UNIVERSITÄT ZU LÜBECK
INSTITUTE OF SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

Logiken für verteilte Laufzeitverifikation

Logics for distributed Runtime-Verification

Masterarbeit

im Rahmen des Studiengangs
Informatik
der Universität zu Lübeck

vorgelegt von
Torben Scheffel

ausgegeben und betreut von
Prof. Dr. Martin Leucker

Lübeck, den 20. Dezember 2013

Kurzfassung

Im Rahmen dieser Arbeit wird eine Logik zum Spezifizieren von Eigenschaften verteilter, asynchroner Systeme entwickelt. Das Ziel, welches dabei mit berücksichtigt wurde, ist der Einsatz dieser Logik für die Laufzeitverifikation von vorher genannten Systemen. Dafür wird die existierende past-time Distributed Temporal Logic (ptDTL), die für diesen Ausgabenbereich entwickelt wurde, als Grundlage genommen und eine neue Logik mit Impartiality, Anticipation und einer dreiwertigen Semantik entwickelt, Eigenschaften, die ptDTL nicht besitzt. Des Weiteren wird ein Automatenmodell für Formeln dieser neuen Logik sowie eine Umwandlung der Formeln in dieses Modell entwickelt.

Abstract

A logic for specifying properties of distributed, asynchronous systems is developed in this thesis. It is intended to use this logic for runtime-verification of previously mentioned systems. The existing past-time Distributed Temporal Logic (ptDTL), which is used for this area of responsibility, is taken as basis and a new logic with impartiality, anticipation and three-valued semantics is defined, properties which ptDTL does not possess. Furthermore an automaton model for formulas of this new logic and a transformation of this formulas in this model is developed.

Erklärung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur angefertigt habe.

_____

(Torben Scheffel)
Lübeck, 20.12.2013

Inhaltsverzeichnis

1 Einleitung

Die Sicherheit von Softwaresystemen zu gewährleisten ist sehr wichtig. Systeme, die, zum Beispiel, in Flugzeugen oder Autos genutzt werden, dürfen nicht ausfallen oder fehlerhaft funktionieren, da bei einem Fehler in einem solchen System direkt Menschen zu Schaden kommen könnten.

Mittels Laufzeitverifikation wird versucht, solche sicherheitskritischen Systeme während der Laufzeit zu überwachen. Dabei werden ein oder mehrere Monitore nebenläufig zu dem sicherheitskritischen System ausgeführt. Jeder Monitor überwacht bestimmte Eigenschaften des Systems, um etwaige Ausfälle oder Fehlverhalten zu diagnostizieren. Nach einer solchen Diagnose, zum Beispiel einem Wert in einer Variable, der dort nicht sein darf, können entsprechende Maßnahmen unternommen werden, damit es nicht zu einem Totalausfall kommt.

Ein Teilgebiet der Laufzeitverifikation ist die sogenannte „verteilte Laufzeitverifikation“. Hierbei wird ein verteiltes System durch einen oder mehrere Monitore überwacht. Dies können ganz unterschiedliche Arten von Systemen sein, unter anderem Netzwerksysteme, bei denen ein Monitor nur die Ein- und Ausgänge der Netzwerkports betrachtet oder auch ein System mit verschiedenen Prozessen auf derselben Einheit des Systems, bei denen jeder Prozess von einem oder mehreren Monitoren überwacht wird, die als ein großer Monitor, organisiert sind. Gibt es, wie in dem gerade genannten Fall, keinen zentralen Monitor sondern verteilte Monitore, von denen jeder eine eigene Einheit bildet, so spricht man auch von dezentralisierter Laufzeitverifikation. Eine besonders interessante und komplexe Art von dezentralisierter Laufzeitverifikation ist die, bei der mehrere Prozesse auf verschiedenen Einheiten des Gesamtsystems, die miteinander kommunizieren, überwacht werden sollen. Dabei gibt es für jeden Prozess einen oder mehrere Monitore auf dem entsprechenden Medium, allerdings stehen dem Monitor grundsätzlich zunächst keine Informationen über die anderen Prozesse außer seinem eigenen zur Verfügung. Soll daher eine Eigenschaft überwacht werden, die mehrere der Prozesse betrifft, so muss ein Monitor irgendwie an Informationen über die anderen Prozesse herankommen.

Weiterhin kann man in dem Gebiet der dezentralisierten Laufzeitverifikation noch zwischen Realisierungen auf synchronen und asynchronen Systemen unterscheiden. Bei synchronen Systemen ist es leichter möglich eine exakte Aussage über den Zustand des Systems zu treffen, da es dort einen synchronen Takt gibt und damit Zeitschritte gezählt werden können. Dadurch ist deutlich klarer, was eine Bedingung „Wenn auf Prozess p in Variable 1 eine 3 steht, so muss vorher auf Prozess q in Variable 2 mal eine 7 gewesen sein.“ meint. Es kann rückwirkend exakt bestimmt werden, was vor einigen Schritten auf Prozess q galt, nachdem Prozess p in Variable 1 eine 3 hatte. In asynchronen Systemen hingegen ist nicht klar, was gemeint ist. Es kann sein, dass Prozess q noch nicht mal einen Schritt durchgeführt hat wenn Prozess p in Variable 1 eine 3 hat. Oder Prozess q kann schon viel weiter sein als Prozess p und dadurch ist „muss vorher“ keine exakte Aussage

mehr. Der Einsatz von verteilter Laufzeitverifikation in asynchronen Systemen ist daher deutlich ungenauer.

Die Einsatzmöglichkeiten für verteilte Laufzeitverifikation sind vielseitig. So ist es beispielsweise möglich Einzelsysteme in einem System, wie einem Flugzeug oder Auto, durch einen oder mehrere Monitore zu überwachen, die auch Informationen über die anderen Einzelsysteme benötigen. Eine andere Möglichkeit für den Einsatz könnten mehrere Roboter sein, die sich selbstständig um ihre eigenen Aufgaben kümmern und abhängig von einander bestimmte Kriterien erfüllen müssen. Diese Kriterien könnten vorher mit einer Logik für solch ein verteiltes System spezifiziert und von, auf den Robotern laufenden, Monitoren überwacht werden. Das zuletzt genannte Einsatzbeispiel für verteilte Laufzeitverifikation wurde für Lego-Mindstorms Roboter in [Sch14] realisiert.

1.1 Bisherige Arbeiten

Es gibt viele Ansätze verteilte Laufzeitverifikation durchzuführen, je nach Zielsystem. In diesem Abschnitt werden einige der bisher zu dem Thema veröffentlichten Arbeiten kurz vorgestellt.

Andreas Bauer und Yliès Falcone haben in [BF12] einen Ansatz für synchrone Systeme entwickelt, bei dem von mehreren, beliebig verteilten Monitoren ausgegangen wird, die miteinander kommunizieren können. Jeder Monitor erhält zu Beginn dieselbe, global zu überprüfende Formel, die er verifizieren soll. Diese Formel kann beliebig viele Teilformeln enthalten, die von einem Monitor selber oder nur von anderen Monitoren überprüft werden können. Jeder Monitor verifiziert in jedem Schritt in der Formel das, was für ihn möglich ist und schickt danach die Restformel in einem Ring an einen anderen Monitor. Dies passiert so lange, bis ein Monitor die Formel irgendwann gesamt auswerten kann.

Alwyn Goodloe und Lee Pike haben in [GP10] Untersuchungen zum Thema verteilte Laufzeitverifikation in hart echtzeitfähigen Systemen vorgenommen. Dabei werden synchrone und asynchrone Systeme betrachtet. Sie gehen weniger auf eine genaue theoretische Definition oder praktische Umsetzung ein, sondern beschreiben mehrere Ideen, wie Laufzeitverifikation in derartigen Systemen umgesetzt werden könnte. Dabei betrachten sie viele verschiedene Ansätze und untersuchen unterschiedliche Systemarchitekturen und wie gut diese geeignet wären. Speziell wird noch mal der Fall des Monitoring in medizinischen Geräten und Fahrzeugen als Beispiel für hart echtzeitfähige Systeme betrachtet.

Koushik Sen, Abhay Vardhan, Gul Agha und Grigore Roşu haben mehrfach Arbeiten zu diesem Thema veröffentlicht. In [SVAR04] haben sie die, auf Vergangenheits-Operatoren basierende, past-time Distributed Temporal Logic (ptDTL) für asynchrone verteilte Systeme vorgestellt. Weiter haben sie in demselben Paper auch noch eine Idee zur Implementierung angegeben. Diese basiert darauf, dass die, von den Monitoren benötigten, Informationen per Knowledge-Vector für jeden Monitor selber gespeichert werden. Die Knowledge-Vectoren sollen durch Nachrichten zwischen den Einzelsystemen aktualisiert werden. Wird in einem Monitor der Wahrheitswert einer Formel eines anderen Monitors benötigt, so wird dieser aus dem Knowledge-Vector abgerufen. In [SVAR06] wurde der Inhalt von [SVAR04] für Multithread-Anwendungen angepasst.

Diese Arbeit basiert auf [SVAR04]. Die Ideen dieses Papers für asynchrone verteilte Laufzeitverifikation werden aufgegriffen und erweitert. Zu ptDTL wird eine entsprechende Logik mit Zukunfts-Operatoren angegeben sowie dieser noch Scheduler und Synchronisationsaktionen hinzugefügt. Danach wird noch ein passendes Automatenmodell definiert und eine Umwandlung in dieses beschrieben.

1.2 Aufbau

In Kapitel zwei werden alle Grundlagen erläutert, die für die weiteren Kapitel dieser Arbeit benötigt werden. Dazu gehören Worte, Verbände, Eigenschaften von Semantiken sowie die Definition der verschiedenen temporalen Eigenschaftsklassen. Danach werden auch noch alle benötigten Logiken definiert. Am Ende des Kapitels folgen noch Nachrichten und es wird eine Version von ptDTL angegeben, die auf diesen Nachrichten basiert.

Das dritte Kapitel behandelt die Definition der Logik, die in dieser Arbeit neu entwickelt werden sollte. Dazu wird dort zuerst eine Version von ptDTL definiert und analysiert, die auf Zukunfts-Operatoren basiert und eine dreiwertige Semantik besitzt. Danach wird ein Modell des zugrunde liegenden Systems angegeben und mit Hilfe von Schedulern ein Lauf dieses gesamten Systems dargestellt. Darauf folgt die Definition von Synchronisationsaktionen und der neuen Logik, basierend auf dem Modell des Systems, einem Scheduler und den Synchronisationsaktionen. Nach der Definition der Logik folgt noch eine Analyse ihrer Eigenschaften.

In dem vierten Kapitel wird das Automatenmodell für die neue Logik entwickelt. Dafür werden zuerst die benötigten Automaten definiert. Danach wird eine Umwandlung beschrieben, welche eine Formel der Logik nach und nach in verschiedene Automaten umwandelt, bis das gewünschte Modell erreicht ist. In den letzten beiden Teilabschnitten des Kapitels folgt zuerst eine Begrenzung des Modells auf endliche Größen und danach eine Analyse der Komplexität der Umwandlung und des entstehenden Modells.

