüllt ist, ansonsten wertet die Implikation zu $\top$ aus, sodass dies, da zur Verknüpfung aller Werte der Meet-Operator verwendet wird, keine Auswirkungen auf den für den universellen Next-Operator berechneten Wahrheitswert hat.

Die vierwertige Semantik für die Power-Operatoren wurde bislang nur über die Abrollungen definiert. Es ist jedoch ebenfalls möglich, die Semantik direkt zu definieren. Da

die Formeln dafür jedoch komplexer sind, wird exemplarisch nur die Semantik für den starken Power-Operator angegeben. Es gilt:

$$[\![w \models \varphi \mid \alpha \gg \psi]\!] = \bigsqcup_{0 \leq i < |w|} \Big(\Big([\![w^i \models \psi]\!] \sqcup \big(\bot^p \sqcap [\![w^i \models_{\text{RE}} \alpha]\!] \sqcap [\![w^i \models \varphi]\!] \big) \Big) \sqcap \bigsqcup_{Y \in \mathcal{Y}({}^i w)} \Big(\bigsqcap_{\langle y \rangle \in Y} \mathfrak{F}([\![{}^{y_1} w^{y_0} \models_{\text{RE}} \alpha]\!]) \sqcap [\![w^{y_0} \models \varphi]\!] \Big) \Big)$$

Die Funktion ermittelt eine Position im Wort, sodass im vorderen Teil des Wortes an einer Folge von Positionen, die mittels $\mathcal{Y}$ bestimmt wird, der reguläre Ausdruck α und die RLTL-Formel φ gelten. Eine weitere Forderung ist, dass die Länge des Teilwortes, welches α erfüllen muss, festgelegt ist. Dies beschreibt gerade die wiederholte Erfüllung des Pflicht-Teils des Power-Operators zu den von der Verzögerung bestimmten Zeitpunkten. Die Funktion $\mathfrak{F}$ wird verwendet, damit sich die Teilworte der Eingabe, die die regulären Ausdrücke erfüllen, nicht überschneiden und keine Zeichen des Wortes zur Auswertung eines regulären Ausdrucks verwendet werden, wenn bereits Zeichen des Wortes für andere Teilformeln verwendet werden. Der hintere Teil des Eingabewortes w kann zum einen ψ vollständig oder zum Teil erfüllen, wodurch für eine Erfüllung des gesamten Operators die restlichen Teilformeln nicht mehr relevant sind. Der andere Fall, den die Formel in der ersten Zeile betrachtet, berechnet die Semantik für den Fall, dass die Zeichen des Eingabewortes gerade soweit vorliegen, dass der reguläre Ausdruck nicht vollständig erfüllt ist, dies mit zusätzlichen Zeichen aber noch geschehen kann. Der größte Wahrheitswert, der an dieser Stelle Auftreten kann, ist $\bot^p$, da, selbst wenn α und φ erfüllt sein sollen, der starke existentielle Power-Operator als ganzes nicht erfüllt werden kann, solange ψ nicht erfüllt wurde.

Zum Abschluss dieses Kapitels werden einige Worte für gegebene Formeln beispielhaft ausgewertet, um die Sinnhaftigkeit der Definition der vierwertigen Semantik zu veranschaulichen. Die Definitionen der in dem Beispiel verwendeten schwachen Next-Operatoren lassen sich unmittelbar aus Definition 3.3 und den Dualitätsbeziehungen ableiten. Es gilt:

$$\overline{\mathfrak{B}}([\![w \models_{\text{RE}} \alpha]\!]) = \begin{cases} \bot & [\![w \models_{\text{RE}} \alpha]\!] = \bot \\ \top^p & \text{sonst} \end{cases}$$

$$\begin{aligned} [\![w \models \alpha \overline{;_\Sigma} \varphi]\!] &= [\![w \models \neg(\alpha ;_\Pi \neg\varphi)]\!] \\ &= \overline{\mathfrak{B}([\![w \models_{\text{RE}} \alpha]\!]) \sqcap \bigsqcap_{0 < i < |w|} \Big(\mathfrak{F}([\![{}^i w \models_{\text{RE}} \alpha]\!]) \to \overline{[\![w^i \models \varphi]\!]} \Big)} \end{aligned}$$

$$
\begin{aligned}
&= \overline{\mathfrak{B}([\![w \models_{\mathrm{RE}} \alpha]\!])} \sqcup \bigsqcup_{0<i<|w|} \left(\overline{\overline{\mathfrak{F}([\![^i w \models_{\mathrm{RE}} \alpha]\!])} \sqcup \overline{[\![w^i \models \varphi]\!]}}\right) \\
&= \overline{\mathfrak{B}}([\![w \models_{\mathrm{RE}} \alpha]\!]) \sqcup \bigsqcup_{0<i<|w|} \left(\mathfrak{F}([\![^i w \models_{\mathrm{RE}} \alpha]\!]) \sqcap [\![w^i \models \varphi]\!]\right) \\
[\![w \models \alpha \overline{;_\Pi} \varphi]\!] &= [\![w \models \neg(\alpha ;_\Sigma \neg\varphi)]\!] \\
&= \overline{(\bot^p \sqcap [\![w \models_{\mathrm{RE}} \alpha]\!]) \sqcup \bigsqcup_{0<i<|w|} \left(\mathfrak{F}([\![^i w \models_{\mathrm{RE}} \alpha]\!]) \sqcap \overline{[\![w^i \models \varphi]\!]}\right)} \\
&= \left(\overline{\bot^p} \sqcup \overline{[\![w \models_{\mathrm{RE}} \alpha]\!]}\right) \sqcap \bigsqcap_{0<i<|w|} \left(\overline{\mathfrak{F}([\![^i w \models_{\mathrm{RE}} \alpha]\!])} \sqcup \overline{\overline{[\![w^i \models \varphi]\!]}}\right) \\
&= ([\![w \models_{\mathrm{RE}} \alpha]\!] \to \top^p) \sqcap \bigsqcap_{0<i<|w|} \left(\mathfrak{F}([\![^i w \models_{\mathrm{RE}} \alpha]\!]) \to [\![w^i \models \varphi]\!]\right)
\end{aligned}
$$

Die schwachen Next-Operatoren unterscheiden sich ausschließlich in dem Fall, bei dem alle Zeichen des Wortes für den regulären Ausdruck verwendet werden, von den starken Next-Operatoren. In diesem Fall wird das Wort zu $\top^p$ ausgewertet, da die Semantik für die schwachen Next-Operatoren verlangt, dass die spezifizierten Verzögerungen nur eingehalten werden, wenn die Positionen im Wort tatsächlich existieren. Der hintere Teil der jeweiligen Formeln, der den Fall abdeckt, dass ein Suffix des Wortes den RLTL-Teilausdruck erfüllt, ist identisch zu der Definition der starken Next-Operatoren.
Das Beispiel betrachtet nun vier RLTL-Formeln mit unterschiedlichen Next-Operatoren, die für das Wort $\{a\}\{a,b\}\{a\}\{b\}\{b\}$ Schritt für Schritt gemäß der in diesem Kapitel definierten Semantik für RLTL auf endlichen Worten ausgewertet werden. Die folgende Tabelle enthält die von den Funktionen berechneten Wahrheitswerte für alle nichtleeren Präfixe des Wortes.

	$\{a\}$	$\{a,b\}$	$\{a\}$	$\{b\}$	$\{b\}$
`(a;a)*(a;b)` $;_\Sigma a$	$\bot^p$	$\bot^p$	$\top$	$\top$	$\top$
`(a;a)*(a;b)` $;_\Pi a$	$\bot^p$	$\bot^p$	$\bot^p$	$\bot^p$	$\bot$
`(a;a)*(a;b)` $\overline{;_\Sigma} a$	$\top^p$	$\top^p$	$\top$	$\top$	$\top$
`(a;a)*(a;b)` $\overline{;_\Pi} a$	$\top^p$	$\top^p$	$\top^p$	$\top^p$	$\bot$

Während die erste Formel nach dem dritten Zeichen des Wortes erfüllt ist, da der hintere Teil des Kleene-Stern-Ausdrucks sowie die RLTL-Proposition a erfüllt wurden, ist die zweite Formel noch nicht erfüllt, da der universelle Next-Operator verlangt, dass in Anschluss an jede Position, an der das Wort den regulären Ausdruck erfüllt, der RLTL-Teilausdruck, in diesem Fall ist das a, gilt. Nun erfüllen die Teile $\{a\}\{a,b\}$ und $\{a\}\cdots$ des Eingabewortes zwar den regulären Ausdruck und die RLTL-Formel a, dennoch gilt $[\![\{a\}\{a,b\}\{a\} \models_{\mathrm{RE}} \texttt{(a;a)*(a;b)}]\!] = \bot^p$, da der reguläre Ausdruck nach weiteren Zeichen in der Eingabe noch einmal gelten kann, was wiederum erfordert, dass a anschlie-

ßend noch einmal gilt. Da $w_3 = \{b\}$ gilt, tritt genau dieser Fall ein. Dann gilt jedoch $a \notin w_4 = \{b\}$, sodass die Formel nicht erfüllt ist und von der Semantikfunktion daher $\bot$ berechnet wird.
Die schwachen Next-Operatoren verhalten sich analog, der einzige Unterschied ist, dass jeweils $\top^p$ ausgegeben wird, wenn innerhalb der Verzögerungen der Next-Operatoren Propositionen an Positionen in einem Wort verlangt werden, die noch nicht vorliegen.

In diesem Kapitel wurde eine vierwertige Semantik für RLTL definiert, die die Auswertung von endlichen Worten beschreibt. Für die Laufzeitverifikation können Monitore konstruiert werden, welche die entsprechenden Wahrheitswerte ausgeben. Wie sich zu diesem Zweck alternierende Mealy-Maschinen generieren lassen, wird im folgenden Kapitel beschrieben.

4 Monitorkonstruktion

In diesem Kapitel wird beschrieben, wie zu einer RLTL-Formel ein Monitor konstruiert werden kann, der einen Lauf des überwachten Systems gemäß der vierwertigen Semantik auf endlichen Worten, die im vorherigen Kapitel definiert wurde, auswertet. Nach der Definition des zum Einsatz kommenden Automatentyps in Abschnitt 4.1 wird die Generierung eines Monitors für RLTL-Formeln, welche auf dem Bottom-up-Prinzip basiert, erläutert. Hierfür werden in Abschnitt 4.2 die Transitionsfunktionen für die einzelnen Operatoren in regulären Ausdrücken und RLTL-Formeln angegeben, mit denen es möglich ist, die Nachfolgezustände und Ausgabewerte des Monitors zu berechnen. Dieses auf dem Ansatz des Formel-rewritings basierende Verfahren wird zudem an einem Beispiel veranschaulicht. Abschließend wird in Abschnitt 4.3 die Korrektheit des Prozederes bewiesen, wo außerdem dessen Komplexität analysiert wird.

4.1 Automatenmodell

In praktischen Anwendungsfällen für Runtime-Verification wird ein System zur Laufzeit von einem Monitor überwacht, um ein Fehlverhalten zu detektieren oder um zu verifizieren, dass es die spezifizierten Anforderungen erfüllt. In dieser *online*-Variante von RV liegt stets nur ein endlicher Präfix eines unendlich langen Wortes vor, der vom Monitor analysiert werden kann. Anstelle von ω-Automaten, die unendliche Worte akzeptieren oder verwerfen, werden also endliche Automaten eingesetzt. Besonders geeignet sind alternierende Mealy-Maschinen, die auch bereits als Monitor eingesetzt werden, um Worte bezüglich LTL-Formeln gemäß einer vierwertigen Semantik auszuwerten. Bevor die Automaten formal definiert werden, wird noch die Definition der positiven booleschen Kombination benötigt.

Definition 4.1 (Positive boolesche Kombination)**.** *Sei Q eine Menge. Die* positive boolesche Kombination $B^+(Q)$ *von Elementen dieser Menge ist wie folgt induktiv definiert.*

- $true \in B^+(Q)$
- $false \in B^+(Q)$
- $Q \subseteq B^+(Q)$
- $\forall\, a, b \in B^+(Q) : a \vee b \in B^+(Q)$ *und* $a \wedge b \in B^+(Q)$

Definition 4.2 (Alternierende Mealy-Maschine)**.** *Eine alternierende Mealy-Maschine ist ein* 5*-Tupel* $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0)$*. Dabei ist*

- Q *die endliche Zustandsmenge,*
- Σ *das Eingabealphabet,*
- Γ *das Ausgabealphabet,*
- $\delta : Q \times \Sigma \to B^+(Q \times \Gamma)$ *die Transitionsfunktion,*
- $q_0 \in B^+(Q)$ *die positive boolesche Kombination initialer Zustände.*

Auf die Definition akzeptierender Zustände wurde verzichtet, da der Automat ebenso über ein Element im Ausgabealphabet zum Ausdruck bringen kann, ob er ein Wort akzeptiert.
Das hier für die Monitorkonstruktion verwendete Automatenmodell verbindet das Konzept der Alternation [Chandra und Stockmeyer, 1976], mit dem Prinzip der Mealy-Maschine [Mealy, 1955]. Bei einem deterministischen Mealy-Automaten gibt die Transitionsfunktion beim Zustandsübergang ein Symbol aus einem Ausgabealphabet aus, wobei das Ausgabesymbol vom Eingabesymbol abhängt. Durch das Konzept der Alternation wird die Zustandsübergangsfunktion derart erweitert, dass es nicht wie bei deterministischen Automaten maximal einen Nachfolgezustand für eine Eingabe geben kann, sondern eine aus Disjunktionen und Konjunktionen bestehende boolesche Kombination erreicht werden kann. Chandra und Stockmeyer haben zudem bewiesen, dass alternierende endliche Automaten genauso mächtig sind, wie deterministische Automaten. Zu jedem endlichen alternierenden Automaten mit n Zuständen existiert ein äquivalenter deterministischer Automat mit $\xi(n)$ Zuständen, wobei sich die scharfe Schranke durch

$$\log_2 \xi(n) = \binom{n}{\lfloor \frac{n}{2} \rfloor} \left(1 + O\left(\frac{\log n}{n}\right)\right)$$

bestimmen lässt. Der Wert $\xi(n)$ steht dabei für die Anzahl der monotonen booleschen Funktionen mit n Eingaben.

4.2 Generierung von Automaten für RLTL-Formeln

In diesem Abschnitt wird beschrieben, wie RLTL-Formeln zwecks der Monitorgenerierung in alternierende Mealy-Maschinen übersetzt werden können. Die Zustände des Automaten sind dabei RLTL-Formeln, Eingaben stammen aus der Potenzmenge der Propositionen und Ausgaben des Monitors sind Elemente aus dem Verband $\mathbb{B}_4$. Nachfolgezustände einer RLTL-Formel sind in einer positiven booleschen Kombination von RLTL-Formeln enthalten. Die in dieser Kombination enthaltenen Ausgabewerte lassen sich zu

einem einzelnen Wahrheitswert vereinfachen, indem anstelle der booleschen Operatoren $\vee$ und $\wedge$ mit $\sqcup$ und $\sqcap$ die Operanden des Verbandes $\mathbb{B}_4$ verwendet werden. Somit ergibt sich für die Transitionsfunktion des Monitors mit RLTL-Formeln als Zuständen die Signatur $\delta : \text{RLTL} \times \Sigma \to B^+(Q) \times \mathbb{B}_4$.
Alternierende Mealy-Maschinen eignen sich besonders gut als Monitor, da positive boolesche Kombinationen von Nachfolgezuständen direkt in das Automatenmodell übertragen werden können. Außerdem ermöglicht die Ausgabe eines Symbols nach jedem gelesenen Zeichen die schrittweise Auswertung einer Formel, was die vierwertige Semantik auch erfordert, da für die Auswertung in jedem Schritt der aktuell vorhandene Präfix des Wortes verwendet wird.
Die Bestimmung der Nachfolgezustände und Ausgabesymbole erfolgt dabei für jedes mögliche Eingabesymbol nach dem folgenden Prinzip: Beginnend mit der Wurzel wird für jeden Operator im Syntaxbaum der RLTL-Formel überprüft, welche Teilformeln für die Eingabe ausgewertet werden müssen. Dieses Verfahren wird mit den entsprechenden Formeln fortgesetzt, bis jeweils die Blattknoten erreicht sind. Dort wird eine neue Formel Ψ' für die ursprüngliche Formel Ψ gebildet, die so aufgebaut ist, dass alle $\sigma_1\sigma_2 \cdots \in \Sigma^*$ genau dann ein Modell für Ψ' sind, wenn die Worte $\sigma_0\sigma_1\sigma_2 \cdots \in \Sigma^*$ ein Modell für Ψ sind, wobei σ_0 das Eingabesymbol ist. Mit Ψ' als Blattknoten im Syntaxbaum werden auf dem Pfad vom Blatt zur Wurzel in Abhängigkeit der Operatoren gegebenenfalls weitere Transformationen durchgeführt, um die komplette Formel für den Nachfolgezustand zu bestimmen.
Die Definition der Transitionsfunktion für die alternierende Mealy-Maschine wird zwecks der Übersichtlichkeit in zwei Teile aufgeteilt. Zuerst wird $\delta : \text{RE} \times \Sigma \to B^+(\mathbb{B}_4 \times \text{RE})$ für reguläre Ausdrücke definiert, danach folgt $\delta : \text{RLTL} \times \Sigma \to B^+(\mathbb{B}_4 \times \text{RLTL})$ für die RLTL-Operatoren, wobei einige Operatoren über Äquivalenzregeln definiert sind. Die Definition der Funktion δ ist doppeldeutig, es folgt jedoch immer aus dem Kontext, welche der beiden Funktionen verwendet wird.
Da die Auswertung einer Formel Schritt für Schritt erfolgt, wird immer nur ein Zeichen als Eingabe genommen, woraufhin die Transitionsfunktion die Ausgabe und die im nächsten Schritt noch zu erfüllende Formel berechnet. Die Transitionsfunktionen für reguläre Ausdrücke und für RLTL-Formeln werden nun formal definiert. Einige Anmerkungen zur Funktionsweise werden jeweils im Anschluss an die Definitionen gegeben. Dass ein für die FRLTL_4-Semantik generierter Monitor mit den entsprechenden Transitionen die korrekten Ausgaben berechnet, wird gesondert in Abschnitt 4.3 gezeigt.