Kapitel fünf ist die Zusammenfassung der Arbeit. Außerdem wird angegeben, was in zukünftigen Arbeiten noch bezüglich der neu entwickelten Logik und des Automatenmodells zu untersuchen ist.

2 Theoretische Grundlagen

In diesem Kapitel werden die theoretischen Grundlagen für den Rest dieser Arbeit vorgestellt. Zuerst werden Worte und Verbände eingeführt, welche für die später folgenden Definitionen der verschiedenen, für diese Arbeit benötigten, Logiken als Eingabe und als Ergebnis von Bedeutung sind. Darauf folgt eine Definition von Monitorbarkeit, welche angibt, wann eine Eigenschaft durch eine Logik monitorbar ist. Dies wird später benötigt, um für die verschiedenen Logiken zu untersuchen, welche Eigenschaften mit einer Logik monitorbar sind. Danach folgt die Definition der temporallogischen Eigenschaftsklassen Safety, Guarantee, Obligation, Response, Persistence und Reactivity. Die Klassen enthalten Eigenschaften und deshalb wird es durch diese Klassen einfacher anzugeben, welche Eigenschaften durch eine Logik ausdrück- oder monitorbar sind. Des Weiteren wird die Hierarchie der temporallogischen Eigenschaftsklassen dargestellt sowie erläutert, inwiefern Elemente dieser Klassen monitorbar sind oder nicht. Im letzten Abschnitt dieses Kapitels folgen dann die Definitionen der Linear Temporal Logic (LTL), LTL_3, der past-time Linear Temporal Logic (ptLTL) sowie der past-time Distributed Temporal Logic (ptDTL). Des Weiteren werden noch Nachrichten definiert und eine Semantik von ptDTL angegeben, die auf Nachrichten basiert.

Die Definitionen der Verbände sowie der Syntax und Semantiken von LTL und LTL_3 wurden von denen aus [Leu11] inspiriert.

2.1 Wort

In diesem Abschnitt wird ein Wort definiert. Worte werden später für die Definitionen der Logiken, das Verhalten der Prozesse und die Definitionen der Automaten benötigt. Da sowohl Logiken über unendliche Worte als auch Logiken über endliche Worte in dieser Arbeit vorkommen, werden in der folgenden Definition beide Arten von Worten definiert.

Definition 2.1 (Wort)**.** *Sei das Alphabet Σ eine endliche Menge. Ein Wort $w = w_0w_1w_2\cdots \in \Sigma^\omega$ wird als unendliches Wort bezeichnet und ein Wort $w = w_0w_1w_2\ldots w_n \in \Sigma^n$ als endliches Wort.*

Sei $w \in \Sigma^\omega$ ein unendliches Wort. Für w gilt dann folgendes:

- *$w_{\geq i}$ bezeichnet das Wort w ab dem i-ten Zeichen, daher $w_{\geq i} = w_iw_{i+1}w_{i+2}\ldots$*
- *$w_{i..j}$ gibt das Teilwort von Zeichen i bis Zeichen j des Wortes w an, daher, $w_{i..j} = w_iw_{i+1}w_{i+2}\ldots w_j$*

Sei $w \in \Sigma^n$ ein endliches Wort und $i, j \in \mathbb{N}, i \leq j \leq n$. Für w gilt dann folgendes:

- *$w_{\geq i}$ bezeichnet das Wort w ab dem i-ten Zeichen, daher $w_{\geq i} = w_i w_{i+1} w_{i+2} \dots w_n$*
- *$w_{i..j}$ gibt das Teilwort von Zeichen i bis Zeichen j des Wortes w an, daher, $w_{i..j} = w_i w_{i+1} w_{i+2} \dots w_j$*

Ein Wort ist also eine unendliche oder eine endliche Folge von Elementen eines Alphabetes.

2.2 Verbände

Verbände sind Mengen, die einer gewissen Ordnung unterliegen. Dadurch entsteht für die Elemente der Menge eine Hierarchie, welche später sowohl für die Definition von Logiken wichtig ist als auch entscheidet, welche Eigenschaft mit einer Logik monitorbar sind.

In diesem Abschnitt werden zuerst allgemein Verbände definiert und danach die zwei für diese Arbeit wichtigsten Verbände, der Verband $\mathbb{B}_2$ und der Verband $\mathbb{B}_3$, vorstellt. Diese werden im nächsten Abschnitt für die Definitionen der zwei- und dreiwertigen Logiken benutzt.

2.2.1 Definition und Darstellung

Definition 2.2 (Verband)**.** *Ein Verband ist eine partiell geordnete Menge $(\mathcal{L}, \sqsubseteq)$, in der für alle $x, y \in \mathcal{L}$ gilt:*

- *es existiert eine größte untere Grenze $x \sqcap y$ und*
- *es existiert eine kleinste obere Grenze $x \sqcup y$.*

Ein Verband heißt endlich gdw. $\mathcal{L}$ endlich ist. Für jeden endlichen Verband gibt es zwei wohl definierte, einzigartige Elemente:

- *ein kleinstes Element $\bot$ und*
- *ein größtes Element $\top$.*

Diese Definition definiert allgemein Verbände, in dieser Arbeit werden allerdings nur endliche Verbände verwendet. Weiter stellt diese Definition sicher, dass jeder Verband $\top$ und $\bot$ enthält und diese das einzigartige oberste beziehungsweise eindeutige unterste Element in der Hierarchie darstellen. Diese einzigartigen Elemente sind wichtig, da sie später in den Logiken über Verbänden, die mehr als zwei Elemente enthalten, Erfolg und Fehler repräsentieren.

Als nächstes werden zwei Eigenschaften von Verbänden definiert.

Definition 2.3 (Eigenschaften von Verbänden)**.** *Ein Verband $(\mathcal{L}, \sqsubseteq)$ heißt distributiver Verband gdw. für alle $x, y, z \in \mathcal{L}$ gilt*

- $x \sqcap (y \sqcup z) = (x \sqcap y) \sqcup (x \sqcap z)$

- $x \sqcup (y \sqcap z) = (x \sqcup y) \sqcap (x \sqcup z)$

Ein Verband $(\mathcal{L}, \sqsubseteq)$ heißt De-Morgan-Verband gdw. es für alle $x \in \mathcal{L}$ ein einzigartiges duales $\overline{x} \in \mathcal{L}$ gibt, sodass

- $\overline{\overline{x}} = x$ *und*

- $x \sqsubseteq y \Rightarrow \overline{y} \sqsubseteq \overline{x}$

gilt.

Ein distributiver, endlicher De-Morgan-Verband heißt Wahrheitsbereich.

Alle im Folgenden genutzten Verbände sind Wahrheitsbereiche, weil die beiden, eben definierten, Eigenschaften eines Wahrheitsbereiches wichtig für die Semantiken der Logiken sind. Wären die Verbände keine Wahrheitsbereiche, so könnten Hierarchien der Elemente des Verbandes angegeben werden, mit denen die Semantik einer Logik keinen Sinn mehr ergeben würde.

Die Darstellung von Verbänden erfolgt als Hasse-Diagramm. Dies ist eine gute Art der Darstellung, da Verbände endliche, partiell geordnete Mengen sind. Für alle $x, y \in \mathcal{L}$ gibt es in einem Hasse-Diagramm eine Verbindung aufwärts von x nach y, wenn $x \sqsubseteq y$ gilt und es kein $z \in \mathcal{L}$ gibt, so dass $x \sqsubseteq z \sqsubseteq y$ gilt.

2.2.2 Der Verband $\mathbb{B}_2$

Abbildung 2.1: Das Hasse-Diagramm für den Verband $\mathbb{B}_2$.

Der $\mathbb{B}_2$ ist ein Verband mit nur zwei Elementen und damit einer der einfachsten Verbände. Das zugehörige Hasse-Diagramme ist in Abbildung Abbildung 2.1 dargestellt.

In dem $\mathbb{B}_2$ sind die beiden Elemente zueinander dual. Das heißt, es gilt $\overline{\top} = \bot$ und $\overline{\bot} = \top$. Der $\mathbb{B}_2$ ist daher ein Wahrheitsbereich.

Alle zweiwertigen Semantiken der Logiken sind über dem $\mathbb{B}_2$ definiert.

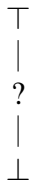


Abbildung 2.2: Das Hasse-Diagramm für den Verband $\mathbb{B}_3$.

2.2.3 Der Verband $\mathbb{B}_3$

Der $\mathbb{B}_3$ ist ein Verband mit drei Elementen. Das zugehörige Hasse-Diagramme ist in Abbildung 2.2 dargestellt.

In dem $\mathbb{B}_3$ sind das oberste und das unterste Element zueinander dual. Das heißt es gilt $\overline{\top} = \bot$ und $\overline{\bot} = \top$. Das dritte Element, ?, ist zu sich selbst dual, daher gilt $\overline{?} = ?$. Wie der $\mathbb{B}_2$ ist auch der $\mathbb{B}_3$ ein Wahrheitsbereich.

Alle dreiwertigen Semantiken der Logiken sind über dem $\mathbb{B}_3$ definiert.

2.3 Eigenschaften von Semantiken

In diesem Abschnitt werden die von Semantiken wünschenswerten Eigenschaften Impartiality und Anticipation definiert sowie erläutert, wann eine Formel einer Logik monitorbar ist. Weiter wird erklärt, welchen Nutzen diese Eigenschaften haben und warum es insbesondere im Bereich der Laufzeitverifikation wünschenswert ist, dass eine Semantik diese Eigenschaften besitzt. Ob und warum eine Semantik eine der Eigenschaften besitzt oder nicht wird in dem Abschnitt der entsprechenden Semantik beschrieben.

2.3.1 Impartiality

Eine Semantik ist impartial, wenn eine Formel über dieser Semantik für ein endliches Wort nur dann zu $\top$ oder $\bot$ ausgewertet wird, wenn sich dies durch beliebige Verlängerungen des Wortes nicht mehr ändern kann. Damit folgt diese Definition:

Definition 2.4 (Impartiality)**.** *Sei Σ ein Alphabet, $\mathbb{B}_x$ ein Wahrheitsbereich und φ eine Formel einer temporalen Logik TL, deren Semantik über $\mathbb{B}_x$ definiert ist. Die Semantik der temporalen Logik TL ist impartial, gdw.*

- $\forall w \in \Sigma^+ : [\![w \models \varphi]\!]_{TL} = \top$ *so folgt* $\forall u \in \Sigma^* : [\![wu \models \varphi]\!]_{TL} = \top$ *und*
- $\forall w \in \Sigma^+ : [\![w \models \varphi]\!]_{TL} = \bot$ *so folgt* $\forall u \in \Sigma^* : [\![wu \models \varphi]\!]_{TL} = \bot$.

gilt.

Ist $\mathbb{B}_x$ zweiwertig, so ist die Semantik impartial, wenn eine der beiden Bedingungen erfüllt ist.

Alle $b \in \mathbb{B}_x$, für die die Bedingungen erfüllt sind, werden auch als impartial bezeichnet.

Ist eine Semantik impartial, so weiß man, dass die Formel ab einer bestimmten Position des Wortes endgültig erfüllt oder endgültig verletzt ist, wenn sich an dieser Position $\top$ oder $\bot$ ergeben. Dadurch kann man genauere Aussagen über ein System treffen, dessen Verhalten der Formel entsprechen soll, da man ab dieser Position eindeutige Aussagen hat. Für die Laufzeitverifikation ist dies eine essentiell wichtige Eigenschaft, denn ohne, dass eine Semantik Impartiality besitzt, kann man aufgrund von Wahrheitswerten nie sagen, ob ein System einen Fehler hat oder nicht, da sich jeder Wahrheitswert wieder ändern kann.

2.3.2 Anticipation

Anticipation heißt, dass eine Formel so früh wie möglich zu $\top$ oder $\bot$ ausgewertet wird. Sobald für eine Formel also feststeht, dass in endlich vielen Schritten $\top$ oder $\bot$ herauskommen muss, so sollte dies schon ab dem Zeitpunkt, seit dem dies feststeht, erkannt werden.

Definition 2.5 (Anticipation)**.** *Sei TL eine temporale Logik, Σ ein Alphabet und φ eine Formel der temporalen Logik TL. Die Semantik von TL besitzt Anticipation gdw.*

- $\forall w \in \Sigma^+ : (\forall u \in \Sigma^* : [\![wu \models \varphi]\!]_{TL} = \top) \Rightarrow [\![w \models \varphi]\!]_{TL} = \top$ *und*
- $\forall w \in \Sigma^+ : (\forall u \in \Sigma^* : [\![wu \models \varphi]\!]_{TL} = \bot) \Rightarrow [\![w \models \varphi]\!]_{TL} = \bot$

gilt.

Diese Definition sagt aus, dass eine Semantik einer temporalen Logik anticipatory heißt, wenn ein endlicher Lauf zu demselben Wahrheitswert ausgewertet wird wie seine endlichen Fortsetzungen, wenn alle seine Fortsetzungen zu demselben Wahrheitswert ausgewertet werden.

Der Vorteil an Anticipation ist, dass dadurch früher erkannt wird, wann man eine endgültige Aussage treffen kann. In Systemen hat dies den Vorteil, dass Fehler und Erfolge so schnell wie möglich erkannt werden und darauf reagiert werden kann, wenn gewollt. Dies kann, zum Beispiel, dazu führen, dass Fehler vor einem kritischen Zeitpunkt erkannt werden statt erst danach.

Ein Beispiel hierfür wäre die Aussage „Erfülle im übernächsten Schritt etwas unerfüllbares“. Das System würde nun normalerweise zwei Schritte machen und in beiden wäre noch nicht bekannt, was insgesamt herauskommt. Beim dritten Schritt wird dann erkannt, dass das unerfüllbare nicht erfüllt werden kann und es wird $\bot$ ausgegeben. Ist die Semantik der Logik anticipatory, so würde sie schon zwei Schritte bevor das unerfüllbare erfüllt werden soll $\bot$ ergeben, da, egal was passiert, das unerfüllbare im übernächsten Schritt immer zu $\bot$ führen würde.

Um Anticipation sicherzustellen muss ein Erfüllbarkeitstest für die Formel durchgeführt werden. Ein Weg um dies zu erreichen ist es, die Formel in einen Automaten umzuwandeln. Auf diesem

können dann durch einen Leerheitstest alle Zustände gefunden werden, von denen aus die Formel noch erfüllt werden kann. Führt man dieselbe Prozedur mit dem Automaten für die negierte Formel durch, erhält man einen Automaten der durch die nicht-akzeptierenden Zustände ausdrückt, wann die Formel endgültig erfüllt ist und einen der ausdrückt, wann die Formel endgültig nicht erfüllt ist. Näheres dazu befindet sich in [Leu11].

2.3.3 Monitorbarkeit

Monitorbar heißt, dass eine Eigenschaft durch einen Monitor sinnvoll überwacht werden kann. In verschiedenen Definitionen der Monitorbarkeit wird das „sinnvoll“ allerdings verschieden definiert. Im Allgemeinen kann man sagen, dass eine Formel sinnvoll durch einen Monitor überwacht werden kann, wenn Fehler oder Erfolge anhand eines endlichen Präfixes des unendlichen Laufes erkannt werden können.

Wie vorher erwähnt, gibt es mehrere mögliche Definitionen von Monitorbarkeit. Von diesen wurden einige in [FFM12] vorgestellt und untersucht. In dieser Arbeit wird eine Erweiterung der Definition aus [BLS11] verwendet.

Die Definition aus [BLS11] sagt, dass eine Eigenschaft monitorbar ist, wenn es keinen endlichen Präfix eines unendlichen Wortes gibt, ab dem es nicht mehr nach endlicher Zeit möglich ist, herauszufinden, ob die Eigenschaft erfüllt oder verletzt wird. Ein solcher Präfix wird auch Ugly-Präfix genannt.

Definition 2.6 (Monitorbarkeit nach [BLS11])**.** *Sei Σ ein Alphabet. Eine Formel φ einer temporalen Logik TL heißt genau dann monitorbar nach [BLS11], wenn es keinen Ugly-Präfix gibt, daher, wenn*

$$\forall u \in \Sigma^+ : \exists v \in \Sigma^* : (\forall w \in \Sigma^* : [\![uvw \models \varphi]\!]_{TL} = \top) \vee (\forall w \in \Sigma^* : [\![uvw \models \varphi]\!]_{TL} = \bot).$$

In [FFM12] wurde diese Definition durch eine Parametrisierung mit Verbänden erweitert. Dadurch ist eine Formel in Abhängigkeit der Semantik der Logik und des genutzten Verbandes monitorbar oder nicht. Diese Definition von Monitorbarkeit wird im weiteren Verlauf der Arbeit genutzt.

Definition 2.7 (Monitorbarkeit)**.** *Sei Σ ein Alphabet. Eine Formel φ einer temporalen Logik TL mit einer Semantik über einem Wahrheitsbereich $\mathbb{B}$ heißt monitorbar, gdw.:*

$$\forall u \in \Sigma^+ : \exists b \in \mathbb{B} : \exists v \in \Sigma^* : (b = \top \vee b = \bot) \wedge b \textit{ ist impartial} \wedge [\![uv \models \varphi]\!]_{TL} = b$$

Diese Definition sagt aus, dass eine Formel φ genau dann mit einer Semantik und einem Wahrheitsbereich monitorbar ist, wenn sich jeder endliche Präfix eines unendlichen Wortes mit einer endlichen Verlängerung so erweitern lässt, dass entweder alle weiteren, endlichen Fortsetzungen des verlängerten Präfixes φ erfüllen oder alle φ nicht erfüllen. Dies entspricht dem Inhalt der Definition 2.6, außer, dass noch Impartiality gefordert wird. Die Forderung nach Impartiality ist wichtig

zum Monitoren von Eigenschaften, da auf bestimmte Wahrheitswerte einer Formel bestimmte Reaktionen folgen sollen. Dafür muss man sich aber sicher sein können, dass sich der Wahrheitswert nicht wieder ändert, da man sonst nie weiß, wann wirklich ein Fehler oder ein Erfolg eingetreten ist.

Was für Eigenschaften mit einer bestimmten Semantik und einem Wahrheitsbereich mit dieser Definition monitorbar sind, wird in Unterabschnitt 2.4.1 auf Seite 14 betrachtet.

2.4 Temporallogische Eigenschaftsklassen

Temporallogische Eigenschaftsklassen sind Mengen von Eigenschaften, die von temporalen Logiken dargestellt werden können. Dabei haben die Eigenschaften einer Klasse bestimmte charakteristische Eigenarten. Diese Klassen wurden von Zohar Manna und Amir Pnueli in [MP90] erstmals untersucht und eine Hierarchie dieser Klassen aufgestellt sowie deren Charakteristik erläutert. Die Eigenschaftsklassen wurden in dem Artikel allgemein über Präfixe definiert, unabhängig von einer bestimmten Logik. Im späteren Verlauf wurde allerdings für jede Klasse eine charakteristische Formel angegeben, welche die Grundstruktur der Formeln einer Klasse widerspiegelt.

Im weiteren Verlauf dieses Abschnittes werden die Präfix-Definitionen aus [MP90] noch einmal wiedergegeben und genauer erläutert, aus was für Eigenschaften die einzelnen Klassen bestehen sowie noch die Hierarchie und die charakteristischen Formeln für die Klassen angegeben. Weiter wird die Monitorbarkeit der temporallogischen Eigenschaftsklassen unter bestimmten Verbänden und Eigenschaften von Semantiken betrachtet.

Zuerst wird die Definition der Eigenschafts-Operatoren wiedergegeben, die für die Definition der Eigenschaftsklassen benötigt werden.

Definition 2.8 (Eigenschafts-Operatoren)**.** *Sei* Σ *ein Alphabet,* $\Phi \subseteq \Sigma^+$ *eine beliebige, endliche Eigenschaft und* $Pref : \Sigma^\omega \to 2^{\Sigma^*}$ *mit*

$$Pref(\sigma) = \{w \in \Sigma^* \mid \exists u \in \Sigma^\omega : wu = \sigma\}$$

Eine Funktion, die für ein unendliches Wort die Menge aller endlichen Präfixe dieses Wortes zurückgibt. Für die vier Eigenschafts-Operatoren A, E, R, P *folgt:*

- $A(\Phi) = \{\sigma \in \Sigma^\omega \mid Pref(\sigma) \subseteq \Phi\}$
- $E(\Phi) = \{\sigma \in \Sigma^\omega \mid \exists w \in Pref(\sigma) : w \in \Phi\}$
- $R(\Phi) = \{\sigma \in \sigma^\omega \mid \exists M \subseteq Pref(\sigma) : |M| = \infty \wedge M \subseteq \Phi\}$
- $P(\Phi) = \{\sigma \in \sigma^\omega \mid \forall M \subseteq Pref(\sigma) : |M| = \infty \to M \subseteq \Phi\}$

Informell kann A als die Menge aller unendlichen Worte, von denen alle endlichen Präfixe zu Φ gehören, beschrieben werden. E enthält alle unendlichen Worte, von denen mindestens ein Präfix zu Φ gehört. R beinhaltet alle unendlichen Worte, von denen unendlich viele endliche Präfixe zu Φ gehören und P enthält alle unendlichen Worte, für die es nur endlich viele endliche Präfixe gibt, die nicht zu Φ gehören.