Definition 4.3 (Transitionsfunktion für reguläre Ausdrücke)**.** *Seien $\alpha, \alpha', \beta \in RE$ reguläre Ausdrücke und $\sigma \in \Sigma$ ein Eingabesymbol. Die Transitionsfunktion $\delta : RE \times \Sigma \to B^+(\mathbb{B}_4 \times RE)$ bildet auf die folgenden Disjunktionen ab.*

$$\delta(p,\ \sigma) = \begin{cases} (\top, true) & p \in \sigma \\ (\bot, false) & p \notin \sigma \end{cases}$$

$$\delta(\alpha + \beta,\ \sigma) = \delta(\alpha,\ \sigma) \vee \delta(\beta,\ \sigma)$$

$$\delta(\alpha ; \beta,\ \sigma) = \bigvee_{(b,\alpha') \in \delta(\alpha,\ \sigma)} \varrho^{i}_{\mathrm{RE}}(b, \alpha', \alpha, \beta)$$

$$\text{mit } \varrho^{i}_{\mathrm{RE}}(b, \alpha', \alpha, \beta) = \begin{cases} (\bot_p, \beta) & b = \top \\ (\bot_p, \alpha' ; \beta) & b = \bot^p \\ (\bot, false) & \text{sonst} \end{cases}$$

$$\delta(\alpha * \beta,\ \sigma) = \delta(\beta,\ \sigma) \vee \bigvee_{(b,\alpha') \in \delta(\alpha,\ \sigma)} \varrho^{*}_{\mathrm{RE}}(b, \alpha', \alpha, \beta)$$

$$\text{mit } \varrho^{*}_{\mathrm{RE}}(b, \alpha', \alpha, \beta) = \begin{cases} (\bot_p, \alpha * \beta) & b = \top \\ (\bot_p, \alpha' ; (\alpha * \beta)) & b = \bot^p \\ (\bot, false) & \text{sonst} \end{cases}$$

Gemäß dieser Transitionsfunktion sind die Nachfolgezustände von regulären Ausdrücken eine Disjunktion von regulären Ausdrücken, was in der Automatentheorie als Nichtdeterminismus interpretiert wird. Auf propositioneller Ebene wird in Abhängigkeit davon, ob die vom regulären Ausdruck geforderte Proposition Teil der Eingabe ist, deterministisch ein Nachfolgezustand bestimmt. Im Falle von $true$ symbolisiert dies, dass der reguläre Teilausdruck der RLTL-Formel erfüllt wurde, im Falle von $false$ bedeutet dies, dass der Automat den regulären Ausdruck verwirft.

Ist der reguläre Ausdruck eine Veroderung, wird die Transitionsfunktion rekursiv für beide Teilausdrücke aufgerufen.

Die letzten beiden Operatoren machen Gebrauch von den Funktionen ϱ_{RE}, die verwendet werden, um das Formel-rewriting zu ermöglichen, da in Abhängigkeit von dem durch den rekursiven Aufruf der Transitionsfunktion erhaltenen Wahrheitswert verschiedene Fälle bei der Konstruktion der Formel im Nachfolgezustand unterschieden werden müssen.

Beim Konkatenationsoperator muss zunächst der vordere Teil des regulären Ausdrucks erfüllt werden, sodass die Transitionsfunktion rekursiv auf diesen Teil angewendet wird. Wird hier $(\top, true)$ zurückgegeben, ist der entsprechende Teilausdruck erfüllt und komplett ausgewertet, sodass als Restausdruck, der im Folgenden noch erfüllt werden muss, β verbleibt. Als Wahrheitswert wird $\bot^p$ zurückgegeben, da der reguläre Ausdruck nicht komplett erfüllt ist, dies aber noch möglich ist, wenn weitere Zeichen gelesen werden. Ist

der Rückgabewert hingegen $(\bot^p, \alpha')$, ist der vordere Teil der Konkatenation nicht komplett erfüllt, sodass der Restausdruck α' verbleibt, der noch erfüllt werden kann, wenn mehr Zeichen gelesen werden. Zur weiteren Verarbeitung von anderen Funktionen, die die Transitionsfunktion für den Konkatenationsoperator aufgerufen haben, werden die Teilausdrücke zur neuen Konkatenation $\alpha' ; \beta$ zusammengefügt.
Der letzte Fall betrachtet die Möglichkeit, dass der reguläre Ausdruck α verletzt wurde. Dann ist die komplette Formel auch dann nicht mehr erfüllbar, wenn noch beliebige weitere Zeichen folgen. Dies wird für die weitere Verarbeitung signalisiert, indem in $(\bot, false)$ der Wahrheitswert $\bot$ zurückgegeben wird.
Der Kleene-Stern-Ausdruck $\alpha * \beta$ wird erfüllt, wenn β erfüllt wird und zuvor beliebig oft direkt hintereinander der Ausdruck α erfüllt wurde. Um β zu überprüfen, genügt ein Aufruf der Transitionsfunktion auf diesen Teilausdruck. Formel-rewriting wird beim Kleene-Operator verwendet, um die Restformeln zu bilden, wenn Zeichen der Eingabe Teile des regulären Ausdrucks α erfüllen. In den Fällen $b = \top$ und $b = \bot$ verhält sich die Funktion ϱ^*_{RE} analog zu ϱ^i_{RE}. Tritt der Fall $b = \bot^p$ auf, müssen zukünftig gelesene Zeichen den Rest dieses Ausdrucks erfüllen, bevor der Teilausdruck β der Ursprungsformel erfüllt wird. Dies wird gerade durch den neu gebildeten Ausdruck $\alpha' ; (\alpha * \beta)$ erreicht.
Nun können die Transitionsfunktionen für die RLTL-Operatoren definiert werden, wobei zur Berechnung der Nachfolgezustände und der Ausgaben bei den Next-Operatoren auf die in Definition 4.3 angegebenen Funktionen für reguläre Ausdrücke zurückgegriffen wird.

Definition 4.4 (Transitionsfunktionen für RLTL-Formeln)**.** *Seien $\alpha, \alpha' \in RE,\ \varphi, \psi \in RLTL$ und $\sigma \in \Sigma$. Die Transitionsfunktion $\delta : RLTL \times \Sigma \to B^+(\mathbb{B}_4 \times RLTL)$ ist wie folgt definiert:*

$$
\begin{aligned}
\delta(\varnothing,\ \sigma) &= (\bot, \varnothing) \\
\delta(p,\ \sigma) &= \begin{cases} (\top, \neg\varnothing) & p \in \sigma \\ (\bot, \varnothing) & p \notin \sigma \end{cases} \\
\delta(\varphi \vee \psi,\ \sigma) &= \delta(\varphi,\ \sigma) \vee \delta(\psi,\ \sigma) \\
\delta(\varphi \wedge \psi,\ \sigma) &= \delta(\varphi,\ \sigma) \wedge \delta(\psi,\ \sigma) \\
\delta(\neg\varphi,\ \sigma) &= \overline{\delta(\varphi,\ \sigma)}, \textit{wobei } \overline{(b, \varphi)} = (\bar{b}, \neg\varphi) \\
\delta(\alpha\ ;_\Sigma\ \varphi,\ \sigma) &= \bigvee_{(b,\alpha') \in \delta(\alpha,\ \sigma)} \varrho^{\,;_\Sigma}(b, \alpha', \alpha, \varphi)
\end{aligned}
$$

$$
\textit{mit } \varrho^{\,;_\Sigma}(b, \alpha', \alpha, \varphi) = \begin{cases} (\bot_p, \varphi) & b = \top \\ (\bot_p, \alpha'\ ;_\Sigma\ \varphi) & b = \bot^p \\ (\bot, \varnothing) & \textit{sonst} \end{cases}
$$

$$\delta(\alpha \;;_\Pi \varphi,\ \sigma) = \bigwedge_{(b,\alpha') \in \delta(\alpha,\ \sigma)} \varrho^{\;;_\Pi}(b, \alpha', \alpha, \varphi)$$

$$\textit{mit } \varrho^{\;;_\Pi}(b, \alpha', \alpha, \varphi) = \begin{cases} (\bot_p, \varphi) & b = \top \\ (\bot_p, \alpha' \;;_\Pi \varphi) & b = \bot^p \\ (\top, \neg\varnothing) & \textit{sonst} \end{cases}$$

$$\delta(\alpha \overline{;_\Sigma} \varphi,\ \sigma) = \delta(\neg(\alpha \;;_\Pi \neg\varphi),\ \sigma)$$

$$\delta(\alpha \overline{;_\Pi} \varphi,\ \sigma) = \delta(\neg(\alpha \;;_\Sigma \neg\varphi),\ \sigma)$$

$$\delta(\varphi \mid \alpha \gg \psi,\ \sigma) = \delta(\psi \vee (\varphi \wedge \alpha \;;_\Sigma \varphi \mid \alpha \gg \psi),\ \sigma)$$

$$\delta(\varphi \mid \alpha > \psi,\ \sigma) = \delta(\psi \vee (\varphi \wedge \alpha \overline{;_\Sigma} \varphi \mid \alpha > \psi),\ \sigma)$$

$$\delta(\varphi \mid\mid \alpha \gg \psi,\ \sigma) = \delta(\psi \wedge (\varphi \vee \alpha \;;_\Pi \varphi \mid\mid \alpha \gg \psi),\ \sigma)$$

$$\delta(\varphi \mid\mid \alpha > \psi,\ \sigma) = \delta(\psi \wedge (\varphi \vee \alpha \overline{;_\Pi} \varphi \mid\mid \alpha > \psi),\ \sigma)$$

Die RLTL-Formel $\varnothing$ dient in dem Automaten als Senke für nicht erfüllte Formeln. Daher ist die Ausgabe in diesem Zustand für alle möglichen Eingaben stets $\bot$. Bei der Disjunktion und bei der Konjunktion werden beide Teilformeln separat überprüft. Bei der Negation einer RLTL-Formel werden die Komplemente der Wahrheitswerte als Ausgabe verwendet, die bei der Auswertung der nichtnegierten Formel entstehen würden. Bei den beiden starken Next-Operatoren wird zunächst der reguläre Ausdruck ausgewertet, wobei der dazu notwendige Aufruf der Transitionsfunktion einen regulären Ausdruck berechnet, mit dem die anschließend noch zu erfüllende RLTL-Formel konstruiert wird. Das hierfür notwendige Formel-rewriting übernehmen die für beide Operatoren definierten Funktionen $\varrho^{\;;_\Sigma}$ und $\varrho^{\;;_\Pi}$, welche jeweils drei Fälle unterscheiden.

Falls $b = \top$ gilt, ist der reguläre Ausdruck vollständig erfüllt, sodass beginnend mit dem nächsten Zeichen die Teilformel φ ausgewertet werden muss. Da der Next-Operator als Ganzes aber noch nicht erfüllt ist, ist die Ausgabe $\bot^p$. Tritt der Fall auf, dass der reguläre Ausdruck mit dem zuletzt eingegebenen Zeichen zum Teil erfüllt ist, muss der Restausdruck erfüllt werden, bevor die RLTL-Teilformel φ erfüllt werden muss.

Der dritte Fall beschreibt bei beiden Operatoren den Nachfolgezustand für den Fall, dass der reguläre Ausdruck weder erfüllt ist noch in Zukunft erfüllt werden kann. Gemäß den Semantiken beider Operatoren gibt die Mealy-Maschine im Falle des existentiellen Next-Operators $\bot$ und im Falle des universellen Next-Operators $\top$ aus.

Für die Definition der Transitionsfunktion für die schwachen Next-Operatoren wird die Dualitätsbeziehung zu den beiden starken Next-Operatoren ausgenutzt. Die Nachfolgezustände für die Power-Operatoren und dualen Power-Operatoren ergeben sich aus den jeweiligen Abrollungen und sind daher nicht gesondert aufgeführt.

Mit diesen Transitionsfunktionen kann nun die alternierende Mealy-Maschine generiert

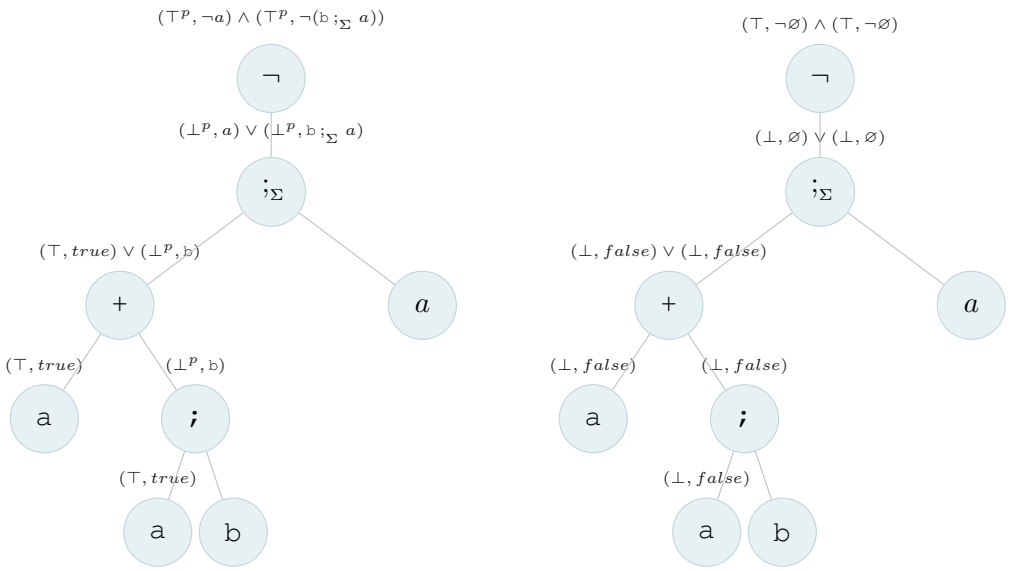

Abbildung 4.1: Am Syntaxbaum veranschaulichte Berechnungen der Transitionsfunktion für die Formel $\varphi = \neg\Big((\mathtt{a} + (\mathtt{a}\,;\mathtt{b}))\,;_\Sigma a\Big)$ bei den Eingaben (φ, a) (links) und (φ, b) (rechts). Die Rückgabewerte stehen jeweils über den Knoten.