Als nächstes werden die temporralogischen Eigenschaftsklassen mit Hilfe der eben definierten Eigenschaft-Operatoren definiert.

Definition 2.9 (Temporallogische Eigenschaftsklassen)**.** *Sei* $\Pi \subseteq \Sigma^\omega$ *eine unendliche Eigenschaft.* Π *gehört zu der temporralogischen Eigenschaftsklasse*

- *Safety, gdw. gilt*
$$\exists \Phi \subseteq \Sigma^+ : \Pi = A(\Phi)$$
- *Guarantee, gdw. gilt*
$$\exists \Phi \subseteq \Sigma^+ : \Pi = E(\Phi)$$
- *Obligation, gdw. gilt*
$$\exists n \in \mathbb{N} : \exists \Phi_1, \Phi_2, \ldots, \Phi_n, \Psi_1, \Psi_2, \ldots, \Psi_n \subseteq \Sigma^+ : \Pi = \bigcap_{i=0}^{n} (A(\Phi_i) \cup E(\Psi_i))$$
- *Response, gdw. gilt*
$$\exists \Phi \subseteq \Sigma^+ : \Pi = R(\Phi)$$
- *Persistence, gdw. gilt*
$$\exists \Phi \subseteq \Sigma^+ : \Pi = P(\Phi)$$
- *Reactivity, gdw. gilt*
$$\exists n \in \mathbb{N} : \exists \Phi_1, \Phi_2, \ldots, \Phi_n, \Psi_1, \Psi_2, \ldots, \Psi_n \subseteq \Sigma^+ : \Pi = \bigcap_{i=0}^{n} (R(\Phi_i) \cup P(\Psi_i))$$

Safety ist die Menge aller Eigenschaften, bei denen nicht erfüllende Worte schon nach endlicher Zeit erkannt werden, indem bei einem Präfix des Wortes bereits festgestellt wird, dass ab diesem Präfix die Eigenschaft nicht mehr erfüllt werden kann. Selbes gilt für Guarantee, nur, dass dabei nach endlicher Zeit erkannt werden muss, dass ein Wort die Eigenschaft erfüllt. Guarantee enthält damit genau alle Negationen der Safety-Eigenschaften. Obligation enthält alle positiven booleschen Verknüpfungen von Safety- und Guarantee-Eigenschaften. Bei Obligation-Eigenschaften gibt es daher im Allgemeinen sowohl Präfixe von Worten, ab denen klar ist, dass die Eigenschaft für alle Fortsetzungen erfüllt ist sowie Präfixe, ab denen klar ist, dass sie für alle Fortsetzungen nicht erfüllt ist. Response enthält alle Eigenschaften, die, wenn ein Wort sie erfüllt, auch von unendlich vielen Präfixen des Wortes erfüllt werden. Das heißt, es gibt immer wieder Präfixe von

einem Wort, dass eine Response-Eigenschaft erfüllt, die aktuell die Eigenschaft erfüllen. Ein längerer Präfix des Wortes kann die Eigenschaft dann wieder nicht erfüllen, aber es gibt immer wieder einen noch längeren, der die Eigenschaft wiederum erfüllt. Bei Persistence ist es so, dass nur endlich viele Präfixe eines erfüllenden Wortes die Eigenschaft nicht erfüllen dürfen. Es gibt also irgendwann einen Präfix, ab dem alle längeren Präfixe des Wortes die Eigenschaft erfüllen. Persistence enthält daher alle Negationen von Response-Eigenschaften. Reactivity ist die all-Klasse. Sie enthält alle regulären Eigenschaften.

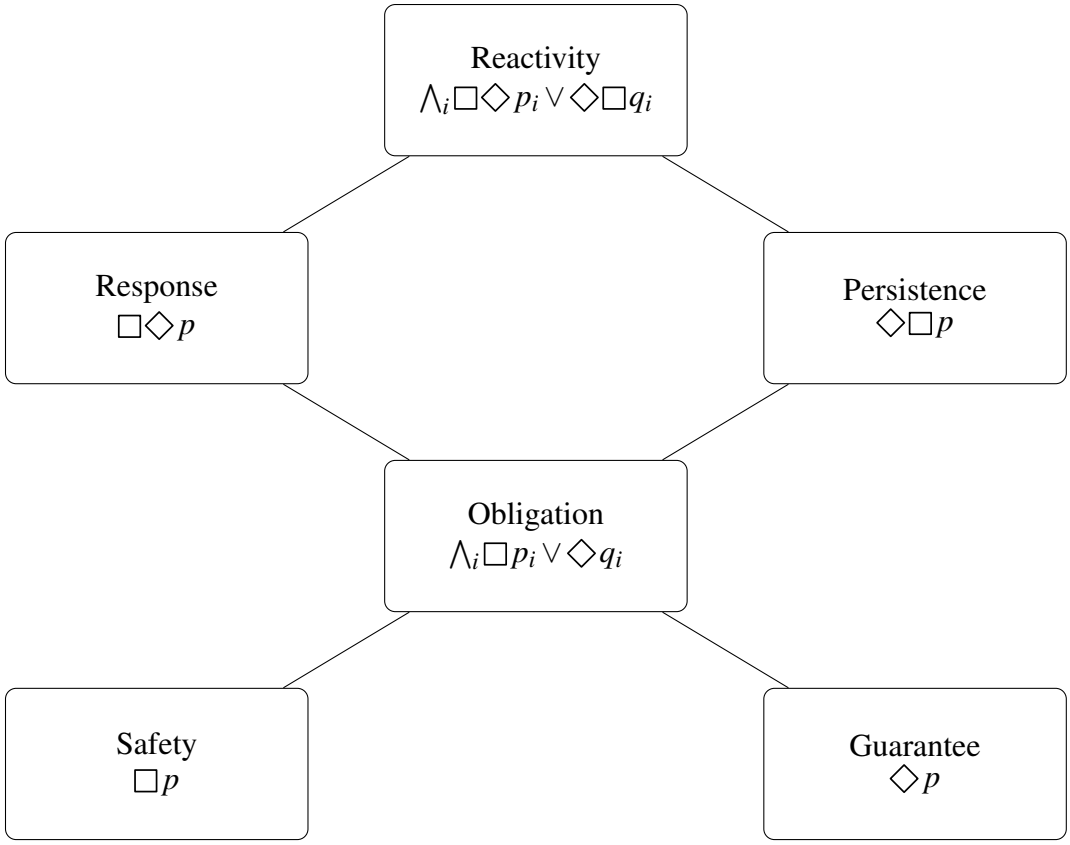


Abbildung 2.3: Die Hierarchie der temporallogischen Eigenschaftsklassen. p, p_i, q und q_i sind Formeln, die ausschließlich aus Vergangenheits-Operatoren, also Operatoren, die von einer Stelle eines Wortes in die Vergangenheit blicken, bestehen.

In Abbildung 2.3 ist die Hierarchie der Klassen dargestellt. Die Klasse Response wird auch in älteren Arbeiten als Recurrence bezeichnet. Jede weiter unten gelegene Klasse ist in den darüber liegenden enthalten, dies stellen die Verbindungen dar. Weiter ist jede Klasse eine echte Obermenge der Klassen, die in ihr enthalten sind. Für jede Eigenschaftsklasse gibt es eine charakteristische Formel für die Eigenschaften, die in der Klasse enthalten sind. Sie ist in Abbildung 2.3 unterhalb der Klassennamen angegeben. Diese charakteristischen Formeln geben Hinweise darauf, zu welcher Klasse eine Formel aus den, später definierten, temporalen Logiken LTL und ptLTL und deren darauf folgenden Erweiterungen gehört.

2.4.1 Monitorbarkeit von temporallogischen Eigenschaftsklassen

In diesem Abschnitt wird die Monitorbarkeit der Klassen bezüglich der, vorher definierten, Verbände $\mathbb{B}_2$ und $\mathbb{B}_3$ betrachtet. Neben den Verbänden sind aber auch die Eigenschaften der Semantik der Logik wichtig für die Monitorbarkeit. Die folgenden Aussagen wurden in [FFM12] bewiesen.

Für den Verband $\mathbb{B}_2$, ohne, dass die Semantik der Logik Impartiality besitzt, sind keine Eigenschaften monitorbar, was auch direkt aus der Definition der Monitorbarkeit folgt. Wie vorher schon beschrieben ist dann das Problem, dass ein Wahrheitswert niemals endgültig ist, da er sich jederzeit wieder ändern könnte. Ist die Semantik impartial bezüglich $\bot$, so sind alle Safety-Eigenschaften monitorbar. Dies folgt daraus, dass eine Eigenschaft φ nur genau dann eine Safety-Eigenschaft ist, wenn es einen endlichen Präfix jedes Wortes gibt, dass φ nicht erfüllt, so dass alle unendlichen Fortsetzungen des Präfixes φ nicht erfüllen. Ist die Semantik impartial bezüglich $\top$, so sind alle Guarantee-Eigenschaften monitorbar. Dies folgt mit einer analogen Begründung wie bei den Safety-Eigenschaften, da jede Guarantee-Eigenschaft eine negierte Safety-Eigenschaft ist. Es ist noch möglich einige Obligation-Eigenschaften zu monitoren, allerdings ist es mit keinem zweiwertigen Verband möglich, die gesamte Eigenschaftsklasse Obligation oder darüber liegende Klassen zu monitoren. Dies liegt daran, dass mit einem zweiwertigen Verband nicht einmal alle Safety- und alle Guarantee-Eigenschaften gleichzeitig monitorbar sind. Dafür müssten $\top$ und $\bot$ impartial sein und deshalb sind Verbände mit mehr als zwei Elementen nötig.

Für den Verband $\mathbb{B}_3$ sind auch nur Eigenschaften monitorbar, wenn die Semantik impartial ist. Dies folgt als demselben Grund wie bei dem Verband $\mathbb{B}_2$. Im Gegensatz zu dem Verband $\mathbb{B}_2$ sind mit dem Verband $\mathbb{B}_3$ allerdings sowohl Safety- als auch Guarantee-Eigenschaften monitorbar, wenn die Semantik impartial ist. Dies liegt daran, dass mit dem Verband $\mathbb{B}_3$ sowohl erfüllende als auch nicht-erfüllende Präfixe betrachtet werden können. Weiter sind auch alle Eigenschaften der Klasse Obligation monitorbar. Da dies nur Verundungen und Veroderungen von Safety- und Guarantee-Eigenschaften sind, sind sie monitorbar, weil sowohl erfüllende als auch nicht-erfüllende Präfixe betrachtet werden können, im Gegensatz zu dem Verband $\mathbb{B}_2$. Wie in [FFM12] gezeigt wurde, gibt es auch noch Eigenschaften in den Response und Persistence Klassen, die mit dem Verband $\mathbb{B}_3$ monitorbar sind, wie, zum Beispiel, die Eigenschaft „Es gilt immer: Wenn an einer Position *a* gilt, so muss danach solange *b* gelten, bis *c* gilt.“. Zum Einen ist $\bot$ impartial und zum Anderen gibt es immer eine Möglichkeit, die Eigenschaft nach endliche vielen Schritten zu verletzen (daher, zu $\bot$ auszuwerten), indem nacheinander nur zwei mal die Proposition *a* auftritt. Es gibt aber auch Formeln in Response und Persistence, die nicht mit dem Verband $\mathbb{B}_3$ monitorbar sind. Dazu gehört die Eigenschaft „Es muss immer mal wieder *a* gelten“. Diese Eigenschaft kann nach endlich vielen Schritten nie verletzt und nie erfüllt werden, da man nie weiß, ob noch ein *a* kommt oder nicht. Um zu sehen, dass kein *a* mehr auftritt, muss aber unendlich weit in die Zukunft geblickt werden. Aus diesem Grund ist diese Response-Eigenschaft nicht monitorbar.