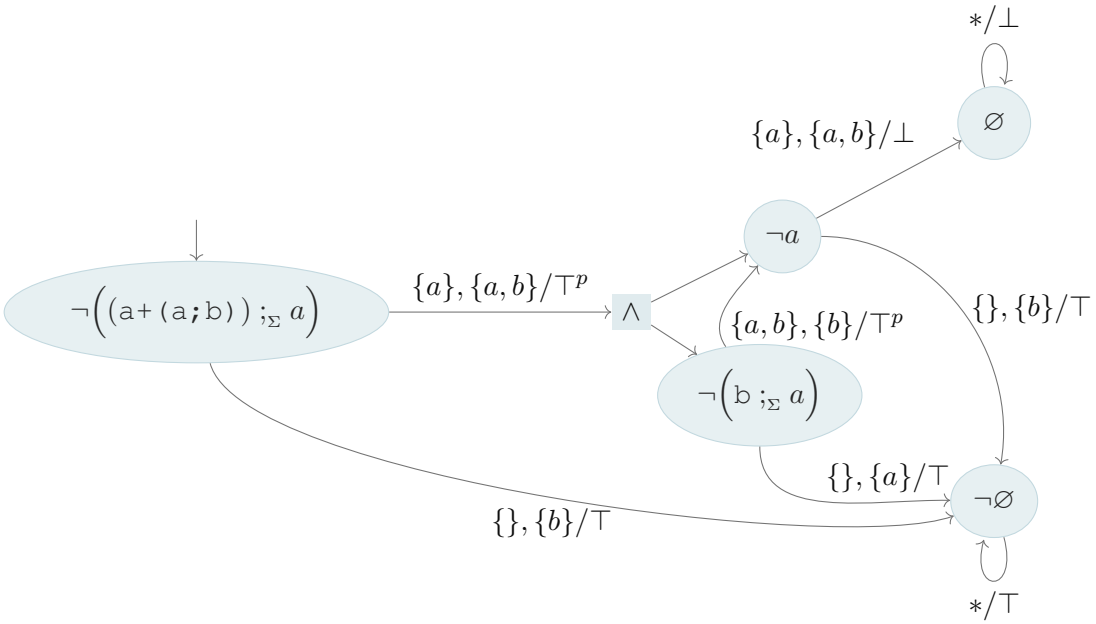

Abbildung 4.2: Alternierende Mealy-Maschine für die RLTL-Formel $\neg\Big((\mathtt{a} + (\mathtt{a}\,;\mathtt{b}))\,;_\Sigma a\Big)$. Das Eingabealphabet ist $2^{\{a,b\}}$.

werden. Aus der RLTL-Formel $\neg(\texttt{(a+(a;b))} \;;_\Sigma a)$, die äquivalent zu $\texttt{(a+(a;b))} \overline{;_\Pi} \neg a$ ist, entsteht so für das Eingabealphabet $\Sigma = 2^{\{a,b\}}$ der in Abbildung 4.2 dargestellte Automat. Vom Startzustand aus wird, wenn kein a in der Eingabe enthalten ist, direkt in die Senke gegangen, die die Eingabe akzeptiert, da der reguläre Ausdruck im Next-Operator nicht erfüllt ist und die gesamte Formel negiert ist. Wenn das erste Zeichen der Eingabe hingegen ein a enthält, gibt es zwei Möglichkeiten, wie es im regulären Ausdruck verarbeitet werden kann. Aufgrund der Semantik des schwachen universellen Next-Operators müssen beide Varianten zur Erfüllung der Formel führen. Daher wird über die Verundung in beide Nachfolgezustände gewechselt. Die Ausgabe »$\top^p$« lässt sich am einfachsten dadurch begründen, dass der reguläre Ausdruck `a;b` teilweise erfüllt ist und ein negierter starker Next-Operator einem schwachen Next-Operator entspricht. Die Berechnung der Wahrheitswerte, die bei den vom Startzustand ausgehenden Transitionen bei den beiden Eingaben $\{a\}$ und $\{b\}$ ausgegeben werden, ist detailliert in Abbildung 4.1 beschrieben, wo für jeden Operator die jeweiligen Zwischenergebnisse der Transitionsfunktion ausgeführt sind.
Wenn beim zweiten Symbol ein a in der Eingabe enthalten wäre, würde die Mealy-Maschine vom Zustand $\neg a$ zu $\varnothing$ wechseln, wodurch die Ausgabe $\bot$ wäre, da für alle Elemente x, die im Wahrheitsverband $\mathbb{B}_4$ enthalten sind, $x \sqcap \bot = \bot$ gilt. Gilt im zweiten Zeitschritt keine Proposition, sind die Formeln in den beiden Zuständen, in denen sich die Mealy-Maschine befindet, erfüllt, sodass von beiden aus in den Zustand $\neg\varnothing$ gewechselt wird, wobei $\top$ ausgegeben wird. Falls hingegen nur b gilt, wechselt der Automat in die Zustände $\neg a$ und $\neg\varnothing$, sodass sich erst nach dem dritten Eingabesymbol entscheidet, was der endgültige Wahrheitswert ist.

4.3 Korrektheit und Komplexität

Nach der formalen Definition der Zustandsübergangsfunktion im vorangegangenen Abschnitt wird nun gezeigt, dass diese die von der vierwertigen Semantik gestellten Anforderungen erfüllt. Dazu wird eine Analyse der Funktionen, die das Formel-rewriting übernehmen, durchgeführt, wobei gezeigt wird, dass die vorgenommenen Ersetzungen korrekt sind. Außerdem wird gezeigt, dass die Wahrheitswerte, welche die Transitionsfunktion für die gesamte Formel ermittelt, den in der Semantik definierten Elementen des Verbands $\mathbb{B}_4$ entsprechen. Des Weiteren wird die Komplexität des Verfahrens analysiert, indem eine Schranke für die Größe des Monitors bewiesen wird.
Bevor die Erläuterungen zu den der Formelersetzung dienenden Funktionen folgen, werden einige Aussagen bewiesen, die ebenfalls für die Automatenkonstruktion benötigt werden. Dazu wird zunächst gezeigt, dass zu Formeln, bei denen der dem Next-Operator vorangestellte reguläre Ausdruck eine Disjunktion ist, äquivalente Formeln existieren, in

denen die Disjunktion im regulären Ausdruck nicht mehr enthalten ist. In Abhängigkeit davon, ob es sich um einen existentiellen oder universellen Next-Operator handelt, enthält die Formel eine Disjunktion oder eine Konjunktion von RLTL-Formeln. Anschließend wird gezeigt, dass es bei einer Konkatenation einer Disjunktion zweier regulärer Ausdrücke mit einem anderen regulären Ausdruck auch möglich ist, die Konkatenation auf die Operanden der Disjunktion anzuwenden, wobei die beiden Ausdrücke zueinander äquivalent sind.

Lemma 4.5 (Äquivalenz der Disjunktion in regulären Ausdrücken und RLTL-Formeln)**.** *Seien $\alpha_1, \alpha_2 \in RE$ sowie $\varphi \in RLTL$. Dann gilt $\alpha_1 + \alpha_2 ;_\Sigma \varphi \equiv (\alpha_1 ;_\Sigma \varphi) \vee (\alpha_2 ;_\Sigma \varphi)$.*

Beweis. Die Aussage wird über die Definition der Semantik gezeigt.

$$
\begin{aligned}
(w,i) \models \alpha_1 + \alpha_2 ;_\Sigma \varphi &\iff \exists k : (w,i,k) \models_{\text{RE}} \alpha_1 + \alpha_2 \wedge (w,k) \models \varphi \\
&\iff \exists k : ((w,i,k) \models_{\text{RE}} \alpha_1 \vee (w,i,k) \models_{\text{RE}} \alpha_2) \wedge (w,k) \models \varphi \\
&\iff \exists k : ((w,i,k) \models_{\text{RE}} \alpha_1 \wedge (w,k) \models \varphi \\
&\qquad \vee (w,i,k) \models_{\text{RE}} \alpha_2 \wedge (w,k) \models \varphi) \\
&\iff \exists k : ((w,i,k) \models_{\text{RE}} \alpha_1 \wedge (w,k) \models \varphi) \\
&\qquad \vee \exists k : ((w,i,k) \models_{\text{RE}} \alpha_2 \wedge (w,k) \models \varphi) \\
&\iff (w,i) \models (\alpha_1 ;_\Sigma \varphi) \vee (\alpha_2 ;_\Sigma \varphi)
\end{aligned}
$$

□

Lemma 4.6 (Äquivalenz von Konjunktion regulärer Disjunktion beim universellen Next–Operator)**.** *Seien $\alpha_1, \alpha_2 \in RE$ sowie $\varphi \in RLTL$. Dann gilt $\alpha_1 + \alpha_2 ;_\Pi \varphi \equiv (\alpha_1 ;_\Pi \varphi) \wedge (\alpha_2 ;_\Pi \varphi)$.*

Beweis. Die Aussage wird über die Definition der Semantik gezeigt.

$$
\begin{aligned}
(w,i) \models \alpha_1 + \alpha_2 ;_\Pi \varphi &\iff \forall k : (w,i,k) \models_{\text{RE}} \alpha_1 + \alpha_2 \rightarrow (w,k) \models \varphi \\
&\iff \forall k : ((w,i,k) \models_{\text{RE}} \alpha_1 \vee (w,i,k) \models_{\text{RE}} \alpha_2) \rightarrow (w,k) \models \varphi \\
&\iff \forall k : ((w,i,k) \not\models_{\text{RE}} \alpha_1 \wedge (w,i,k) \not\models_{\text{RE}} \alpha_2) \vee (w,k) \models \varphi \\
&\iff \forall k : ((w,i,k) \not\models_{\text{RE}} \alpha_1 \vee (w,k) \models \varphi \\
&\qquad \wedge (w,i,k) \not\models_{\text{RE}} \alpha_2 \vee (w,k) \models \varphi) \\
&\iff \forall k : (w,i,k) \not\models_{\text{RE}} \alpha_1 \vee (w,k) \models \varphi \\
&\qquad \wedge \forall k : (w,i,k) \not\models_{\text{RE}} \alpha_2 \vee (w,k) \models \varphi \\
&\iff \forall k : (w,i,k) \models_{\text{RE}} \alpha_1 \rightarrow (w,k) \models \varphi \\
&\qquad \wedge \forall k : (w,i,k) \models_{\text{RE}} \alpha_2 \rightarrow (w,k) \models \varphi \\
&\iff (w,i) \models (\alpha_1 ;_\Pi \varphi) \wedge (\alpha_2 ;_\Pi \varphi)
\end{aligned}
$$

□

Lemma 4.7 (Distributivität von Disjunktion und Konkatenation)**.** *Seien $\alpha_1, \alpha_2, \alpha_3 \in RE$ reguläre Ausdrücke. Dann gilt $(\alpha_1 + \alpha_2)\,;\alpha_3 \equiv (\alpha_1\,;\alpha_3) + (\alpha_2\,;\alpha_3)$.*

Beweis. Die Aussage wird über die Definition der Semantik gezeigt.

$$\begin{aligned}
(w,i,j) \models (\alpha_1 + \alpha_2)\,;\alpha_3 &\iff \exists k : (w,i,k) \models_{\text{RE}} (\alpha_1 + \alpha_2) \wedge (w,k,j) \models_{\text{RE}} \alpha_3 \\
&\iff \exists k : ((w,i,k) \models_{\text{RE}} \alpha_1 \vee (w,i,k) \models_{\text{RE}} \alpha_2) \wedge (w,k,j) \models_{\text{RE}} \alpha_3 \\
&\iff \exists k : (w,i,k) \models_{\text{RE}} \alpha_1 \wedge (w,k,j) \models_{\text{RE}} \alpha_3 \\
&\qquad \vee \exists k : (w,i,k) \models_{\text{RE}} \alpha_2 \wedge (w,k,j) \models_{\text{RE}} \alpha_3 \\
&\iff (\alpha_1\,;\alpha_3) + (\alpha_2\,;\alpha_3)
\end{aligned}$$

□

Obwohl es die RLTL-Syntax nicht zulässt, dass eine Formel lediglich aus einem regulären Ausdruck besteht und die generierten Monitore somit nicht ausschließlich die in Definition 4.3 definierten Transitionsfunktionen beinhalten können, werden im Folgenden aus beweistechnischen Gründen derartige Automaten betrachtet. Hierzu wird die folgende Definition benötigt.

Definition 4.8 (Akzeptanz von regulären Ausdrücken)**.** *Eine Mealy-Maschine akzeptiert einen regulären Ausdruck, wenn sie $\top$ ausgibt.*

Lemma 4.9 (Automatenausgaben für reguläre Ausdrücke)**.** *Die Ausgaben für eine Mealy-Maschine, die genau die Worte akzeptiert, die in der Sprache eines regulären Ausdrucks enthalten sind, sind $\top$, $\bot^p$ und $\bot$. Der vierte denkbare Wert, $\top^p$, ist keine mögliche Ausgabe.*

Beweis. Der Wahrheitswert $\top$ wird benötigt, um zu signalisieren, dass ein Wort akzeptiert wird. Weitere Eingabesymbole können danach ignoriert werden. Da die Erfüllung von Propositionen erfordert, dass das betrachtete Teilwort genau ein Zeichen lang ist und Konkatenationen innerhalb eines regulären Ausdrucks semantisch einem starken Next-Operator entsprechen, kann der Wahrheitswert $\top^p$ nicht auftreten. $\bot$ zeigt an, dass das Wort verworfen wurde. In allen anderen Fällen wurde das Wort nicht akzeptiert, während es dennoch möglich ist, dass eine Verlängerung des Wortes um weitere Zeichen dazu führt, dass dieses akzeptiert wird. Dies entspricht genau der Bedeutung von $\bot^p$. □

Um zu zeigen, dass die Monitorkonstruktion in dem Sinne korrekt ist, dass die Ausgaben der vierwertigen Semantik entsprechen, wird zunächst gezeigt, dass der Automat für die regulären Ausdrücke korrekt ist.

Theorem 4.10 (Automatenkonstruktion für reguläre Ausdrücke)**.** *Sei α ein regulärer Ausdruck. Dann ist $\mathcal{F} = (Q_\alpha, \Sigma, \mathbb{B}_4, \delta, \alpha)$ eine Mealy-Maschine mit $L(\mathcal{F}) = L(\alpha) \circ \Sigma^\omega$, wobei $\circ$ die elementweise Konkatenation bezeichnet und Q_α die Menge der Zustände ist.*

Beweis. Für Propositionen werden direkt die endgültigen Wahrheitswerte verwendet. Die Richtigkeit bei der Disjunktion in regulären Ausdrücken muss nicht näher betrachtet werden, da diese aufgrund der Lemmata 4.5 bis 4.7 als boolesche Verknüpfung von RLTL-Formeln beschrieben werden kann. Bei der Konkatenation werden die Nachfolgezustände von $\alpha ; \beta$ über die in Definition 4.3 definierte rewrite-Funktion $\varrho^{i}_{\mathrm{RE}}$ bestimmt, deren Korrektheit nun gezeigt wird. Hierzu werden drei Fälle unterschieden. Sei b dazu der von der rewrite-Funktion zurückgegebene Wahrheitswert, der sich auf die Auswertung von α bezieht.

i) Sei $b = \top$. Dann wurde der Ausdruck α erfüllt. Dann muss noch β erfüllt werden. Die Ausgabe ist folglich $\bot^p$, da der gesamte Ausdruck weder erfüllt noch verworfen wurde.

ii) Sei $b = \bot^p$. Dann wurde der Ausdruck α zum Teil erfüllt. Es bleibt noch ein Restausdruck α' übrig, der erfüllt werden muss, bevor schließlich β erfüllt werden kann. Folglich ist die Ausgabe $\bot^p$. Der noch zu erfüllende Restausdruck ist somit $\alpha' ; \beta$.

iii) Sei $b = \bot$. Dann wurde α nicht erfüllt, sodass auch die gesamte Konkatenation nicht mehr erfüllt werden kann. Daher ist die Ausgabe $\bot$.

Bei dem Kleene-Operator kommt die rewrite-Funktion $\varrho^{*}_{\mathrm{RE}}$ zum Einsatz. Hierfür sind nochmals drei Fälle bezüglich der Auswertung des regulären Teilausdrucks α zu unterscheiden.

i) Sei $b = \top$. Dann wurde α erfüllt. Gemäß der Semantik des Kleene-Operators gibt es für die noch zu erfüllende Restformel zwei Möglichkeiten. Es gilt direkt β oder es folgt eine Eingabe, die beliebig oft α immer wieder erfüllt, bis schließlich β erfüllt ist. Genau dies drückt der reguläre Ausdruck $\alpha * \beta$ aus, weswegen dies der Nachfolgezustand ist. Die Ausgabe ist $\bot^p$, da die gesamte Formel noch nicht erfüllt ist.

ii) Sei $b = \bot^p$. Dann wurde α zum Teil erfüllt, sodass noch weitere Zeichen den Rest dieses Ausdrucks erfüllen müssen, bis anschließend die Überlegungen aus dem ersten Fall gelten. Ein regulärer Ausdruck, der genau diese Anforderungen erfüllt, ist $\alpha' ; (\alpha * \beta)$, wobei α' der entsprechende Restausdruck ist. Die Ausgabe ist $\bot^p$, da der Ausdruck nicht komplett erfüllt ist.

iii) Sei $b = \bot$. Der Fall ist analog zum dritten Fall der Konkatenation.

Beim Kleene-Operator kann auch direkt β gelten. Daher definiert die Transitionsfunktion auch einen Zustandsübergang zu den in $\delta(\beta,\ \sigma)$ enthaltenen Zuständen, wobei die Auswahl der Transition durch den Automaten nichtdeterministisch erfolgt.
Hieraus folgt, dass die rewrite-Funktionen und Transitionsfunktionen für die regulären Ausdrücke korrekt sind. □

Um die Schranke für die Größe der Monitore zu beweisen, werden nun zunächst nichtdeterministische endliche Automaten formal definiert.

Definition 4.11 (NFA)**.** *Ein* nichtdeterministischer endlicher Automat (nondeterministic finite automaton, NFA) *ist ein* 5*-Tupel* $\mathcal{N} = (Q, \Sigma, \delta, q_0, F)$*. Dabei ist*

- Q *die endliche Zustandsmenge,*
- Σ *das Eingabealphabet,*
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q)$ *die Zustandsübergangsfunktion,*
- $q_0 \in Q$ *der Startzustand und*
- $F \subseteq Q$ *die Menge der akzeptierenden Zustände.*

Mit $\mathcal{P}$ wird dabei wie üblich die Potenzmenge bezeichnet. In dem Alphabet Σ ist ε, das für das leere Wort steht, nicht enthalten. Mit einer ε-Transition kann der Automat den Zustand wechseln ohne ein Zeichen zu lesen.

Die regulären Ausdrücke, die durch die rewrite-Funktionen generiert werden, können deutlich länger sein, als die ursprünglichen regulären Ausdrücke. Dies wird nun an einem Beispiel verdeutlicht, in dem der von der Transitionsfunktion für den Ausdruck `(((a;b)*c)*d)*e` und dem Eingabesymbol a generierte reguläre Ausdruck für den Nachfolgezustand betrachtet wird. Es gilt nach Definition

$$\delta(\texttt{(((a;b)*c)*d)*e}, a) = (\bot^p, \texttt{b;(a;b)*c;((a;b)*c)*d;(((a;b)*c)*d)*e})$$

Obwohl die Länge des Ausdrucks deutlich anwächst, da für jeden vorhandenen Kleene-Stern eine neue Konkatenation erzeugt wird, besitzt der Automat, in den der reguläre Ausdruck übersetzt werden kann, bezüglich der Anzahl der Operatoren linear viele Zustände. Diese Aussage wird gleich formal bewiesen. Informell liegt der Sachverhalt darin begründet, dass die regulären Ausdrücke so miteinander konkateniert werden, dass der vordere Ausdruck der Konkatenation ein Teilausdruck des hinteren Teils ist. Dies ermöglicht es, dass Teile der Automatenkonstruktion erneut verwendet werden können.

Lemma 4.12 (Größe des Automaten)**.** *Sei $\mathcal{F}$ der Automat, der einen regulären Ausdruck α akzeptiert. $\mathcal{F}$ hat $O(n)$ Zustände, wobei n die Anzahl der Operatoren in α ist.*

Beweis. Zu einem regulären Ausdruck kann bekanntlich über die Thompsonsche Konstruktion [Thompson, 1968] ein NFA erzeugt werden, der bezüglich der durch die Anzahl der Operatoren definierte Länge des regulären Ausdrucks linear viele Zustände hat. Nun wird gezeigt, dass die hier beschriebene Konstruktion ebenso viele Zustände benötigt. Dazu werden die einzelnen Operatoren separat betrachtet.

Für Propositionen werden zwei Zustände und eine Transition benötigt, um das Eingabezeichen zu überprüfen. Bei der Disjunktion wird nichtdeterministisch einer der beiden Startzustände der Automaten für die Teilausdrücke gewählt. Die Konkatenation lässt sich repräsentieren, indem für die Teile α und β jeweils Automaten konstruiert werden, die über ε-Transitionen, die von den akzeptierenden Zuständen von α zum Startzustand von β führen, miteinander verbunden werden. Folglich müssen nur die über die rewrite-Funktion vom Kleene-Operator erzeugten Formeln, die deutlich länger werden können, als die ursprünglichen regulären Ausdrücke, genauer betrachtet werden. Die Abbildung 4.3 zeigt schematisch die Thompsonsche Konstruktion eines NFAs für einen regulären Ausdruck, der zwei Teilausdrücke mit dem binären Kleene-Operator verknüpft. Hierzu werden die Automaten für die regulären Ausdrücke α und β, die selbst beliebig komplex sein können, mit ε-Transitionen miteinander verbunden. Im oberen Teil ist die Standardkonstruktion dargestellt. Der Automat für $\alpha * \beta$ im unteren Teil akzeptiert offenbar trotz der Änderungen bei einigen ε-Transitionen die gleiche Sprache wie der Automat für die gleiche Formel im oberen Teil der Abbildung. Nachdem α' durchlaufen wurde, geht der Automat in den Startzustand des Teilautomaten für $\alpha * \beta$. Dies entspricht genau der Konkatenation. Neben den beiden zur Übersichtlichkeit hinzugefügten Zuständen, die nur Nachfolgezustände haben, die über ε-Transitionen erreicht werden und daher bei einer Minimierung komplett wegfallen, sind alle Zustände für $\alpha'; \alpha * \beta$ somit bereits in dem Automaten für den Ausdruck $\alpha * \beta$ enthalten. Dieser NFA hat bekanntlich linear viele Zustände, wenn die Automaten für α und β ebenfalls aus linear vielen Zuständen bestehen, was auch der Fall ist, wenn sie nach dem gleichen Prinzip generiert werden. □

Theorem 4.13 (Monitor für FRLTL$_4$)**.** *Sei $\mathcal{M} = (Q_\varphi, \Sigma, \mathbb{B}_4, \delta, \varphi)$ mit δ aus den Definitionen 4.3 und 4.4 eine als Monitor für die RLTL-Formel φ dienende Mealy-Maschine. Dann sind ihre Ausgaben gemäß der vierwertigen Semantik auf endlichen Worten korrekt.*

Beweis. Nachdem ein Eingabesymbol eine Proposition außerhalb eines regulären Ausdrucks erfüllt, sind die restlichen Zeichen für die Auswertung nicht relevant. Daher werden vom Monitor korrekterweise die endgültigen Wahrheitswerte ausgegeben. Konjunktion und Disjunktion werden als universelle und nichtdeterministische Transitionen realisiert, sodass die Richtigkeit bei diesen Operatoren aus der Korrektheit des Monitors für die Teilausdrücke folgt.
Bei den Next-Operatoren wird zunächst der reguläre Ausdruck ausgewertet. Die Korrektheit der nichtdeterministischen Automaten, welche von der Konstruktion verwendet werden, wurde in Lemma 4.10 gezeigt. Bei den beiden starken Next-Operatoren wird, während der reguläre Ausdruck ausgewertet wird, »$\bot^p$« ausgegeben, bei den schwachen analog dazu »$\top^p$«. Ist der reguläre Ausdruck nach einem Eingabesymbol erfüllt, sodass der Rückgabewert der Transitionsfunktion des regulären Ausdrucks $(true, \top)$ ist, ist der von

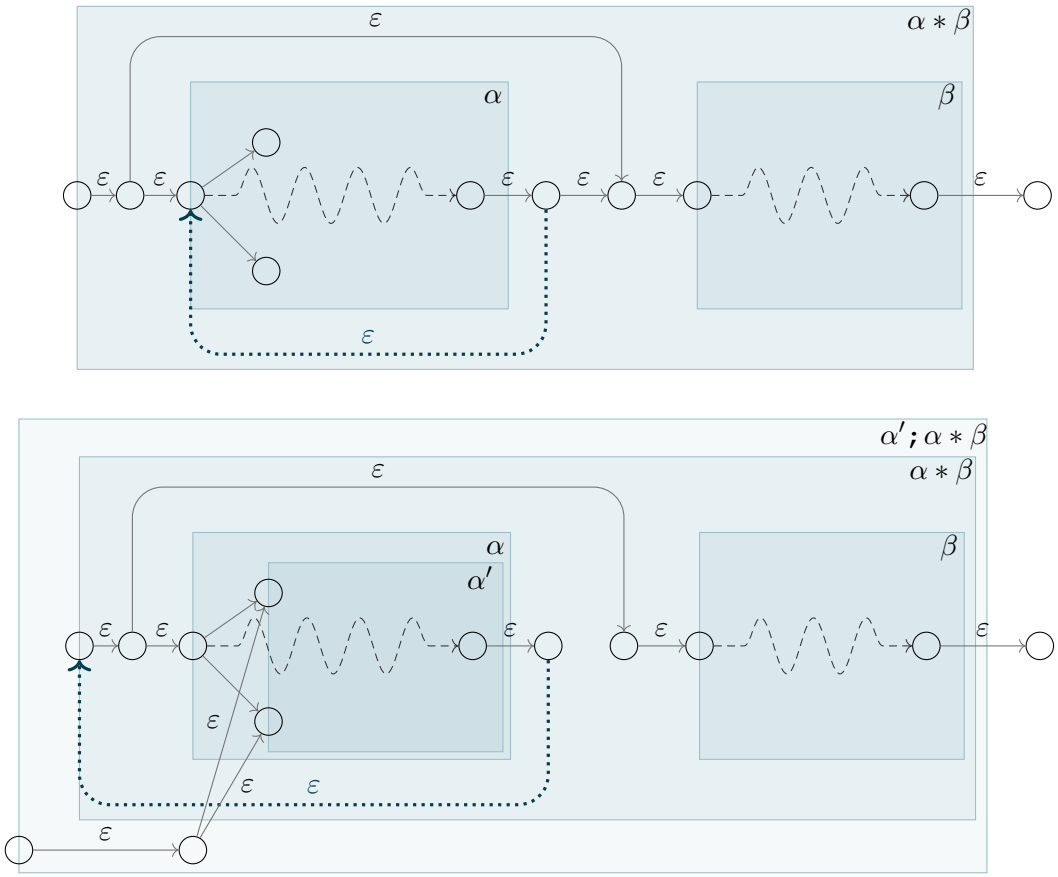

Abbildung 4.3: Graphische Darstellung der Konstruktion von nichtdeterministischen endlichen Automaten mit ε-Transitionen für den Kleene-Operator. Die Zustände, die auf den Rändern der Boxen abgebildet sind, sind die Startzustände der jeweiligen Teilautomaten. Oben und unten akzeptieren die Automaten mit $\alpha * \beta$ die gleiche Sprache. Das Fehlen der ε-Transition zwischen den Automaten für α und β in der unteren Abbildung wird dadurch kompensiert, dass die gepunktet dargestellte Kante zu einem Zustand führt, von dem aus der Startzustand des Automaten für β erreichbar ist und alle Zustandsübergänge dorthin ε-Transitionen sind.

der Transitionsfunktion für die starken Next-Operatoren berechnete Nachfolgezustand korrekterweise φ, wobei »$\bot^p$« ausgegeben wird, was ebenfalls korrekt ist, da die gesamte Formel noch nicht erfüllt sein kann, eine Erfüllung von φ aber gleichzeitig noch möglich ist. Bei dem starken existentiellen Next-Operator reicht gemäß der Semantik eine Aufteilung des Wortes, sodass der vordere Teil den regulären Ausdruck und der hintere Teil die RLTL-Teilformel erfüllt, sodass der Monitor den Übergang von der Auswertung des regulären Ausdrucks zur Auswertung der RLTL-Teilformel nichtdeterministisch wählen kann. Genau dieses Verhalten wird durch die Veroderung der von der rewrite-Funktion berechneten Rückgabewerte repräsentiert.
Bei dem starken universellen Next-Operator werden die Rückgabewerte miteinander verundet. Zusammen mit der Tatsache, dass im Falle der Nichterfüllung des regulären Ausdrucks »$\top$« ausgegeben wird, was das Ergebnis der Konjunktion nicht verändert, entspricht auch dies genau der Semantik, da nach allen Positionen, an denen der reguläre Ausdruck erfüllt ist, auch die RLTL-Teilformel des Next-Operators gelten muss.
Für die Power-Operatoren lassen sich über die Abrollungen äquivalente Formeln konstruieren, die aus Konjunktionen, Disjunktionen und einem der vier Next-Operatoren bestehen, wobei der Power-Operator wiederum im RLTL-Teil des Next-Operators enthalten ist. Somit folgt die Korrektheit der Monitorgenerierung für diese Operatoren aus der Korrektheit der Konstruktionen für die Next-Operatoren, Disjunktionen und Konjunktionen. Somit sind die rewrite-Funktionen und die Transitionsfunktionen aus Definition 4.4 korrekt.
Es bleibt zu zeigen, dass der Monitor $\mathcal{M}$ die Semantik von Worten für die RLTL-Formel φ berechnet. Dies folgt dadurch, dass der Startzustand gerade die zu analysierende Formel repräsentiert und die Menge Q_φ als Zustandsmenge des Monitors sämtliche auftretende Formeln enthält. Dabei ist Q_φ der Fixpunkt der Gleichung

$$\mathcal{Q}_{i+1} = \bigcup_{\sigma \in \Sigma, \widehat{q} \in \mathcal{Q}_i} \tilde{\delta}(\widehat{q}, \sigma)$$

mit $\tilde{\delta}(\widehat{q}, \sigma) = q \Leftrightarrow \delta(\widehat{q}, \sigma) = (q, b)$, wobei initial $\mathcal{Q}_0 = \{\varphi\}$ gilt.
Aus diesen Punkten folgt, dass die Ausgaben des Monitors bezüglich der vierwertigen Semantik korrekt sind. □

Theorem 4.14 (Größe des Monitors für RLTL-Formeln)**.** *Die Größe der alternierenden Mealy-Maschine ist linear bezüglich der Anzahl der Operatoren in der RLTL-Formel.*

Beweis. In Lemma 4.12 wurde bereits gezeigt, dass die Automaten für reguläre Ausdrücke linear viele Zustände benötigen. Für die über einen der vier Next-Operatoren realisierte Konkatenation eines regulären Ausdrucks α mit einer RLTL-Formel φ, wird über Transitionen vom Automaten für α zum Startzustand des Automaten für den Teilausdruck φ

gegangen, sodass außer für die Zustände des Automaten für φ keine weiteren Zustände eingefügt werden müssen. Bei der Disjunktion und der Konjunktion kommen keine weiteren Zustände hinzu, da lediglich Transitionen in die Startzustände der Teilautomaten hinzugefügt werden. Da die Power-Operatoren über die Abrollungen in Automaten überführt werden können, kommt pro Operator nur ein weiterer Zustand hinzu. Für die Propositionen in der RLTL-Formel wird jeweils ein Zustand ergänzt. Mit den Senken $\varnothing$ und $\neg\varnothing$ werden ebenfalls nur konstant viele Zustände hinzugefügt. Für die Teile der RLTL-Formel, die keine regulären Ausdrücke sind, werden somit pro Operator konstant und insgesamt nur linear viele Zustände benötigt. Folglich ist die Größe des Automaten auch insgesamt linear in der Anzahl der Operatoren in der RLTL-Formel. □

Mit dem in diesem Abschnitt vorgestellten Verfahren ist es möglich, zu einer RLTL-Formel eine alternierende Mealy-Maschine zu konstruieren, welche die Formel für ein endliches Wort Schritt für Schritt auswertet und dabei die von der vierwertigen Semantik festgelegten Wahrheitswerte ausgibt. Dabei besteht der Monitor bezüglich der über die Anzahl der Operatoren bestimmte Länge der Formel aus linear vielen Zuständen. Gleiches gilt auch für das Monitorgenerierungsverfahren für LTL-Formeln bei der FLTL_4-Semantik. Somit ist die Generierung von Automaten für RLTL-Formeln trotz der Tatsache, dass sich in dieser Logik mehr Eigenschaften als mit LTL spezifizieren lassen, nicht komplexer.

5 Implementierung in RltlConv

In diesem Kapitel wird auf die Implementierung des Verfahrens zur Monitorgenerierung in der Automatenbibliothek *RltlConv*, die in der Programmiersprache *Scala*[1] geschrieben ist, eingegangen. Außerdem wird die Umsetzung hinsichtlich ihrer Effizienz evaluiert. Dazu wird analysiert, inwiefern sich beim Vergleich von LTL- und äquivalenten RLTL-Formeln Unterschiede in der Laufzeit bemerkbar machen. Zudem werden das Laufzeitverhalten bei unterschiedlich langen Formeln, einige Modifikationen der Konstruktion sowie deren Auswirkungen auf die Performanz betrachtet.
Dieser Abschnitt ist dazu wie folgt gegliedert: Zunächst wird die Automatenbibliothek *RltlConv* in Abschnitt 5.1 vorgestellt, wo auch die bereits vorhandenen Funktionalitäten beschrieben werden. Anschließend wird in Abschnitt 5.2 auf die Funktionen, die im Rahmen dieser Arbeit zur Bibliothek hinzugefügt wurden, eingegangen. Danach werden in Abschnitt 5.3 anhand von Beispielen einige RLTL-Formeln und die dazugehörigen Automaten betrachtet, welche mit den neu hinzugefügten Funktionen generiert wurden. Den Abschluss bilden die Performanzanalysen in Abschnitt 5.4.