2.5 Bekannte Logiken

Im Folgenden werden verschiedene Logiken definiert, die im weiteren Verlauf dieser Arbeit benötigt werden. Zuerst wird die Linear Temporal Logic (LTL) mit zwei- und dreiwertiger Semantik definiert. Darauf folgt die Vergangenheitsvariante von LTL, die past-time Linear Temporal Logic (ptLTL), mit einer zweiwertigen Semantik. Als letztes wird die Erweiterung von ptLTL für verteilte, asynchrone Systeme, die Past-Time Distributed Temporal Logic (ptDTL) aus [SVAR04], vorgestellt.

Alle Logiken in dieser Arbeit sind auf endlichen Präfixen unendlicher Worte oder auf unendlichen Worten definiert. Weiter sind alle Logiken entweder über dem Wahrheitsbereich $\mathbb{B}_2$ oder $\mathbb{B}_3$ definiert. Näheres dazu wird bei den Definitionen der einzelnen Logiken angegeben. Die Eigenschaften einer Semantik, Impartiality und Anticipation, werden nur bei den Semantiken diskutiert, bei denen dies für den weiteren Verlauf dieser Arbeit wichtig ist.

2.5.1 Linear Temporal Logic

In diesem Abschnitt werden zuerst die Syntax und danach die Semantiken der Linear Temporal Logic (LTL) angegeben. LTL basiert rein auf Zukunfts-Operatoren. Eine Semantik für LTL auf unendlichen Worten wird über dem Verband $\mathbb{B}_2$ angegeben und LTL genannt und eine auf endlichen Präfixen unendlicher Worte wird über dem Verband $\mathbb{B}_3$ angegeben, welche mit LTL_3 bezeichnet wird. Danach werden noch die Eigenschaften von LTL_3 beschrieben.

Definition 2.10 (LTL-Syntax)**.** *Sei a eine atomare Proposition aus einer endlichen Menge AP von atomaren Propositionen. Die Menge der LTL-Formeln ist induktiv über die folgende Grammatik definiert:*

$$\begin{array}{rlllllllllll} \varphi ::= & \mathit{true} & | & a & | & \varphi \vee \varphi & | & \bigcirc \varphi & | & \varphi \mathcal{U} \varphi & | & \Diamond \varphi & | \\ & \mathit{false} & | & \neg a & | & \varphi \wedge \varphi & | & & & \varphi \mathcal{R} \varphi & | & \Box \varphi & | \\ & \neg \varphi & & & & & & & & & & & \end{array}$$

Definition 2.11 (LTL-Semantik)**.** *Sei AP eine Menge atomarer Proposition, $a \in AP$ eine atomare Proposition und $\Sigma = 2^{AP}$ das Alphabet. Sei weiter $w \in \Sigma^\omega$ ein Wort. Die Semantik von LTL ist durch die Auswertungsfunktion $[\![\cdot]\!]_{LTL} : \Sigma^\omega \times LTL \to \mathbb{B}_2$ wie folgt definiert:*

$$[\![w \models \varphi]\!]_{LTL} = \begin{cases} \top & \text{wenn } w \models \varphi \\ \bot & \text{sonst} \end{cases}.$$

Die Relation $\models \subseteq \Sigma^\omega \times LTL$ ist dabei induktiv wie folgt definiert:

$$
\begin{array}{ll}
w \models \mathit{true} & \text{für alle } w \\
w \models a & \text{gdw. } a \in w_0 \\
w \models \neg\varphi & \text{gdw. } w \not\models \varphi \\
w \models \varphi \vee \psi & \text{gdw. } w \models \varphi \text{ oder } w \models \psi \\
w \models \bigcirc\varphi & \text{gdw. } w_{\geq 1} \models \varphi \\
w \models \varphi \mathcal{U} \psi & \text{gdw. } \exists i : w_{\geq i} \models \psi \wedge \forall j < i : w_{\geq j} \models \varphi
\end{array}
$$

Für die Operatoren sowie false gelten die üblichen Äquivalenzen:

$$
\begin{array}{ll}
\mathit{false} :\equiv \neg\mathit{true} & \varphi \wedge \psi :\equiv \neg(\neg\varphi \vee \neg\psi) \\
\varphi \rightarrow \psi :\equiv \neg\varphi \vee \psi & \Diamond\varphi :\equiv \mathit{true}\,\mathcal{U}\,\varphi \\
\Box\varphi :\equiv \neg\Diamond\neg\varphi & \varphi \mathcal{R} \psi :\equiv \neg(\neg\varphi \mathcal{U} \neg\psi)
\end{array}
$$

Definition 2.12 (LTL_3 Semantik)**.** *Sei AP eine endliche Menge von Propositionen und $\Sigma = 2^{AP}$ das Alphabet. Sei weiter φ eine LTL-Formel und $w \in \Sigma^+$. Die Semantik von LTL_3 ist durch die Auswertungsfunktion $[\![.]\!]_{LTL_3} : \Sigma^+ \times LTL \rightarrow \mathbb{B}_3$ wie folgt definiert:*

$$
[\![w \models \varphi]\!]_{LTL_3} = \begin{cases} \top & \text{wenn } \forall u \in \Sigma^\omega : [\![wu \models \varphi]\!]_{LTL} = \top \\ \bot & \text{wenn } \forall u \in \Sigma^\omega : [\![wu \models \varphi]\!]_{LTL} = \bot \\ ? & \text{sonst.} \end{cases}.
$$

Im Gegensatz zur zweiwertigen Semantik entscheidet die dreiwertige Semantik von LTL anhand von endlichen Präfixen eines unendlichen Wortes, ob eine Formel erfüllt ist oder nicht. Dies tut sie allerdings erst, sobald sicher ist, dass sich an dem Wahrheitswert nichts mehr ändert. Bis zu diesem Zeitpunkt wird ? ausgegeben, was ausdrückt, dass noch keine endgültige Aussage getroffen werden kann.

Der Vorteil von LTL_3 ist, dass die beiden Werte $\top$ und $\bot$ nur noch ausgegeben werden, wenn diese sich nicht mehr ändern können. LTL_3 ist damit impartial.

Ein weiterer Vorteil ist, dass LTL_3 Anticipation hat. Dies liegt daran, dass die Semantik so definiert ist, dass sie endgültige Ergebnisse von $\top$ oder $\bot$ durch das Betrachten aller unendlichen Verlängerungen des endlichen Präfixes möglichst früh erkennt.

Das Beispiel, welches schon bei der Definition von Anticipation betrachtet wurde, kann jetzt Anhand von LTL_3 noch einmal konkret betrachtet werden. Dafür wird die Formel $\bigcirc\bigcirc$false genommen. Normalerweise müsste man jetzt bis zum dritten Zeichen des Wortes laufen, um festzustellen, dass false nicht erfüllt wird. Damit wären die Ergebnisse der drei Schritte ?, ? und $\bot$. Man erkennt auf den ersten Blick, dass false nie erfüllt werden kann und daher auch $\bigcirc\bigcirc$false nicht. Dank der Anticipation merkt die Logik direkt, dass sich schon beim ersten Zeichen $\bot$ ergibt.

2.5.2 Past-Time Linear Temporal Logic

Die past-time Linear Temporal Logic (ptLTL) ist eine Variante von LTL, die nur auf Operatoren basiert, welche ein Wort ab einer bestimmten Stelle in die Vergangenheit betrachten. Die Vergangenheits-Operatoren, die in ptLTL genutzt werden, wurden unter anderem von Zohar Manna und Amir Pnueli in [MP90] vorgestellt.

Definition 2.13 (ptLTL-Syntax)**.** *Sei a eine atomare Proposition aus einer endlichen Menge AP von atomaren Propositionen. Die Menge der ptLTL-Formeln ist induktiv über die folgende Grammatik definiert:*

$$\begin{array}{rccccccccccc} \varphi ::= & \mathit{true} & | & a & | & \varphi \vee \varphi & | & \ominus \varphi & | & \varphi \mathcal{S} \varphi & | & \diamondsuit \varphi & | \\ & \mathit{false} & | & \neg a & | & \varphi \wedge \varphi & | & \circledcirc \varphi & | & \varphi \mathcal{B} \varphi & | & \boxminus \varphi & | \\ & \neg \varphi & & & & & & & & & & & \end{array}$$

Die Syntax ist ähnlich zu der von LTL, nur, dass alle Zukunfts-Operatoren durch die entsprechenden Vergangenheits-Operatoren ersetzt wurden. Zusätzlich gibt es noch ein Weak-Previous ($\circledcirc$), welches aufgrund des eindeutig definierten Anfangs bei dem Blick in die Vergangenheit benötigt wird. Das weak-Previous verhält sich anders als ein normales Previous ($\ominus$), wenn der Anfang des Wortes erreicht ist.

Die Semantik wird in der folgenden Definition beschrieben und ist über dem $\mathbb{B}_2$ sowie in dieser Arbeit über endlichen Worten definiert.

Definition 2.14 (ptLTL-Semantik)**.** *Sei AP eine Menge atomarer Proposition, $a \in AP$ eine atomare Proposition und $\Sigma = 2^{AP}$ das Alphabet. Sei weiter $w \in \Sigma^+$ ein Wort und $i \in \mathbb{N}$ mit $0 \leq i \leq |w|$. Die Semantik von ptLTL ist durch die Auswertungsfunktion $[\![\cdot]\!]_{ptLTL} : \Sigma^+ \times \mathbb{N} \times ptLTL \to \mathbb{B}_2$ wie folgt definiert:*

$$[\![w, i \models \varphi]\!]_{ptLTL} = \begin{cases} \top & \text{wenn } w, i \models \varphi \\ \bot & \text{sonst} \end{cases}.$$

Dabei ist die Relation $\models \subseteq \Sigma^+ \times \mathbb{N} \times ptLTL$ induktiv wie folgt definiert:

$$\begin{array}{ll} w, i \models \mathit{true} & \text{für alle } w, i \\ w, i \models a & \text{gdw. } a \in w_i \\ w, i \models \neg \varphi & \text{gdw. } w, i \not\models \varphi \\ w, i \models \varphi \vee \psi & \text{gdw. } w, i \models \varphi \text{ oder } w, i \models \psi \\ w, i \models \ominus \varphi & \text{gdw. } i > 0 \wedge w, i-1 \models \varphi \\ w, i \models \varphi \mathcal{S} \psi & \text{gdw. } \exists j \leq i : w, j \models \psi \wedge \forall j < k \leq i : w, k \models \varphi \end{array}$$

Für die Operatoren sowie false gelten die üblichen Äquivalenzen:

$$
\begin{array}{ll}
\mathit{false} :\equiv \neg \mathit{true} & \varphi \wedge \psi :\equiv \neg(\neg\varphi \vee \neg\psi) \\
\varphi \rightarrow \psi :\equiv \neg\varphi \vee \psi & \circledcirc\varphi :\equiv \neg\ominus\neg\varphi \\
\diamondsuit\varphi :\equiv \mathit{true}\,\mathcal{S}\,\varphi & \boxminus\varphi :\equiv \neg\diamondsuit\neg\varphi \\
\varphi\,\mathcal{B}\,\psi :\equiv \neg(\neg\varphi\,\mathcal{S}\,\neg\psi) &
\end{array}
$$

Auch die Semantik sieht der von LTL ähnlich. Alle temporalen Operatoren wurden durch ihre entsprechende Vergangenheitsform ersetzt, wodurch nun von einer festen Position aus in die Vergangenheit geschaut wird.

2.5.3 Prozesse

Als nächstes werden Prozesse beschrieben. Diese werden für die darauf folgenden Definition der Logik ptDTL benötigt, da diese eine Formel auf einem Prozess auswertet. Da die Prozesse in [SVAR04] nicht genauer beschrieben wurden, wird in diesem Unterabschnitt auch nur eine kurze Erläuterung gegeben, was ein Prozess ist.

Ein Prozess ist eine Einheit eines Gesamtsystems und in einem Zustand eines Prozesses gelten immer bestimmte Propositionen. Zu jedem Prozess p gehört eine exklusive Menge von Propositionen AP^p, aus denen sich das Alphabet $\Sigma^p = 2^{AP^p}$ ergibt.

Der Lauf des Prozesses stellt eine Ausführung dar und ist ein Wort. Die Monitore eines Prozesses überprüfen, ob sein Lauf verschiedene Eigenschaften erfüllt, die als Formel einer temporalen Logik spezifiziert wurden.

In einem asynchronen, verteilten System gibt es mehrere verteilte Akteure, die asynchron miteinander kommunizieren. Im Folgenden wird jeder Akteur in einem solchen System als ein Prozess aufgefasst. Jeder Prozess wird von mehreren Monitoren überwacht und Prozesse können sich untereinander Nachrichten schicken, damit die Monitore an die Informationen über das Verhalten anderer Prozesse kommen.

In den folgenden Abschnitten wird beschrieben, wie man mit einer Logik eine Formel für einen Monitor in einem asynchronen, verteilten System spezifizieren kann.

2.5.4 Past-Time Distributed Temporal Logic

Die past-time Distributed Temporal Logic (ptDTL) wurde in [SVAR04] definiert. Sie dient als eine Logik für asynchrone, verteilte Systeme, bei der jede Formel einem Monitor eines Prozesses zugeordnet ist, Teilformeln der Formel aber mit Hilfe des @-Operators auf Formeln von Monitoren anderer Prozesse verweisen können und dadurch Platzhalter für deren Wahrheitswerte sind. Dadurch können von einem Monitor sowohl Bedingungen über das Verhalten des ihm zugeordneten Prozesses, als auch über sein Wissen über das Verhalten anderer Prozesse spezifiziert werden. Die Teilformeln, welche von einem @ umschlossen sind und auf einen anderen Prozess verweisen, werden von einem Monitor des Prozesses ausgewertet, auf den das @ verweist und im Folgenden

als entfernte Formeln bezeichnet. Das Ergebnis wird dann an den Prozess des Monitors, zu dem die eigentliche Formel gehört, geschickt. Der Monitor wertet mit Hilfe der Ergebnisse der anderen Monitore seine Formel aus und kann dann eventuell Fehler oder Erfolge erkennen.

Zuerst werden die Syntax und die Semantik von ptDTL analog zu [SVAR04] definiert. Danach wird eine genauere Beschreibung des @-Operators vorgenommen und darauf folgt ein Beispiel. Als letztes werden die Nachteile von ptDTL betrachtet.

Definition 2.15 (ptDTL-Syntax)**.** *Sei $a \in AP^r$ eine atomare Proposition des Prozesses r, auf den das innerste @ verweist, das a umschließt. Seien weiter p und q Prozesse. Die Menge der ptDTL-Formeln ist induktiv über die folgende Grammatik definiert:*

$$\psi ::= @_p \varphi$$

$$\begin{array}{rccccccccccc} \varphi ::= & \mathit{true} & | & a & | & \varphi \vee \varphi & | & \ominus \varphi & | & \varphi \mathcal{S} \varphi & | & \diamondsuit \varphi & | \\ & \mathit{false} & | & \neg a & | & \varphi \wedge \varphi & & & | & \varphi \mathcal{B} \varphi & | & \boxminus \varphi & | \\ & \neg \varphi & | & @_q \varphi & & & & & & & & & \end{array}$$

Die Syntax wurde nur um den @-Operator erweitert und entspricht sonst der von ptLTL. Das ψ existiert, damit es auf jeden Fall ein @ um die gesamte Formel gibt, damit jede Formel eindeutig einem Prozess zugeordnet werden kann.

Definition 2.16 (ptDTL-Semantik)**.** *Seien p und q Prozesse, AP^p eine endliche Menge von atomaren Propositionen, $a \in AP^p$ eine atomare Proposition, $\Sigma^p = 2^{AP^p}$ das Alphabet von Prozess p, $w^p \in \Sigma^{p+}$ der Lauf des Prozesses p und C die Menge der möglichen Konfigurationen des Systems, deren Elemente angeben, welches Wissen ein Prozess über die anderen Prozesse besitzt. Sei weiter $i \in \mathbb{N}$ mit $0 \leq i < |w^p|$. Die Semantik von ptDTL ist für eine Formel über den Prozess p durch die Auswertungsfunktionen $[\![\cdot]\!]_{ptDTL} : C \times \Sigma^{p+} \times \mathbb{N} \times ptDTL \to \mathbb{B}_2$ wie folgt definiert:*

$$[\![\mathcal{C}, w^p, i \models @_p \varphi]\!]_{ptDTL} = \begin{cases} \top & \text{wenn } \mathcal{C}, w^p, i \models \varphi \\ \bot & \text{sonst} \end{cases}.$$

Dabei ist die Relation $\models \subseteq C \times \Sigma^{p+} \times \mathbb{N} \times ptDTL$ induktiv wie folgt definiert:

$$\begin{array}{ll} \mathcal{C}, w^p, i \models \mathit{true} & \text{für alle } \mathcal{C}, w^p, i \\ \mathcal{C}, w^p, i \models a & \text{gdw. } a \in w^p_i \\ \mathcal{C}, w^p, i \models \neg \varphi & \text{gdw. } \mathcal{C}, w^p, i \not\models \varphi \\ \mathcal{C}, w^p, i \models \varphi \vee \psi & \text{gdw. } \mathcal{C}, w^p, i \models \varphi \text{ oder } \mathcal{C}, w^p, i \models \psi \\ \mathcal{C}, w^p, i \models \ominus \varphi & \text{gdw. } \mathcal{C}, w^p, i-1 \models \varphi \vee i = 0 \wedge \mathcal{C}, w^p, i \models \varphi \\ \mathcal{C}, w^p, i \models \varphi \mathcal{S} \psi & \text{gdw. } \exists j \leq i : \mathcal{C}, w^p, j \models \psi \wedge \forall j < k \leq i : \mathcal{C}, w^p, k \models \varphi \\ \mathcal{C}, w^p, i \models @_q \varphi & \text{gdw. } p \text{ weiß, dass } \varphi \text{ auf Prozess } q \text{ gilt} \end{array}$$

Für die Operatoren sowie false mit Ausnahme des Weak-Previous-Operators, weil dieser in ptDTL nicht existiert, gelten dieselben Äquivalenzen wie in ptLTL.

Zusätzlich zu dem @-Operator gibt es noch eine weitere Änderung in der Semantik im Vergleich zu ptLTL. Es ist noch das $\mathcal{C}$ hinzugekommen, welches jeweils die aktuelle Konfiguration des Systems ist und damit angibt, was ein Prozess über die anderen Prozesse weiß.

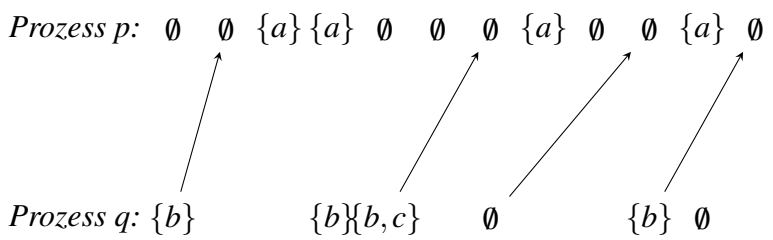
Wie zuvor beschrieben werden über den @-Operator Teilformeln über andere Prozesse spezifiziert. Der äußerste @-Operator gibt dabei an, auf welchen Prozess sich die Gesamtformel bezieht. Sie wird dann von einem, zu dem Prozess zugehörigen, Monitor ausgewertet. Wird beispielsweise eine Formel $@_p\varphi$ betrachtet, so wird sie von einem Monitor des Prozesses p überwacht.

In ptDTL sind weder $\top$ noch $\bot$ impartial, denn es kann in einer Formel $\boxminus a$ mit einer Proposition a von $\top$ nach $\bot$ gewechselt werden, wenn das Wort verlängert wird und in einer Formel $\diamondminus a$ von $\bot$ nach $\top$. Nach [FFM12] sind deshalb mit ptDTL keine Eigenschaften monitorbar. Um nun zu erreichen, dass mit ptDTL Eigenschaften monitorbar sind, muss einer der beiden Wahrheitswerte impartial werden. Dies ist möglich, indem um jede ptDTL-Formel entweder ein Globally-Operator ($\Box$) oder ein Finally-Operator ($\diamondsuit$) eingefügt wird, wobei deren Semantiken wie bei LTL sind. Mit einem Globally-Operator ist $\bot$ impartial und damit Safety-Formeln monitorbar und mit einem Finally-Operator ist $\top$ impartial und damit Guarantee-Formeln monitorbar. Aus diesem Grund wird im Folgenden immer entweder ein Globally- oder ein Finally-Operator um eine ptDTL-Formel eingefügt.