5.1 RltlConv

Die in der Programmiersprache Scala geschriebene Automatenbibliothek *RltlConv*, welche am Institut für Softwaretechnik und Programmiersprachen an der Universität zu Lübeck entwickelt wird, stellt bereits etliche Funktionen zur Generierung von verschiedenen Automaten zu LTL- und RLTL-Formeln zur Verfügung. Sie unterstützt die Konstruktion von alternierenden Mealy-Maschinen gemäß der FLTL_4-Semantik, sowie die Übersetzung von LTL-Formeln in nichtdeterministische Büchi-Automaten und die Generierung von alternierenden Paritätsautomaten zu RLTL-Formeln. Zudem können Paritätsautomaten in Büchi-Automaten transformiert werden. Eine weitere bereits implementierte Funktion ist die Erzeugung von RLTL-Formeln aus ω-regulären Ausdrücken und LTL-Formeln. Außerdem existieren bereits Funktionalitäten, die verschiedene Automatentypen minimieren oder nichtdeterministische beziehungsweise alternierende Automaten in deterministische Automaten umwandeln. Des Weiteren können die von RltlConv erzeugten Automaten visualisiert werden, indem eine Ausgabe im DOT-Format erzeugt und mittels *GraphViz*[2] die graphische Ausgabe generiert wird.

[1] http://www.scala-lang.org
[2] http://graphviz.org

5.2 Implementierung der Automatengenerierung

In dieser Arbeit wurde RltlConv dahingehend erweitert, dass alternierende Mealy-Maschinen auch aus RLTL-Formeln erzeugt werden können. Diese werden nach der in Kapitel 4 vorgestellten Konstruktion generiert und ermöglichen es somit, dass sie als Monitore verwendet werden können, die Systeme überwachen und dabei die von der vierwertigen Semantik berechneten Wahrheitswerte ausgeben.
Die maßgeblichen Änderungen wurden in den Klassen `RegexExpression` und `Rltl-Expression` vorgenommen, in denen die Funktion *delta* hinzugefügt wurde, welche die in den Definitionen 4.3 und 4.4 aufgeführten Transitionsfunktionen für die einzelnen Operatoren implementiert. Die Transitionen werden dadurch realisiert, dass die Funktion *delta* den Typ `PosBool` zurückgibt, der eine positive boolesche Kombination repräsentiert. Das UML-Diagramm in Abbildung 5.1 gibt einen Überblick über die vorhandenen Klassen und Funktionen. Zur besseren Übersicht sind nicht alle Klassen, die Operatoren repräsentieren, enthalten. Die eigentliche Konstruktion der alternierenden Mealy-Maschine zu einer RLTL-Formel φ wird über die in der Klasse `RltlExpression` implementierten Funktion *toAMealy* initiiert, welche eine Tiefensuche durchführt, die in dieser Form auch bereits in der Klasse `LtlExpression` existiert, in der über die Funktion *toAMealy-Handler* zu LTL-Formeln alternierende Mealy-Maschinen erzeugt werden. Die Tiefensuche berechnet rekursiv, ausgehend von der Formel φ, die den Startzustand repräsentiert, die Nachfolgezustände für die möglichen Eingabesymbole. Dabei wird der Automat solange sukzessiv um neue Zustände erweitert, bis die Transitionsfunktionen nur noch bereits vorhandene Zustände als Nachfolgezustände zurückgeben. Der folgende Pseudocode veranschaulicht das Prinzip.

```
dfs(φ, ∅)
fun dfs(q, allstates)
  if (q ∩ allstates ≠ ∅) return
  allstates ← allstates ∪ q
  newstates ← ⋃_{σ∈Σ} transform(delta(q, σ))
  foreach q' in newstates
    dfs(q', allstates)
```

In der Klasse `RltlExpression` sind zwei Funktionen enthalten, die eine alternierende Mealy-Maschine zurückgeben. Diese unterscheiden sich jedoch in der Anzahl ihrer Parameter. Eine Funktion bekommt nur Optionen übergeben, die unter anderem Informationen über das Alphabet enthalten. Diese Funktion ist in der Klasse enthalten, damit eine Generierung der Monitore für RLTL-Formel analog zur Monitorkonstruktion bei LTL-Formeln möglich ist, wo die entsprechende Funktion die gleichen Parameter erwartet. Darüber hinaus existiert in `RltlExpression` eine weitere Funktion mit zusätzlichen Parametern, welche von der zuvor beschriebenen Funktion mit Standardwerten aufge-

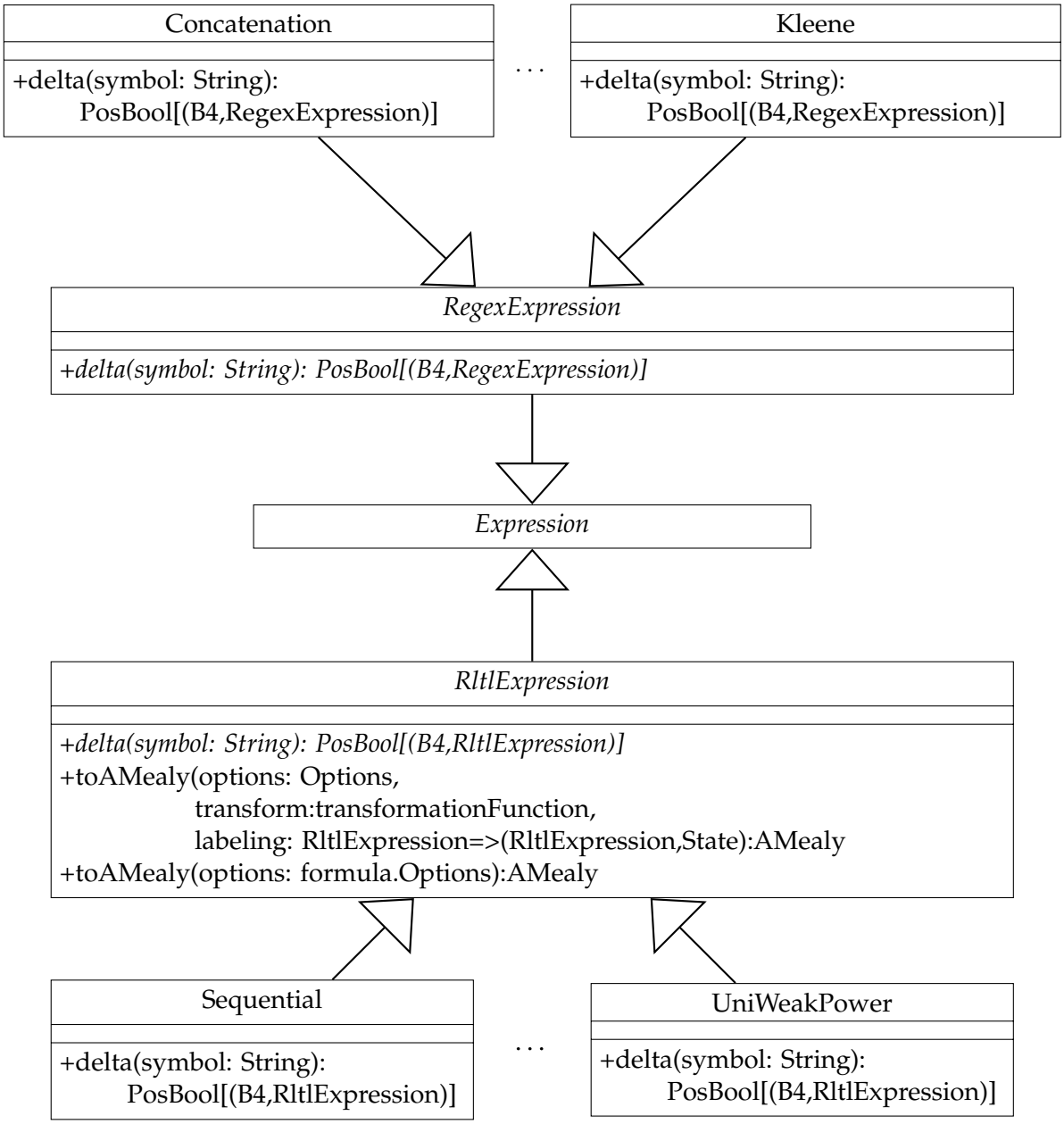

Abbildung 5.1: Vereinfachtes UML-Diagramm zur Veranschaulichung der implementierten Funktionen. Funktionen, die für diese Arbeit irrelevant sind, sind nicht im Diagramm enthalten.

rufen wird. Die auf diesem Weg übergebenen Funktionen ermöglichen es, während der Konstruktion des Monitors Einfluss auf das Verfahren zu nehmen. Mittels *transform* wird eine Funktion übergeben, die die positiven booleschen Kombinationen, die bei der Konstruktion des Automaten als Rückgabewerte der Transitionsfunktionen auftreten, modifiziert. Dieser Schritt erfolgt, wie dem Pseudocode entnommen werden kann, für alle Aufrufe der Transitionsfunktion während der Tiefensuche. Sinnvollerweise können an dieser Stelle Funktionen verwendet werden, welche die positiven booleschen Kombinationen vereinfachen. So kann zum Beispiel $\varnothing \wedge \psi$ direkt zu $\varnothing$ vereinfacht werden, sodass die Nachfolgezustände von ψ nicht berechnet werden müssen. Außerdem ist es an dieser Stelle möglich, aus der positiven booleschen Kombination eine aus Disjunktionen und Konstruktionen bestehende RLTL-Formel zu erzeugen, um komplett auf nichtdeterministische und universelle Transitionen zu verzichten, sodass der erzeugte Monitor in diesem Fall deterministisch ist. Dies ist über die Berechnung der minimalen Modelle einer positiven booleschen Kombination realisiert, damit für keine Formel ein neuer Zustand ergänzt wird, obwohl bereits in dem Automaten eine äquivalente Formel existiert. Außerdem können anstelle der Senken, welche für die beiden Formeln $\varnothing$ und $\neg\varnothing$ stehen, extra Zustände hinzugefügt werden, die angeben, dass der Automat die Eingabe akzeptiert oder verwirft. Die Vorteile dieses Verfahrens sind in Abschnitt 5.3 näher beschrieben, wo die Funktionsweise der Transformationen auch an Beispielen verdeutlicht wird.

Die Implementierung ist an dieser Stelle erweiterbar. Sie ist so gestaltet, dass auf einfache Weise zusätzliche Transformationsfunktionen hinzugefügt werden können. Diese können zudem miteinander kombiniert werden, sodass die einzelnen Funktionen in einer beliebigen Reihenfolge hintereinander ausgeführt werden können.

In der Datei `RltlToAMealyUtils` stellt das Objekt `Simplify` zusätzlich zur trivialen Funktion, welche die Eingabe unverändert zurückgibt, insgesamt drei verschiedene Funktionen zur Verfügung, welche die von den Transitionsfunktionen zurückgegebenen booleschen Kombinationen von Nachfolgezuständen vereinfachen oder geeignet transformieren. Deren genaue Funktionsweise ist in Tabelle 5.1 beschrieben.

Des Weiteren gibt es in der Funktion *toAMealy* den Parameter *labeling*, mit dem die Bezeichnung der Zustände des Monitors in der graphischen Ausgabe festgelegt werden kann. In der bisherigen Version der Automatenbibliothek wurden die Zustände von q_0 bis $q_{|Q|-1}$ durchnummeriert; über die neu hinzugefügte Funktion ist es zum Beispiel auch möglich, die RLTL-Formeln selbst als Beschriftung für die Zustände zu verwenden, sodass die graphische Ausgabe leichter verständlich ist. Die einzelnen Operatoren der RLTL-Formel sind dabei in unterschiedlichen Farben dargestellt, damit die Struktur der Formel problemlos erkannt werden kann. Die hierzu benötigten Funktionen befinden sich in der Klasse `RltlPrinter`. Im Abschnitt 5.3 werden Beispiele für die verschiedenen implementierten Funktionen aufgeführt.

Name	Beschreibung
`none`	Führt keinerlei Optimierungen durch
`simple`	Vereinfacht die boolesche Kombination rekursiv durch Anwendung von Regeln wie zum Beispiel $\varnothing \wedge \varphi = \varnothing$, $\varphi \vee \varphi = \varphi$ und $\varphi \vee \varnothing = \varphi$.
`onestate`	Bildet boolesche Kombinationen von Formeln auf Formen ab. Aus $(\top^p, a) \vee (\bot^p, b \overline{;_\Pi} \neg\varnothing)$ wird zum Beispiel $(\top^p, a \vee b \overline{;_\Pi} \neg\varnothing)$.
`boolsink`	Fügt zu jeder Transition, die in die Zustände $\varnothing$ oder $\neg\varnothing$ führen würde, eine eigene boolesche Senke ein.

Tabelle 5.1: Zur Verfügung stehende Funktionen zur Transformation der positiven booleschen Kombinationen bei der Automatenkonstruktion. Eine Kombination dieser Funktionen ist ebenfalls möglich.

Die alternierenden Mealy-Maschinen können auch in einen deterministischen Monitor transformiert werden. Hierzu wird einfach die Funktion, die schon für die Monitorgenerierung bei LTL-Formeln zum Einsatz kommt, verwendet. Bei dieser Umwandlung können nach der in Abschnitt 4.1 erwähnten Schranke doppelt exponentiell viele Zustände entstehen.
Das Programm lässt sich über die Kommandozeile bedienen. Zur Generierung einer alternierenden Mealy-Maschine muss die Automatenbibliothek mit den Kommandozeilenoptionen „`RLTL=<Formel>" --props --amealy` ausgerufen werden, wobei durch diese Argumente die Potenzmenge der Propositionen als Eingabealphabet verwendet wird und der Automatentyp eine alternierende Mealy-Maschine ist. Bereits vorhandene Funktionalitäten, wie die Umwandlung in deterministische Monitore, die Minimierung sowie die graphische Ausgabe als `pdf`-Datei, können weiterhin über `--mealy`, `--min` und `--pdf` aufgerufen werden. Damit auch Formeln der um die beiden Next-Operatoren erweiterten Logik eingegeben werden können, wurde RltlConv diesbezüglich von den ursprünglichen Autoren erweitert. Mittels welcher Zeichen und Symbole die Formeln so eingegeben werden können, dass die von der Automatenbibliothek verarbeitet werden können, ist in Tabelle 5.2 aufgeführt.

5.3 Beispiele

In diesem Abschnitt werden als Beispiel einige RLTL-Formeln betrachtet, zu denen dann nach den in der Automatenbibliothek implementierten Verfahren verschiedene Monitore generiert werden. Sei als erstes $\varphi_1 = a \mid \texttt{true;true} \gg (a \mid \texttt{true} > b)$. Diese Formel verlangt, dass solange an jeder zweiten Position a gilt, bis die Teilformel erfüllt ist, die aus dem schwachen Power-Operator besteht, was der Fall ist, wenn nur noch a gilt, bis irgendwann b gilt, oder a unendlich lange gilt. Gemäß der Semantik des schwachen

		Operator	Symbol	Eingabe
RE	Reguläre Ausdrücke	Disjunktion	$+$	$\vert$
		Konkatenation	$;$	$\sqcup$
		Kleene-Stern	$*$	$*$
RLTL-Operatoren	Konstanten	leere Sprache	$\varnothing$	% , EMPTY
	Boolesche Operatoren	Disjunktion	$\vee$	|| , OR
		Konjunktion	$\wedge$	&& , AND
		Negation	$\neg$	! , NOT
	Next-Operatoren	stark, existentiell	$;_{\Sigma}$	;
		stark, universell	$;_{\Pi}$	;;
		schwach, existentiell	$\overline{;_{\Sigma}}$	:
		schwach, universell	$\overline{;_{\Pi}}$	::
	Power-Operatoren	stark, existentiell	$\vert \gg$	/ >>
		stark, universell	$\Vert \gg$	// >>
		schwach, existentiell	$\vert >$	/ >
		schwach, universell	$\Vert >$	// >

Tabelle 5.2: Eingabesymbole für die RLTL-Formeln für die Verwendung in RltlConv. Falls eine alternative Eingabe möglich ist, ist diese, getrennt durch ein Komma, ebenfalls aufgelistet. Propositionen werden durch kleine Buchstaben eingegeben.