Eine entfernte Formel in der Gesamtformel ist nur ein Platzhalter für den Wahrheitswert dieser Teilformel, die sich auf einen entfernten Prozess bezieht und auf einem von dessen Monitoren ausgewertet wird. Wird eine Teilformel von einem $@_q$ umschlossen, so wird diese von dem Monitor des Prozesses q ausgewertet. Sei $@_p\varphi$ eine Formel über Prozess p und $@_q\psi$ eine Teilformel von φ und eine entfernte Formel auf Prozess q. Ein Monitor m_q des Prozesses q würde dauerhaft die Formel $@_q\psi$ auswerten und das Ergebnis an den Prozess p übermitteln, damit der Monitor m_p, der φ auswertet, die benötigten Informationen erhält. m_p findet dann mit Hilfe der Informationen von m_q heraus, ob $@_q\psi$ gilt und kann dadurch $@_p\varphi$ auswerten.

Beispiel 2.17 (ptDTL)**.** *Seien p und q Prozesse, m_p ein Monitor von Prozess p und $m_{q,1}$ und $m_{q,2}$ zwei Monitore von Prozess q, $AP^p = \{a\}$ und $AP^q = \{b,c\}$ die Mengen der möglichen Propositionen der beiden Prozesse und $\Sigma^p = 2^{AP^p}$ und $\Sigma = 2^{AP^q}$ die zugehörigen Alphabete. $\varphi = @_p\Box(a \rightarrow @_q\boxminus b)$ und $\psi = @_q\Box(\diamondminus c)$ seien die zu betrachtenden Formeln. Von m_p wird die Formel φ überwacht, da dies der Monitor für Prozess p ist, von $m_{q,1}$ wird die Formel ψ überwacht und die Formel $@_q\boxminus b$ wird von $m_{q,2}$ ausgewertet, da diese von m_p benötigt wird.*

Die beiden Prozesse haben nun folgenden Lauf:



Man sieht, dass in diesem Beispiel zwei Prozesse gewählt wurden, die unterschiedlich oft und schnell Schritte durchführen. Prozess p nimmt gleichmäßig neue Zustände an, insgesamt zwölf Stück. q hingegen ist unbeständiger und nimmt gerade mal sechs Zustände in derselben Zeit an.

Zu prüfen, ob ψ erfüllt ist, ist vergleichsweise einfach, da ψ lokal auf Monitor $m_{q,1}$ ausgewertet werden kann. $\diamondminus c$ sagt aus, dass irgendwann in der Vergangenheit mal ein c aufgetreten sein muss. Durch den Globally-Operator außen herum wird dies in jedem Schritt geprüft. Deshalb muss in jedem Schritt gelten, dass irgendwann in der Vergangenheit mal ein c aufgetreten ist. Im ersten Schritt von q ergibt sich für ψ zu $\bot$. Wegen dem Globally-Operator gilt daher auch schon insgesamt $\bot$. Ohne den Globally-Operator würde die Formel ab dem dritten Schritt zu $\top$ ausgewertet werden, da ab diesem Schritt immer in der Vergangenheit ein c vorkommt und es würde nicht auffallen, dass vorher schon ein Fehler passiert war.

Um φ auszuwerten wird zunächst die Formel $@_q \boxminus b$ auf Prozess q betrachtet. Diese Formel wird bis zu dem dritten Schritt, $\{b, c\}$, zu $\top$ und danach zu $\bot$ ausgewertet. Die Pfeile sollen darstellen, wann in diesem Beispiel Wissen über die aktuellen Wahrheitswerte der Formel von Prozess q an Prozess p gesandt wird. Da es sich um ein asynchrones System handelt können diese Informationen beliebig verschickt und bei der Übertragung verzögert werden. Aus diesem Grund kommen die Pfeile in dem Beispiel teilweise erst verspätet bei Prozess p an. Das erste Mal, dass $@_q \boxminus b$ für m_p wichtig ist, ist in dem dritten Schritt von Prozess p, da dort ein a auftritt. Da vorher Informationen von $m_{q,2}$ ankommen, die enthalten, dass $@_q \boxminus b$ dort $\top$ ergab, wird auch φ zu $\top$ ausgewertet. Dies ändert sich nicht im vierten Schritt. Ein interessanter Fall tritt bei dem nächsten a auf, in Schritt 8. Wie man sieht galt vorher auf Monitor $m_{q,2}$ $@_q \boxminus b$ schon nicht mehr, allerdings ist das Wissen darüber noch nicht bei m_p angekommen. Daher ergibt sich für φ trotzdem noch $\top$. In Schritt 10 erfährt Monitor m_p dann, dass $@_q \boxminus b$ nicht mehr gilt. Aus diesem Grund wird φ in Schritt 11 auch zu $\bot$ ausgewertet sowie durch den Globally-Operator auch in allen folgenden Schritten.

Das am meisten auffallende Problem in der Definition von ptDTL ist die ungenaue Formulierung des @-Operators aus [SVAR04]. Das @ wurde in [SVAR04] nur als Wissen eines Monitors eines Prozesses über die Wahrheitswerte der Formeln auf dem Monitor eines anderen Prozesses definiert. Es ist unklar, wann dieses Wissen entsteht und daher unmöglich, eine ptDTL-Formel exakt auszuwerten. Es könnte zum Beispiel sein, dass auf Prozess p irgendwann ein a gilt und auf Prozess q vorher mal nicht b gegolten hat. Nach der Definition von ptDTL ist nicht klar, ob $@_p \Box(a \rightarrow @_q \boxminus b)$ erfüllt ist, da unklar ist, wann bei dem Monitor von Prozess p ankommt, dass bei dem Monitor von q die Formel $@_q \boxminus b$ verletzt ist. Dadurch ist eine exakte Definition der Semantik nicht möglich. Dies soll im nächsten Abschnitt durch einführen eines Nachrichtenmodells für ptDTL zuerst einmal leicht verbessert werden.

Ein weiteres, wichtiges Problem in ptDTL ist, dass es kein Modell für eine gesamte Ausführung des Systems gibt. Das einzig Bekannte sind die Läufe der Prozesse. Allerdings ist völlig unklar, wann die Prozesse im Verhältnis zu den anderen Schritte gemacht haben. Dadurch ist es unmöglich genau zu sagen, welches Wissen eine entfernte Formel zu einem Zeitpunkt beinhaltet, da es mehrere Möglichkeiten für die Reihenfolge der Schritte der Prozesse gibt und dadurch auch mehrere Möglichkeiten für das Wissen, welches für eine entfernte Formel benötigt wird. Aus diesem Grund kann mit ptDTL im Allgemeinen kein eindeutiger Wahrheitswert bestimmt werden.

Obwohl ptDTL nur eine zweiwertige Semantik hat, ist die Logik dennoch in einem begrenzten Bereich impartial. In der Definition der Logik ist dies nicht der Fall, abhängig von dem Zukunfts-Operator, der um eine ptDTL-Formel außen herum eingefügt wird, allerdings schon, wie vorher bereits erwähnt. Dies ist aber nur bei der Gesamtformel der Fall und nicht bei den Teilformeln, die von Monitoren von anderen Prozessen ausgewertet werden, da diese nicht von einem Zukunfts-Operator umgeben sind. Als Gesamtformeln sind dadurch entweder Safety- oder Guarantee-Eigenschaften darstell- und monitorbar, was auch so beabsichtigt ist, wie in [SVAR04] beschrieben. Das Problem ist, dass durch die Zweiwertigkeit auch nur Safety- oder Guarantee-Eigenschaften monitorbar sind, wie in Unterabschnitt 2.4.1 auf Seite 14 erläutert.

Anticipation ist in der ptDTL-Semantik nicht vorhanden. Es wird kein Erfüllbarkeitstest für die Formeln durchgeführt und aus diesem Grund kann die Semantik von ptDTL keine Anticipation besitzen.

2.5.5 Nachrichten

Nachrichten stellen in den folgenden Abschnitten dieser Arbeit die Kommunikation und Informationsübertragung zwischen zwei Prozessen dar. In diesem Abschnitt werden Sende- und Empfangszeitpunkte definiert und in späteren Abschnitten werden die Nachrichten noch erweitert.

Sende- und Empfangszeitpunkte werden genutzt, um in den Läufen der Prozesse anzugeben, wann die Prozesse Nachrichten versandt haben und wann die Nachrichten von einem anderen Prozess empfangen wurden.

Definition 2.18 (Nachricht)**.** *Seien p und q Prozesse aus der Menge P aller Prozesse, m_p und m_q zwei beliebige, zu den entsprechenden Prozessen zugehörige, Monitore und TL die temporale Logik, die verwendet wird. Eine Nachricht von p nach q ist eine Funktion $TL \to \mathbb{B}_x$, wobei $\mathbb{B}_x$ der Verband, über den die Semantik von TL definiert ist. Die Funktion bildet von einer Formel eines Monitors m_p, die von einem Monitor m_q benötigt wird, auf deren Wahrheitswert bei dem Versand der Nachricht ab.*

Werden Formeln von einem Monitor m_p ausgewertet und von einem Monitor m_q benötigt, so werden immer mal wieder Nachrichten von Prozess p an Prozess q versandt, um die aktuellen Daten zu übermitteln. Der Versandzeitpunkt wird durch die Proposition $\uparrow_i^q$ im Lauf w^p des Prozesses p bezeichnet, wobei $i \in \mathbb{N} \setminus \{0\}$ die ID der Nachricht ist und angibt, dass dies die i-te Nachricht von p an q ist. Der Empfangszeitpunkt der i-ten Nachricht auf Prozess q von Prozess p wird in w^q, dem Lauf von Prozess q, mit der Proposition $\downarrow_i^p$ bezeichnet.

Für eine Formel $\varphi \in TL$, die von m_p benötigt und von m_q ausgewertet wird, einem $b \in \mathbb{B}_x$ und $\uparrow_i^p \in w_k^q$ gilt:

$$\uparrow_i^p (\varphi) = b \wedge \downarrow_i^q (\varphi) = b \wedge [\![w_{0..k}^q \models \varphi]\!]_{TL} = b$$

Eine Nachricht ist eine Funktion, welche von einer Formel auf den aktuellen Wahrheitswert der Formel bei dem Versand der Nachricht abbildet. Dabei ist jede Nachricht von einem Prozess p zu einem Prozess q eindeutig durch eine ID identifizierbar. Wie vorher beschrieben werden Sende-

und Empfangszeitpunkte in die Läufe der Prozesse eingefügt, um zu symbolisieren, wann Nachrichten versandt und empfangen wurden. Dabei werden diese Zeitpunkte an der Stelle des Laufes, an der sie auftreten, in die Menge eingefügt.