Power-Operators ist die Formel bei einem Wort, das auf a endet, erfüllt, wobei eine spätere Verletzung, die dadurch auftreten kann, dass keine Proposition gilt, nicht ausgeschlossen ist. Daher ist der vom Monitor in solch einem Fall ausgegebene Wahrheitswert $\top^p$. Solange a nur an jeder zweiten Position gilt, ist die Ausgabe $\bot^p$, da die Formel nicht erfüllt sein kann, ohne den Versuch-Teil des Power-Operators zu erfüllen, ein geeignet verlängertes Wort dies jedoch noch erreichen kann. Die Ausgaben des Monitors für eine Beispieleingabe sind in der folgenden schrittweisen Auswertung aufgeführt.

$$\begin{array}{cccccccc} \{a\} & \{\} & \{a\} & \{a\} & \{a\} & \{\} & \{a\} & \{b\} \\ \top^p & \bot^p & \top^p & \top^p & \top^p & \bot^p & \top^p & \top \end{array}$$

An der sechsten Position ist die Ausgabe $\bot^p$. Hier wurde die vom schwachen Power-Operator gestellte Anforderung verletzt, allerdings gilt noch an jeder zweiten Position a, sodass eine spätere Erfüllung der Formel noch immer möglich ist.
In Abbildung 5.2 wurden für die Formel φ_1 auf zwei verschiedene Arten eine alternierende Mealy-Maschine erzeugt. Bei dem oben abgebildeten Automaten wurden keinerlei Optimierungen vorgenommen. Es ist zu erkennen, dass etliche Transitionen vorhanden sind, die nicht benötigt werden. So sind zum Beispiel vom Startzustand q_3 aus, wenn die Eingabe $\{\}$ ist, die Zustände q_1, q_2 und q_4 erreichbar. Bei genauem Hinsehen ist jedoch zu erkennen, dass dabei die verwerfende Senke q_1 über jede nichtdeterministische und

jeweils über mindestens eine universelle Transition erreicht wird, sodass die Ausgabe auf jeden Fall $\bot$ ist und anstelle der positiven booleschen Kombination eine einzelne Transition zu q_1 mit der Ausgabe $\bot$ verwendet werden kann. Derartige Vereinfachungen führt die Funktion `simple` durch. Wird diese Funktion während der Automatengenerierung aufgerufen, entsteht der Automat, der im unteren Teil von Abbildung 5.2 dargestellt ist. Außerdem sind dort die durch das Formel-rewriting erzeugten RLTL-Formeln enthalten, die in den jeweiligen Zuständen erfüllt sein müssen. Für die Formel φ_1 können fast alle nichtdeterministischen und universellen Transitionen eliminiert werden. Zum Vergleich ist im oberen Teil der Automat abgebildet, der entsteht, wenn während der Tiefensuche keine Optimierung stattfindet. Neben der Tatsache, dass Automaten mit vereinfachten Transitionen leichter vom Menschen zu analysieren sind, erfordern sie auch weniger Speicherplatz und ermöglichen schnellere Berechnungen, da die Übergänge von weniger Zuständen berücksichtigt werden müssen, sodass sich auch bei der maschinellen Verarbeitung Vorteile ergeben. Mit den bereits vorhandenen Funktionalitäten der Automatenbibliothek RltlConv kann eine alternierende Mealy-Maschine in einen deterministischen Monitor transformiert werden. Der minimierte Automat ist in Abbildung 5.3 dargestellt.

Der in Abbildung 5.4 dargestellte Monitor ist das Resultat der Generierung einer Mealy-Maschine zu der Formel $\varphi_2 = a \mid (\texttt{a;a;a})\star(\texttt{a;a}) \gg b$, wobei der Automat in einen deterministischen Monitor transformiert wurde. Der Verzögerungsteil des Power-Operators enthält einen Kleene-Operator, sodass die Länge der Verzögerung nicht fest vorgegeben ist. Die Verzögerung kann für eine natürliche Zahl $n \in \mathbb{N}_0$ aus einer Folge von $2 + 3 \cdot n$ Schritten bestehen, an denen jeweils a gelten muss. Nachdem beginnend mit dem initialen Zustand viermal die Proposition a gelesen wurde und sich der Automat im Zustand q_1 befindet, existiert nach jeder beliebigen Anzahl an weiteren as die Möglichkeit die Eingabe zu akzeptieren, wenn im Wort ein b folgt. Da der größte gemeinsame Teiler der Längen aller vom regulären Ausdruck ermöglichten Verzögerungen offenbar 1 ist, existiert eine Schranke, ab der sich sämtliche Verzögerungen als positive Kombination von Verzögerungen der Längen $2, 5, 8, 11, \cdots$ beschreiben lassen [Brauer und Shockley, 1962]. Die größte Zahl, die sich nicht als Summe von positiven Vielfachen von teilerfremden Zahlen ausdrücken lässt, heißt auch *Frobenius-Zahl*. Das Problem die entsprechende Zahl zu finden wird entsprechend als *Frobenius-Problem* bezeichnet und ist für eine beliebig große Eingabemenge NP-hart [Alfonsín, 1996]. Durch die mehrfache Abrollung des Power-Operators ist es möglich, die denkbaren Verzögerungen, die vom regulären Ausdruck vorgegeben werden, beliebig oft zu durchlaufen. Im Beispiel ist die Frobenius-Zahl 3, da $4 = 2 \cdot 2$ und $5 = 1 \cdot (2 + 3)$ gilt und sich alle größeren Zahlen durch die Addition von einem Vielfachen von 2 erreichen lassen. Dass, bevor die Proposition b gilt, viermal a gilt, wird dabei durch die zweimalige Abrollung des Power-Operators erreicht, wobei in beiden Fällen direkt der hintere Teil der Kleene-Operators erfüllt wird. Eine Verzögerung der Länge 5 entsteht

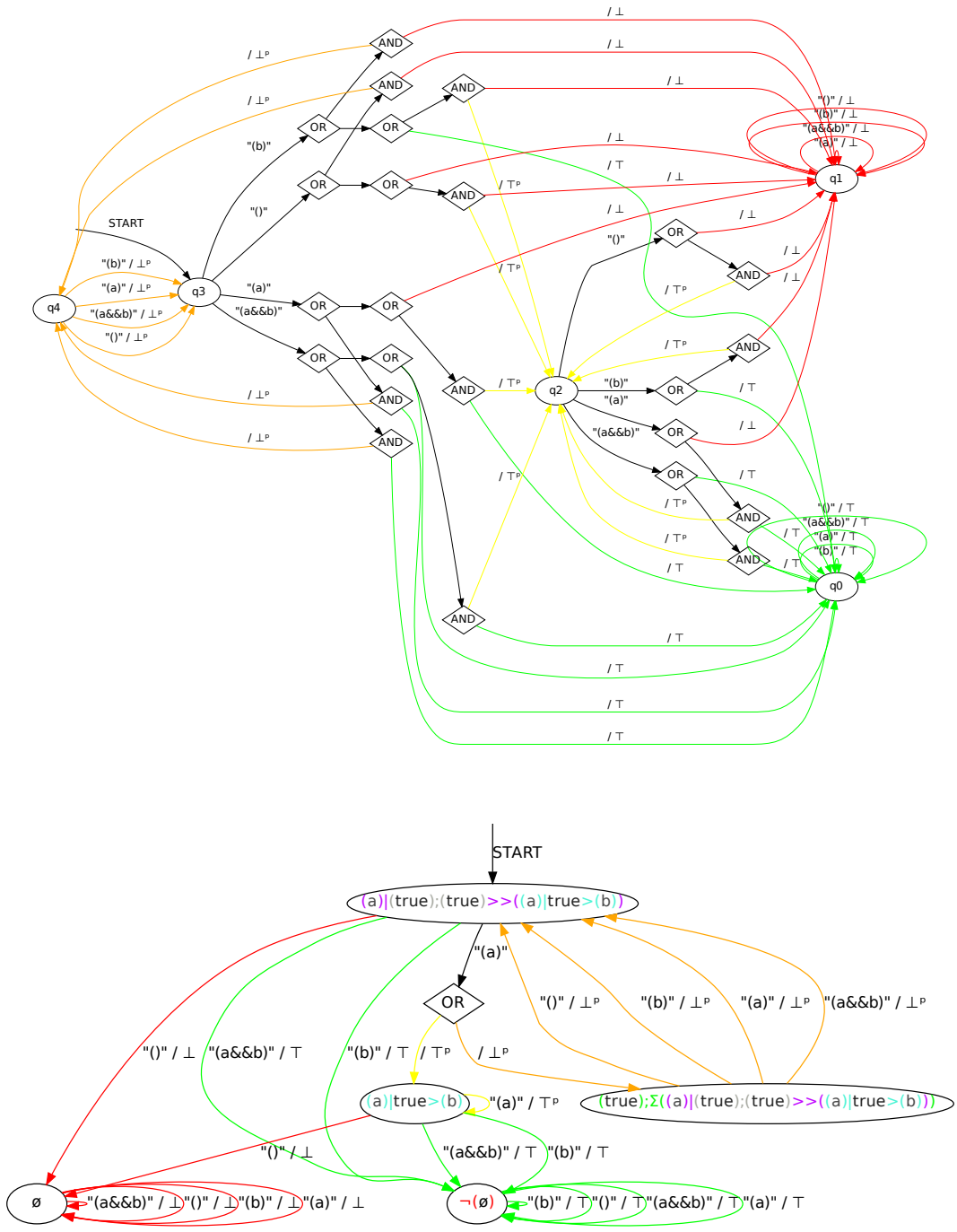

Abbildung 5.2: Von RltlConv generierte und mit GraphViz gezeichnete Monitore für die Formel $\varphi_1 = a \mid \texttt{true;true} \gg (a \mid \texttt{true} > b)$. Der obere Automat wurde nach dem bisherigen Verfahren erzeugt, der untere mit vereinfachten Zustandsübergängen sowie Formeln als Zustandsbeschriftung.

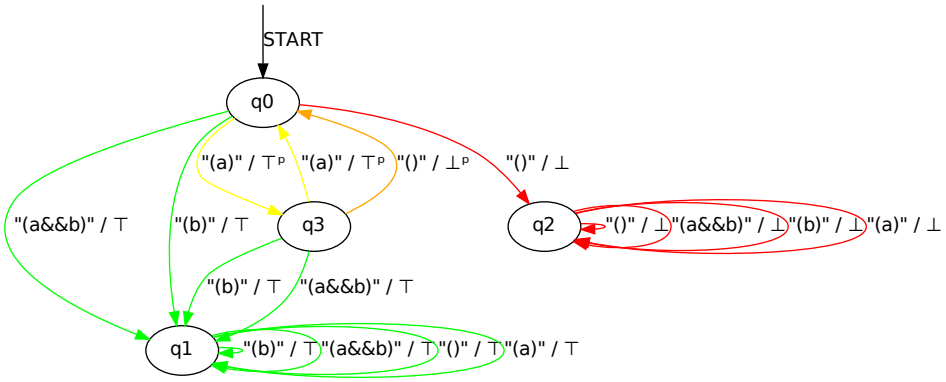

Abbildung 5.3: Der Monitor für die RLTL-Formel $\varphi_1 = a \mid \texttt{true;true} \gg (a \mid \texttt{true} > b)$ nach Umwandlung in einen deterministischen Automaten und anschließender Minimierung.

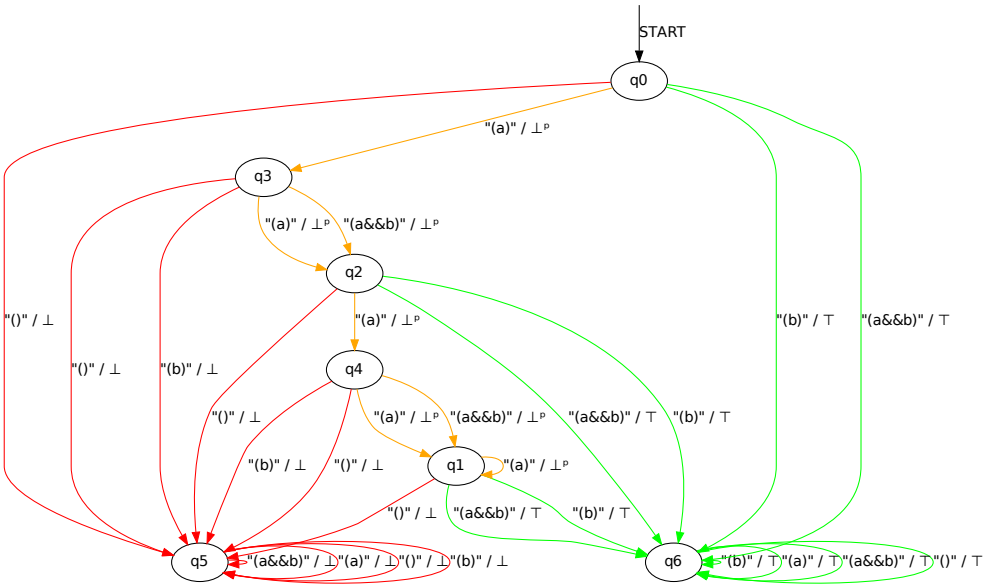

Abbildung 5.4: Der Monitor für die RLTL-Formel $\varphi_2 = a \mid \texttt{(a;a;a)*(a;a)} \gg b$ nach der Umwandlung in einen deterministischen Automaten. Die Optimierungsfunktion `boolsink` berechnet einen äquivalenten Monitor.

dadurch, dass der Power-Operator einmal abgerollt wird und der vordere und der hintere Teilausdruck des Kleene-Operators genau einmal erfüllt werden.

Das Beispiel ist noch aus einem weiteren Grund interessant. Der Monitor in Abbildung 5.4, der offenbar minimal ist, lässt sich auf verschiedene Arten erzeugen. Die alternierende Mealy-Maschine wird dabei zunächst in einen deterministischen Automaten umgewandelt, der anschließend minimiert wird. Die Umwandlung kann effizienter durchgeführt werden, wenn anstelle der Zustände im Automaten, die als akzeptierende und verwerfende Senke dienen und für die RLTL-Formeln $\varnothing$ und $\neg\varnothing$ stehen, gesonderte boolesche Senken verwendet werden. Die beiden unterschiedlichen Automaten sind in Abbildung 5.5 dargestellt. Der oben abgebildete Automat hat nach der Umwandlung in einen deterministischen Automaten vor der Minimierung 14 Zustände, der untere nur sieben. Da nur nichtdeterministische Transitionen enthalten sind, kann dort direkt gefolgert werden, dass sich der Automat nach einer Transition in einer akzeptierenden Senke befindet, wenn ein $true$ in der positiven booleschen Kombination der Nachfolgezustände enthalten ist. Folglich sind die anderen Zustände in der positiven booleschen Kombination nicht relevant, insbesondere müssen für diese keine weiteren Nachfolgezustände berechnet werden.

Die nicht optimierte Methode, welche nach der bekannten Potenzmengenkonstruktion [Scott und Rabin, 1959] einen nichtdeterministischen Automaten in einen deterministischen Automaten umwandelt, ermöglicht es, dass akzeptierende Zustände wieder verlassen werden können. Bei den hier betrachteten Automaten ist dies aufgrund der endgültigen Wahrheitswerte jedoch nicht möglich. Somit würde das Verfahren unnötigerweise die Nachfolgezustände von zu vielen Powerzuständen berechnen. So gelangt der Automat in Abbildung 5.5 zum Beispiel vom Startzustand über die Eingabe $\{a\}$ in den Powerzustand $\{A, D\}$, von wo aus er nach der gleichen Eingabe zu $\{B, q_0\}$ gelangt. Dann kann nach der Eingabe $\{a, b\}$ zu $\{C, q_6\}$ gewechselt werden. Da q_6 alle Fortsetzungen des Wortes akzeptiert und ausschließlich nichtdeterministische Transitionen vorhanden sind, ist es nicht nötig, weitere Nachfolgezustände zu berechnen, was der naive Algorithmus aber dennoch macht. So wird ausgehend von $\{C, q_6\}$ bei der Eingabe $\{a\}$ mit $\{A, C, q_6\}$ ein weiterer Zustand hinzugefügt, von dem aus wiederum weitere neue Zustände entstehen.

Analog dazu können auch unnötigerweise neue Zustände zum Automaten hinzugefügt werden, wenn Zustände betrachtet werden, in denen sich der Automat in einer Konjunktion von Zuständen befindet, von denen einer der Zustand ist, in dem sämtliche Eingaben über die Ausgabe des Wahrheitswertes $\bot$ verworfen werden. Ebenso sind beliebige Kombinationen von beiden Varianten denkbar, sodass sich insbesondere bei großen Automaten enorme Unterschiede in der Laufzeit ergeben können. Dies gilt sowohl für die Transformation in einen deterministischen als auch für die Minimierung eines derartigen Automaten.

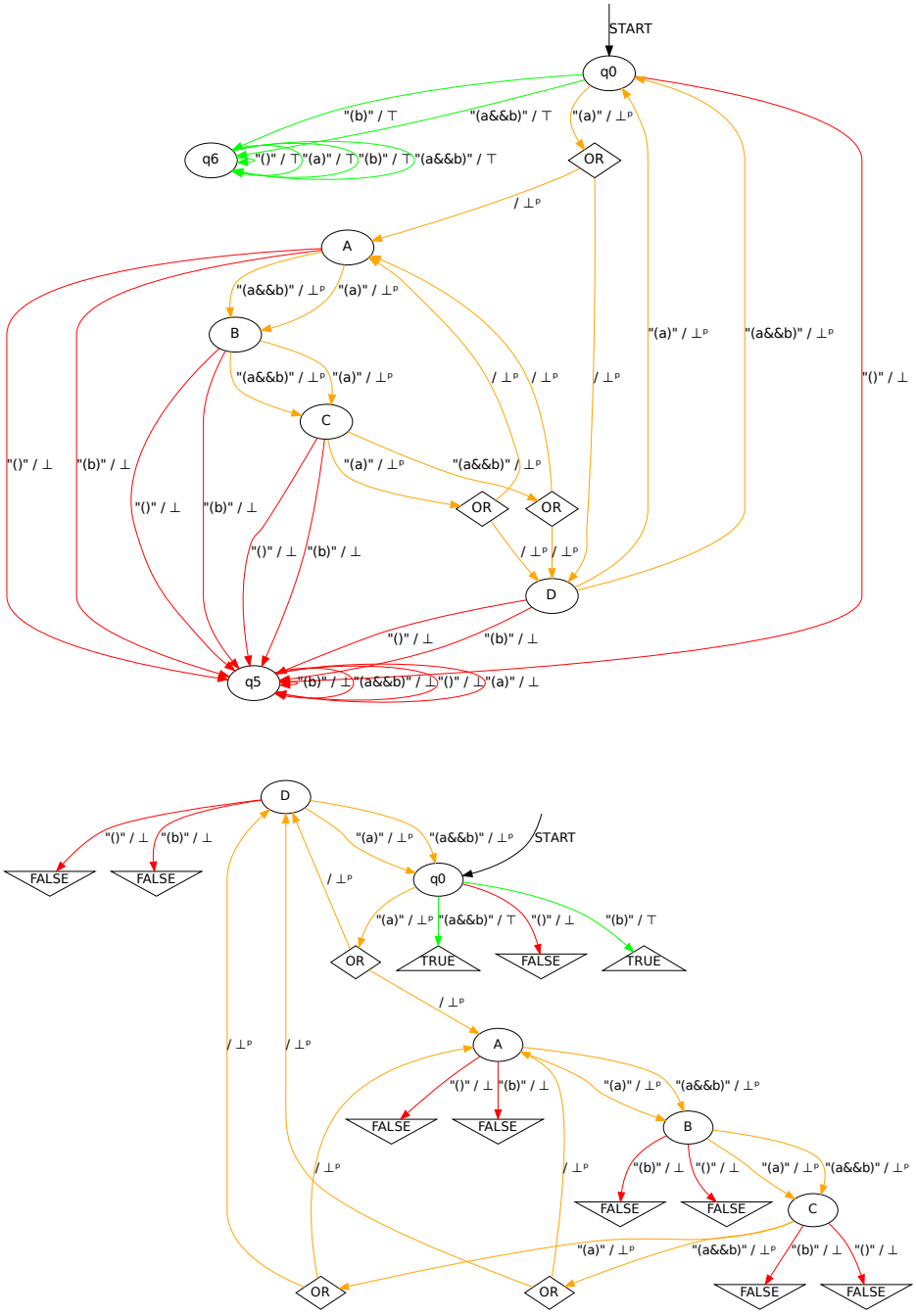

Abbildung 5.5: Nach verschiedenen Verfahren generierte alternierende Mealy-Maschinen für die RLTL-Formel $\varphi_2 = a \mid (\mathtt{a;a;a})^\star(\mathtt{a;a}) \gg b$.

5.4 Analyse

In diesem Abschnitt wird die Implementierung des Verfahrens zur Monitorgenerierung in RltlConv empirisch evaluiert. Dazu wird sowohl die Konstruktion von alternierenden als auch die Generierung von deterministischen Automaten analysiert. Außerdem werden experimentell die Laufzeiten der verschiedenen implementierten Verfahren bei der Erstellung von Monitoren zu RLTL-Formeln verglichen. Sämtliche in diesem Abschnitt genannten Laufzeiten für die Generierung von Monitoren beziehen sich auf einen Rechner, der mit einem Intel Core i7-4770 Prozessor mit 3,4 GHz ausgestattet ist.

Der Vergleich der Laufzeiten bei der Übersetzung von LTL- und RLTL-Formeln in alternierende Mealy-Maschinen ist in Abbildung 5.6 graphisch dargestellt. Es wurden zufällige LTL-Formeln generiert, bei denen die Anzahl der Operatoren zwischen 1 und 150 lag. Für jeweils 25 verschiedene Formeln wurde die Zeit gemessen, die zur Konstruktion einer alternierenden Mealy-Maschine benötigt wurde. Dies geschah sowohl für die LTL- als auch für die dazu äquivalenten RLTL-Formeln. Die Messwerte geben den jeweiligen Mittelwert an. Es ist zu erkennen, dass die Automatengenerierung bei RLTL-Formeln geringfügig länger dauert. Dies liegt vor allem daran, dass im Falle der Next-Operatoren, die in RLTL mit $\texttt{true};\varphi$ ausgedrückt werden, in RLTL für jedes Eingabesymbol überprüft werden muss, ob es den regulären Ausdruck erfüllt, da die Syntax dieser Logik beliebige Verzögerungen erlaubt. Bei LTL-Formeln kann der nächste Zustand ohne eine solche Überprüfung hinzugefügt werden, da die Transitionen auf jeden Fall für sämtliche Zeichen mit der gleichen Ausgabe zu diesem Zustand führen. Dennoch ist der Unterschied zwischen den beiden Logiken relativ gering, die Laufzeiten liegen jeweils im Millisekundenbereich. Die Streuung dieser Werte ist für Formeln mit 100 Operatoren in Abbildung 5.6 dargestellt. Die alternierenden Automaten können für derartige Formeln im Mittel in 57 beziehungsweise 87 Millisekunden konstruiert werden. Diese Mittelwerte sind in der Graphik farbig auf der y-Achse markiert. Größere Abweichungen von diesen Werten nach oben kommen verhältnismäßig selten vor und betreffen sowohl LTL- als auch RLTL-Formeln. Die Generierung von alternierenden Mealy-Maschinen nach dem in Kapitel 4 beschriebenen Verfahren hat also ein vorhersagbares Laufzeitverhalten.

Die verschiedenen Optimierungsfunktionen führen hingegen zu einem sehr unterschiedlichen Laufzeitverhalten bei der Generierung von deterministischen Monitoren. Die Dauer der Konstruktion einer alternierenden Mealy-Maschine mit anschließender Umwandlung in einen deterministischen Automaten ist für die verschiedenen Funktionen, welche die positiven booleschen Kombinationen während der Monitorgenerierung modifizieren, in Abbildung 5.7 dargestellt. Außerdem werden die Optimierungsfunktionen mit der Transformationsfunktion `none` verglichen, damit ersichtlich ist, in wie vielen Fällen eine Verkürzung der Laufzeit erreicht werden kann. Dabei fällt auf, dass die Streuung der

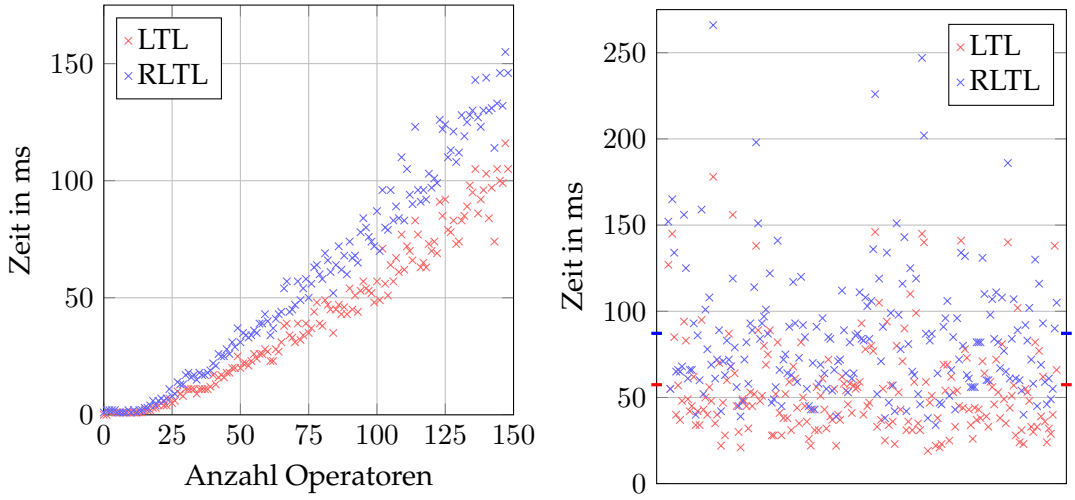

Abbildung 5.6: Laufzeiten für die Generierung einer alternierenden Mealy-Maschine für LTL- und RLTL-Formeln mit bis zu 150 Operatoren und maximal sieben verschiedenen Propositionen. Jeder Messwert steht für den Mittelwert aus 25 durchgeführten Experimenten (links). Laufzeiten für die Generierung von alternierenden Mealy-Maschinen für LTL-Formeln und äquivalente RLTL-Formeln mit 100 Operatoren und sieben Propositionen (rechts).

Messwerte bei der Funktion `simple` besonders hoch ist. Wenn nur diese Funktion verwendet wird, die eine alternierende Mealy-Maschine generiert, die möglichst wenig universelle und nichtdeterministische Transitionen hat, kann es passieren, dass die Laufzeit bei der Umwandlung in einen deterministischen Automaten deutlich über dem Wert liegt, der erreicht werden kann, wenn keine Optimierung der alternierenden Mealy-Maschine erfolgt. So lassen sich Formeln konstruieren, bei denen eine Vereinfachung des alternierenden Monitors die Laufzeit bei der Determinisierung von einigen Millisekunden auf eine halbe Stunde anhebt.[3] Solch ein Fall tritt auf, wenn eine positive boolesche Kombination von Zuständen, die durch überflüssige Transitionen in der alternierenden Mealy-Maschine unnötig groß ist, im weiteren Verlauf auch von der optimierten Variante erreicht werden kann, diese auf dem Weg dorthin jedoch noch Zustände generiert, die von der nichtoptimierten Variante nicht erreicht werden. Wie in Graphik 5.7 unten rechts zu sehen ist, kommt es in anderen Fällen hingegen mit `simple` zu einer leichten Zeitersparnis, die darauf zurückzuführen ist, dass für entsprechende Formeln insgesamt weniger Zustände betrachtet werden.

Mit `onestate` sind die Unterschiede zur Variante `none` bei allen getesteten Formeln gering, liegen aber größtenteils leicht darüber. Das arithmetische Mittel der Messwerte, das

[3] Eine derartige Formel ist $(a \mid \texttt{((a;a);(a;a))*a} \gg b \vee a \mid \texttt{a*b} \gg b) \mid \texttt{(a;(b;a))*(a;b)} \gg b$

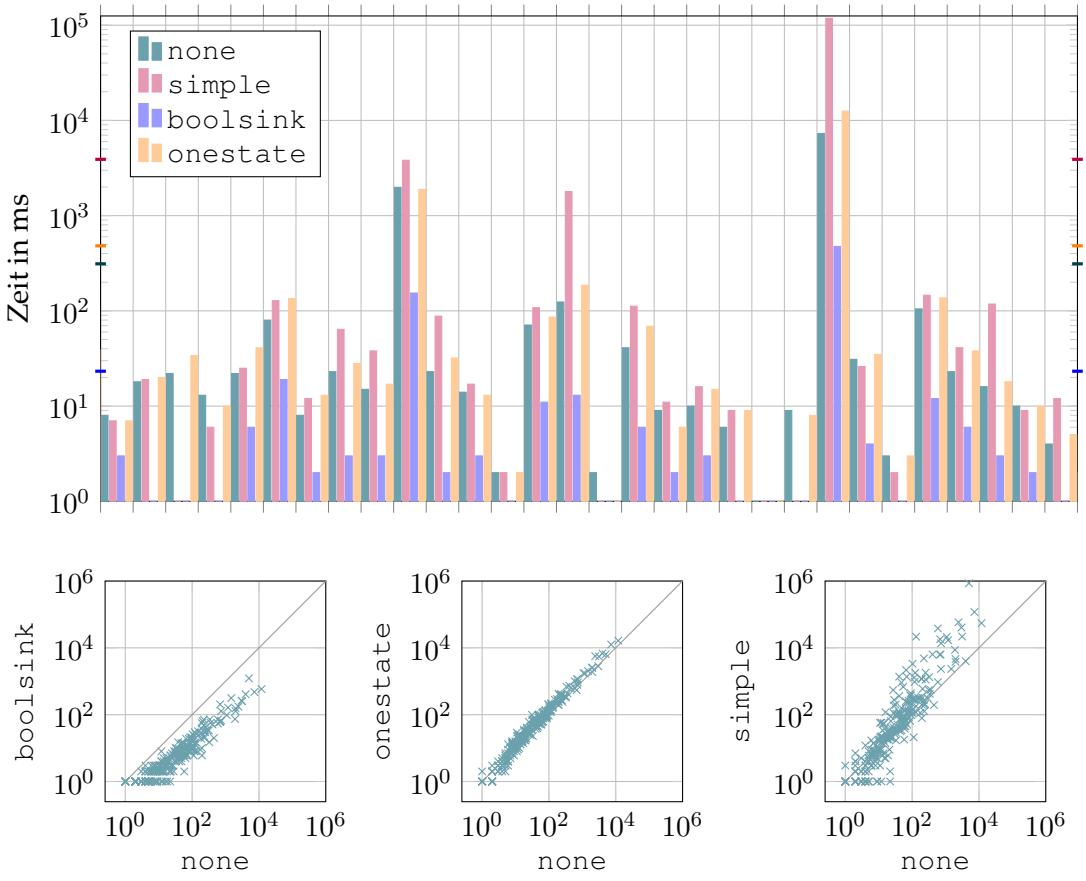

Abbildung 5.7: Laufzeiten der Generierung deterministischer Monitore unter Verwendung verschiedener Transformationsstrategien für zufällige RLTL-Formeln mit sieben Operatoren und fünf Propositionen im Vergleich zur unoptimierten Variante. Oben sind einige der 250 Tests im direkten Vergleich gegenübergestellt, wobei die Auswahl zufällig vorgenommen wurde. Die Graphiken unten zeigen jeweils das Verhältnis der Laufzeiten einer der Transformationsfunktionen zur unoptimierten Variante für sämtliche Tests. Alle Zeitangaben sind in Millisekunden aufgeführt.

in der Abbildung auf der y-Achse markiert ist, zeigt, dass die Laufzeit auch im Durchschnitt ansteigt.

Außerdem ist der Graphik zu entnehmen, dass mit der Funktion `boolsink` gegenüber der Funktion `none`, die keine Veränderungen vornimmt, bei sämtlichen getesteten Formeln teils erhebliche Verbesserungen bezüglich der Laufzeit erreicht werden können. Die zusätzliche Zeit, die zur Berechnung dieser Funktion aufgebracht werden muss, kann dadurch, dass bei der Konstruktion des deterministischen Monitors akzeptierende und verwerfende Zustände direkt erkannt werden können, wodurch keine unnötigen Zustände hinzugefügt werden, mehr als kompensiert werden.

Der Erfolg der einzelnen Transformationen ist somit maßgeblich von der Struktur der RLTL-Formel abhängig. Eine Analyse, welche Formeln für welche Optimierungsvarianten besonders geeignet sind, ist im Rahmen dieser Arbeit nicht durchführbar.

Abbildung 5.8 zeigt die Ergebnisse der Analyse, die die Laufzeiten der Generierung eines deterministischen Monitors für LTL-Formeln und für RLTL-Formeln miteinander vergleicht. Dazu wurden zufällige LTL-Formeln generiert, die aus bis zu acht Operatoren und vier verschiedenen Propositionen bestehen. Zu diesen wurde jeweils eine äquivalente RLTL-Formel generiert, woraufhin zu beiden Formeln über den Zwischenschritt der alternierenden Mealy-Maschinen deterministische Monitore erzeugt wurden. Für LTL-Formeln wurde das bestehende Verfahren aus der Automatenbibliothek RltlConv verwendet. Für RLTL-Formeln kam das in dieser Arbeit beschriebene Prozedere zum Einsatz, wobei bei der Konstruktion der alternierenden Automaten zwei unterschiedliche Varianten verwendet wurden. Einmal wurden keine Transformationen vorgenommen, das andere mal kamen die Optimierungsfunktionen `simple` und `boolsink` zum Einsatz. Obwohl es sich bei RLTL um die ausdrucksmächtigere Logik handelt, lassen sich die Monitore mit dem optimierten Verfahren im Allgemeinen in der kürzeren Zeit generieren. Der Mittelwert liegt bei LTL-Formeln bei 78 Millisekunden und bei dem optimierten Verfahren für RLTL-Formeln bei 6,5 Millisekunden. Es ist allerdings anzunehmen, dass die Laufzeiten des Algorithmus für die Monitorkonstruktion bei LTL-Formeln durch geeignete Optimierungen auf ein Niveau gesenkt werden können, das unter dem liegt, das für RLTL-Formeln erreicht wurde. Des Weiteren fällt auf, dass die Varianz der Laufzeiten der Monitorgenerierung für unterschiedliche Eingabeformeln relativ hoch ist. Das liegt daran, dass je nach Verschachtelungstiefe und Auswahl der Operatoren unterschiedlich komplexe Automaten entstehen. Eine Formel, in der mehrere Until-Operatoren ineinander verschachtelt sind, erfordert zum Beispiel einen höheren Berechnungsaufwand als eine Formel, in der ebenso viele Next-Operatoren aufeinander folgen. Detailliertere Analysen, welche die Verknüpfung von Transformationsfunktionen mit einbeziehen, zeigen, dass im Allgemeinen die Hintereinanderausführung der Funktionen `boolsink` und `simple` zu den geringsten Laufzeiten führt. Der Nutzen von `boolsink` wurde be-

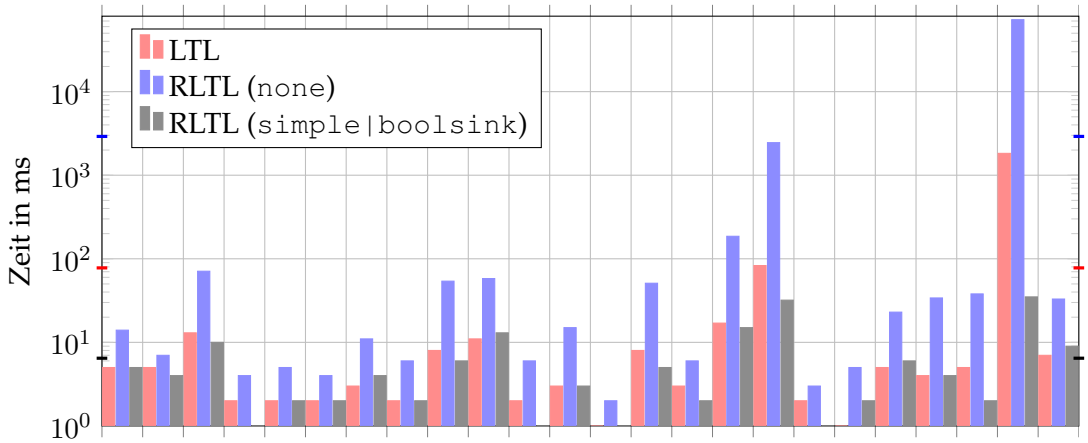

Abbildung 5.8: Vergleich der Laufzeiten bei der Konstruktion deterministischer Mealy-Maschinen für LTL- und äquivalente RLTL-Formeln. Die optimierte Monitorgenerierung für RLTL-Formeln ist im Durchschnitt schneller als das Standardverfahren bei LTL-Formeln.

reits ausführlich beschrieben. Wenn diese Funktion auf Transitionen einer alternierenden Mealy-Maschine, die mittels `simple` vereinfacht wurde, angewendet wird, können die Vorteile beider Funktionen kombiniert werden, da der Algorithmus, der den Automaten in einen deterministischen Monitor transformiert, keine überflüssigen Zustände aufgrund von redundanten Transitionen generiert, und gleichzeitig akzeptierende und verwerfende Senken als solche erkennen kann.
Aus der Analyse folgt, dass alternierende Mealy-Maschinen besonders effizient aus RLTL-Formeln generiert werden können. Die Umwandlung dieser Automaten in deterministische Monitore ist möglich und am effizientesten realisierbar, indem bei der Konstruktion des alternierenden Monitors die Kombination der Transformationsfunktionen `simple` und `boolsink` verwendet wird.

6 Zusammenfassung und Ausblick

In dieser Arbeit wurde mit FRLTL_4 eine vierwertige Semantik für RLTL definiert, sodass es bei der Laufzeitverifikation möglich ist, für ein endliches Wort, das noch um weitere Zeichen verlängert werden kann, zu prüfen, ob es die in der Temporallogik spezifizierten Eigenschaften einhält. Dazu wurde ein Verfahren beschrieben und implementiert, mit dem zu gegebenen RLTL-Formeln alternierende Mealy-Maschinen konstruiert werden können, die ein System als Monitor überwachen und die von der Semantik definierten Wahrheitswerte ausgeben. Im Vergleich zu FLTL_4 lassen sich infolgedessen zusätzliche Korrektheitseigenschaften überprüfen, da RLTL aufgrund der Tatsache, dass sich durch die in der Logik enthaltenen regulären Ausdrücke sämtliche ω-regulären Sprachen ausdrücken lassen, mächtiger als die Lineare Temporale Logik ist, mit der sich lediglich die sternfreien ω-regulären Sprachen beschreiben lassen.

Dazu wurden zunächst die Grundlagen der Runtime-Verification sowie die Definitionen von LTL und RLTL aufgeführt, welche sich auf unendliche Worte beziehen. Nach einem Überblick über weitere temporale Logiken wurden die bereits existierenden verschiedenwertigen Semantiken für LTL auf endlichen Worten beschrieben. Um eine vierwertige Semantik auch für RLTL angeben zu können, musste diese Logik erweitert werden, indem zusätzlich ein schwacher existentieller Next-Operator eingeführt wurde, zu dem der starke universelle Next-Operator dual ist. Zusammen mit dem starken existentiellen sowie dem schwachen universellen Next-Operator, die aus der Definition der RLTL-Semantik für unendliche Worte stammen, gibt es somit insgesamt vier Next-Operatoren, deren Semantik sich auf endlichen Worten unterscheidet. Die Definition der Semantik erfolgte induktiv für die einzelnen Operatoren der RLTL-Formeln, wobei der angegebene Wahrheitswert zum einen beschreibt, ob ein Wort mit den bislang bekannten Zeichen eine Formel erfüllt und zum anderen Auskunft darüber gibt, ob diese Entscheidung endgültig ist oder ob ein anderes Ergebnis möglich ist, wenn das Wort um weitere Zeichen verlängert wird. Im Anschluss an die Definition wurde begründet, warum diese den Anforderungen genügt. Das Verfahren zur Monitorgenerierung konstruiert eine alternierende Mealy-Maschine, deren Zustände die RLTL-Formeln sind, die noch erfüllt werden müssen, wenn sich der Automat in ihnen befindet. Die entsprechenden Formeln werden von der in dieser Arbeit definierten Transitionsfunktion berechnet, die für ein Eingabesymbol in einer bottom-up-Konstruktion durch Auswertung der relevanten Teilformeln auch den von der vierwertigen Semantik festgelegten Wahrheitswert bestimmt,

der vom Monitor ausgegeben wird. Es wurde zudem gezeigt, dass das Verfahren korrekt ist und einen Automaten konstruiert, der bezüglich der Anzahl der in der RLTL-Formel enthaltenen Operatoren linear viele Zustände hat. Des Weiteren wurde das erarbeitete Verfahren in der Automatenbibliothek RltlConv, die am Institut für Softwaretechnik und Programmiersprachen an der Universität zu Lübeck entwickelt wird, implementiert. Der Programmcode zur Berechnung der Zustände des Automaten ist dabei so gestaltet, dass es möglich ist, beliebige Optimierungen bei der positiven booleschen Kombination von Nachfolgezuständen, die von der Transitionsfunktion für einen Zustand berechnet wird, vorzunehmen. Somit können sowohl die Größe des Automaten als auch die Laufzeit, die für die Monitorgenerierung benötigt wird, reduziert werden. So existieren die Funktionen `simple`, welche die positive boolesche Kombination der Nachfolgezustände vereinfacht, `boolsink`, die spezielle Senken für definitiv erfüllte beziehungsweise verletzte Formeln einfügt und `onestate`, mit der direkt deterministische Automaten erzeugt werden. Über die in der Bibliothek bereits vorhandenen Funktionen ist es außerdem möglich, die erzeugten alternierenden Mealy-Maschinen in deterministische Monitore zu transformieren. Für beide Varianten sowie für die Optimierungsfunktionen, zeigt die Arbeit die Funktionalität an Beispielen. Zum Abschluss wurde die Implementierung hinsichtlich ihrer Effizienz evaluiert, indem zu verschiedenen Formeln die Laufzeiten gemessen wurden, die zur Generierung der Automaten benötigt wurden. Während die Generierung alternierender Monitore für RLTL-Formeln im Vergleich zu LTL-Formeln unwesentlich länger dauert, hat sich gezeigt, dass die Laufzeiten bei der Konstruktion deterministischer Monitore besonders stark von der Struktur der RLTL-Formel abhängig sind. Außerdem hat sich gezeigt, dass die Kombination der Optimierungsfunktionen `simple` und `boolsink` zur stärksten Laufzeitreduktion führt.

Da die Transformation alternierender Automaten in deterministische Monitore dazu führen kann, dass doppelt exponentiell viele Zustände entstehen können, sodass die Laufzeit bei großen Automaten so hoch ist, dass die Umwandlung in der Praxis nicht durchführbar ist, können auch direkt alternierende Automaten zur Überwachung des Systems einsetzt werden, wobei verschiedene Strategien bei der Auswertung der Läufe möglich sind [Finkbeiner und Sipma, 2004]. Eine möglichst effiziente Analyse der Worte zur Laufzeit eines Systems mit aus RLTL-Formeln generierten Monitoren kann ebenso Gegenstand für zukünftige Arbeiten sein, wie die Verwendung von mehrwertigen Semantiken für RLTL bei der Laufzeitverifikation von verteilten Systemen, die in [Scheffel und Schmitz, 2014] für LTL beschrieben wurde.

Literaturverzeichnis

[Alfonsín, 1996] Alfonsín, J. L. R. (1996). Complexity of the Frobenius Problem. *Combinatorica*, 16(1):143–147.

[Bauer, 2010] Bauer, A. (2010). Monitorability of omega-regular languages. *CoRR*, abs/1006.3638.

[Bauer et al., 2007] Bauer, A., Leucker, M., und Schallhart, C. (2007). The Good, the Bad, and the Ugly, but How Ugly is Ugly? In *Proceedings of the 7th International Conference on Runtime Verification*, RV'07, pages 126–138, Berlin, Heidelberg. Springer-Verlag.

[Bauer et al., 2010] Bauer, A., Leucker, M., und Schallhart, C. (2010). Comparing LTL Semantics for Runtime Verification. *Journal of Logic and Computation*, 20(3):651–674.

[Bauer et al., 2011] Bauer, A., Leucker, M., und Schallhart, C. (2011). Runtime Verification for LTL and TLTL. *ACM Trans. Softw. Eng. Methodol.*, 20(4):14:1–14:64.

[Brauer und Shockley, 1962] Brauer, A. und Shockley, J. E. (1962). On a problem of Frobenius. *Journal für die reine und angewandte Mathematik*, 211:215–220.

[Chandra und Stockmeyer, 1976] Chandra, A. K. und Stockmeyer, L. J. (1976). Alternation. *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, 0:98–108.

[De Giacomo und Vardi, 2013] De Giacomo, G. und Vardi, M. Y. (2013). Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI'13, pages 854–860. AAAI Press.

[Eisner et al., 2003] Eisner, C., Fisman, D., Havlicek, J., Gordon, M., Mcisaac, A., und Campenhout, D. V. (2003). Formal Syntax and Semantics of PSL.

[Finkbeiner und Sipma, 2004] Finkbeiner, B. und Sipma, H. (2004). Checking Finite Traces Using Alternating Automata: Special Issue on Selected Papers from the First International Workshop on Runtime Verification Held in Paris, July 2001 (RV01). *Formal Methods in System Design*, pages 101–127.

[Fischer und Ladner, 1979] Fischer, M. J. und Ladner, R. E. (1979). Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18(2):194 – 211.

[Gabbay et al., 1980] Gabbay, D., Pnueli, A., Shelah, S., und Stavi, J. (1980). On the Temporal Analysis of Fairness. In *Proceedings of the 7th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '80, pages 163–173, New York, NY, USA. ACM.

[Henriksen und Thiagarajan, 1999] Henriksen, J. G. und Thiagarajan, P. (1999). Dynamic linear time temporal logic. *Annals of Pure and Applied Logic*, 96(1–3):187 – 207.

[Lange, 2007] Lange, M. (2007). Linear Time Logics Around PSL: Complexity, Expressiveness, and a Little Bit of Succinctness. In *CONCUR 2007 - Concurrency Theory, 18th International Conference, CONCUR 2007, Lisbon, Portugal, September 3-8, 2007, Proceedings*, pages 90–104.

[Leucker, 2012] Leucker, M. (2012). Teaching Runtime Verification. In *Proceedings of the Second International Conference on Runtime Verification*, RV'11, pages 34–48, Berlin, Heidelberg. Springer-Verlag.

[Leucker und Sánchez, 2007] Leucker, M. und Sánchez, C. (2007). Regular Linear Temporal Logic. In *Proceedings of the 4th International Conference on Theoretical Aspects of Computing*, ICTAC'07, pages 291–305, Berlin, Heidelberg. Springer-Verlag.

[Lichtenstein et al., 1985] Lichtenstein, O., Pnueli, A., und Zuck, L. (1985). The glory of the past. In Parikh, R., editor, *Logics of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 196–218. Springer Berlin Heidelberg.

[Manna und Pnueli, 1987] Manna, Z. und Pnueli, A. (1987). A Hierarchy of Temporal Properties. Technical report, Stanford, CA, USA.

[Mealy, 1955] Mealy, G. H. (1955). A Method for Synthesizing Sequential Circuits. *Bell System Technical Journal*, 34(5):1045–1079.

[Olender und Osterweil, 1990] Olender, K. M. und Osterweil, L. J. (1990). Cecil: A sequencing constraint language for automatic static analysis generation. *IEEE Trans. Softw. Eng.*, 16(3):268–280.

[Pnueli, 1977] Pnueli, A. (1977). The temporal logic of programs. *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, 0:46–57.

[Sánchez und Leucker, 2010] Sánchez, C. und Leucker, M. (2010). Regular Linear Temporal Logic with Past. In Barthe, G. und Hermenegildo, M., editors, *Proceedings of the 11th International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI'10)*, volume 5944 of *Lecture Notes in Computer Science*, pages 295–311. Springer.

[Sànchez und Samborski-Forlese, 2012] Sànchez, C. und Samborski-Forlese, J. (2012). Efficient Regular Linear Temporal Logic Using Dualization and Stratification. In *19th International Symposium on Temporal Representation and Reasoning, TIME 2012, Leicester, United Kingdom, September 12-14, 2012*, pages 13–20.

[Scheffel und Schmitz, 2014] Scheffel, T. und Schmitz, M. (2014). Three-valued asynchronous distributed runtime verification. In *International Conference on Formal Methods and Models for System Design (MEMOCODE)*, volume 12, EPFL, Lausanne, Switzerland. IEEE, IEEE.

[Scott und Rabin, 1959] Scott, D. und Rabin, M. O. (1959). Finite Automata and Their Decision Problems. *Ibm Journal of Research and Development*, 3:114–125.

[Stockmeyer, 1974] Stockmeyer, L. J. (1974). *The complexity of decision problems in automata theory and logic*. PhD thesis, Massachusetts Institute of Technology.

[Thompson, 1968] Thompson, K. (1968). Programming Techniques: Regular Expression Search Algorithm. *Commun. ACM*, 11(6):419–422.

[Vardi, 2008] Vardi, M. Y. (2008). From church and prior to PSL. In *In O. Grumberg and H. Veith (Eds.), 25 Years of Model Checking*, pages 150–171. Springer.

[Vardi und Wolper, 1986] Vardi, M. Y. und Wolper, P. (1986). An Automata-Theoretic Approach to Automatic Program Verification. In *Proc. 1st Symp. on Logic in Computer Science*, pages 332–344, Cambridge.

[Wolper, 1981] Wolper, P. (1981). Temporal logic can be more expressive. *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, 0:340–348.