2.5.6 Nachrichtenbasierte Semantik von ptDTL

Mit Hilfe der vorher definierten Nachrichten wird jetzt eine auf Nachrichten basierende Semantik von ptDTL entwickelt, ptDTL$_N$, die auf dem Versand und Empfang dieser Nachrichten aufbaut. Das Auftreten der Nachrichten wurde dazu in den Läufen der Prozesse vermerkt, wie vorher beschrieben. Durch die Nachrichten ist dann anhand der Läufe der Prozesse eindeutig klar, ob die Formel bei einer Ausführung erfüllt ist oder nicht. Deshalb wird die Konfiguration $\mathcal{C}$, welche bei ptDTL benutzt wurde, nicht mehr benötigt.

Definition 2.19 (ptDTL$_N$-Semantik)**.** *Es wird dieselbe Notation verwendet, wie in Definition 2.16 auf Seite 19. Für die Auswertungsfunktionen* $[\![\cdot]\!]_{ptDTL_N} : {\Sigma^p}^+ \times \mathbb{N} \times ptDTL \to \mathbb{B}_2$ *gilt*

$$[\![w^p, i \models @_p \varphi]\!]_{ptDTL_N} = \begin{cases} \top & \text{wenn } w^p, i \models \varphi \\ \bot & \text{sonst} \end{cases}.$$

Dabei ist die Relation $\models \subseteq {\Sigma^p}^+ \times \mathbb{N} \times ptDTL$ *induktiv wie folgt definiert:*

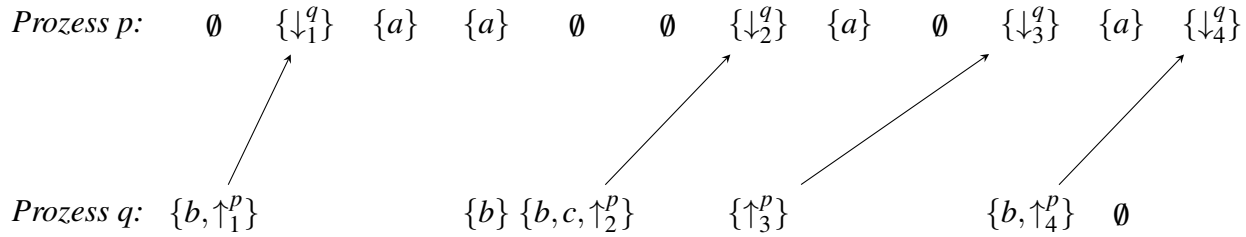
$$\begin{array}{ll}
w^p, i \models true & \text{für alle } w^p, i \\
w^p, i \models a & \text{gdw. } a \in w^p_i \\
w^p, i \models \neg\varphi & \text{gdw. } w^p, i \not\models \varphi \\
w^p, i \models \varphi \vee \psi & \text{gdw. } w^p, i \models \varphi \text{ oder } w^p, i \models \psi \\
w^p, i \models \ominus\varphi & \text{gdw. } w^p, i-1 \models \varphi \vee i = 0 \wedge w^p, i \models \varphi \\
w^p, i \models \varphi \mathcal{S} \psi & \text{gdw. } \exists j \leq i : w^p, j \models \psi \wedge \forall j < k \leq i : w^p, k \models \varphi \\
w^p, i \models @_q\varphi & \text{gdw. } \exists j \leq i : \downarrow^q_k \in w^p_j \wedge \forall l > k : \nexists h \leq i : \downarrow^q_l \in w^p_h \wedge \downarrow^q_k (\varphi) = \top
\end{array}$$

Es gelten dieselben Äquivalenzen wie in ptDTL.

Neben dem Wegfallen der Konfiguration ist die einzige Änderung an der Definition von ptDTL für ptDTL$_N$ die Definition des @-Operators, welcher nun mit Hilfe von Nachrichten spezifiziert wurde. Das heißt, dass für einen Monitor von Prozess p eine, auf einem Monitor von Prozess q ausgewertete, Formel nur erfüllt ist, wenn es auf Prozess p eingehende Nachrichten von Prozess q gab und beim Versenden der neusten dieser Nachrichten auf Prozess q die Formel erfüllt war.

Durch die Nachrichten gibt es nun ein eindeutiges Modell für den Nachrichtenaustausch, im Gegensatz zu der Definition des @-Operators aus der ursprünglichen Definition der Semantik in Definition 2.16 auf Seite 19. Dadurch kann eindeutig festgestellt werden, ob eine Ausführung der Prozesse die Formel erfüllt oder nicht.

Beispiel 2.20 (ptDTL$_N$)**.** *Die beiden Prozesse aus dem Beispiel 2.17 auf Seite 20 haben mit der neuen Definition nun folgende Läufe:*



Die Auswertung der Formeln funktioniert wie in Beispiel 2.17 auf Seite 20, außer dass man nun anhand der Läufe sehen kann, wann Nachrichten versandt werden und ankommen. Außerdem werden die Wahrheitswerte der entfernen Formeln nun aus den Nachrichten abgefragt. Dadurch kann allein durch das Auswerten der Läufe festgestellt werden, ob die Formeln erfüllt sind.

ptDTL$_N$ behebt das Problem der ungenauen Definition des @-Operators in ptDTL. Allerdings gibt es noch kein theoretisches Modell, welches die Erstellung der Läufe der Prozesse und die Entstehung der Nachrichten beschreibt. Nachrichten können nicht beliebig in den Läufen auftreten, denn es darf nicht vorkommen, dass eine Nachricht ankommt, bevor sie versandt wurde. Damit Nachrichten in den Läufen korrekt vorhanden sein können wird allerdings das Wissen über die Abfolge der Schritte der Prozesse im Gesamtsystem benötigt. Ein Modell, welches diese Anforderungen erfüllt, wird im nächsten Kapitel entwickelt.

Die weiteren Nachteile, wie die fehlenden Möglichkeiten zur Synchronisation oder die sehr beschränkte Menge an monitorbaren Eigenschaften, die aus der zweiwertigen Semantik folgt, können durch ptDTL$_N$ auch nicht behoben werden. Diese Nachteile werden auch im nächsten Kapitel durch die Definition einer neuen Logik verbessert.

Die Modellierung des Nachrichtenaustausches wird in den folgenden Kapiteln weiter verwendet und noch erweitert.

3 Future Synchronized Distributed Temporal Logic

In diesem Kapitel wird die future Synchronized Distributed Temporal Logic (fSDTL) definiert. Sie ist eine Abwandlung und Erweiterung von ptDTL$_N$, welche die Nachteile, die bei ptDTL beschrieben wurden, entfernt.

Zuerst werden noch einmal die Anforderungen an fSDTL zusammengefasst, welche in dem vorherigen Kapitel, insbesondere bei der Definition von ptDTL, als Eigenschaften oder Probleme beschrieben wurden. Danach wird zuerst die future Distributed Temporal Logic (fDTL) definiert, welche ptDTL mit Zukunfts-Operatoren anstelle von Vergangenheits-Operatoren entspricht und statt einer zweiwertigen eine dreiwertige Semantik hat. Diese behebt einige Probleme, die es bei ptDTL gab. Das fehlende, grundlegende Modell wird durch die Definition der Prozesse als Transitionssysteme sowie die Definition von Schedulern erstellt werden. Das Problem der Asynchronität soll später durch Hinzunahme von Synchronisationsaktionen verbessert werden. Die dabei entstehende Logik wird dann fSDTL genannt. Nach der Definition von fSDTL wird die Logik analysiert, um zu untersuchen, welche Eigenschaften die Logik besitzt.

3.1 Anforderungen

Die Anforderungen werden in diesem Abschnitt noch einmal kurz aufgeführt und erläutert.

Die ungenaue Definition des @-Operators wurde schon durch ptDTL$_N$ verbessert, es fehlt aber noch ein Modell für eine Ausführung des Systems. Dieses soll jeweils eine mögliche zeitliche Abfolge der Schritte der Prozesse darstellen, sowie die Kommunikation zwischen diesen modellieren, damit auf Basis dieses Modells für eine Ausführung des Systems ein eindeutiger Wahrheitswert bestimmt werden kann.

fSDTL sollte auch sowohl eine dreiwertige Semantik als auch Impartiality besitzen. Wie in Unterabschnitt 2.4.1 auf Seite 14 beschrieben, sind damit deutlich mehr Eigenschaften monitorbar als mit einer zweiwertigen Semantik mit Impartiality, wie es bei ptDTL durch das Hinzufügen eines Globally- oder Finally-Operators der Fall ist. Des Weiteren sind durch das Hinzufügen eines Globally- oder Finally-Operators um eine ptDTL-Formel auch nur noch Safety- oder Guarantee-Eigenschaften spezifizerbar. Eine Anforderung an fSDTL wäre also auch noch eine grundsätzliche Erhöhung der Mächtigkeit.

Die letzte wichtige Anforderung an fSDTL ist der Besitz von Anticipation, um endgültige Wahrheitswerte möglichst früh zu erkennen. Anticipation war in ptDTL nicht vorhanden, ist aber eine wünschenswerte Eigenschaft einer Logik, die für Laufzeitverifikation genutzt werden soll.

3.2 Future Distributed Temporal Logic

Als Zwischenschritt zur Definition von fSDTL wird die Future Distributed Temporal Logic (fDTL) definiert. Diese soll bereits mehrere Eigenschaften von ptDTL verbessern, wird aber nicht alles abdecken, wie später beschrieben.

Zum einen soll in fDTL die Grundidee von ptDTL beibehalten werden, nämlich entfernte Formeln mit Hilfe des @-Operators anzugeben. Allerdings soll die Logik nun auf Zukunfts-Operatoren basieren und eine dreiwertige Semantik mit Impartiality und Anticipation besitzen, um die Menge der monitorbaren Formeln zu vergrößern. Eine nicht-verteilte Logik, die diese Eigenschaften besitzt, ist LTL_3. Aus diesem Grund bietet es sich an, fDTL auf LTL_3 basieren zu lassen, um die gewünschten Eigenschaften zu erreichen.

Des Weiteren sollen die Werte des @-Operators möglichst unabhängig vom Versand- und Empfangszeitpunkt der Nachrichten sein, damit dies auch für den Wert der Gesamtformel gilt. Das heißt, die Werte der Nachrichten sollen zum Einen nicht abhängig vom Versandzeitpunkt zwischen verschiedenen Wahrheitswerten schwanken, sondern, wie die Gesamtformel auch, sich irgendwann festlegen. Deshalb müssen die Werte, die per Nachricht versandt werden, impartial sein. Um dies zu erreichen müssen auch die entfernten Formeln mit derselben Semantik wie die eigentliche Formel ausgewertet werden. Zum Anderen darf der @-Operator sich aber zu keinem Zeitpunkt mit einem ? zufrieden geben, denn das Ziel ist es herauszufinden, ob auf dem anderen Prozess etwas gilt oder nicht. Aus diesem Grund muss von dem @-Operator das gesamte Wort, unabhängig von der aktuell betrachteten Stelle des Wortes, betrachtet werden, um eine Nachricht mit einem Wahrheitswert einer entfernten Formel zu finden, der impartial ist, wenn es bereits einen solchen gibt. Durch die fehlenden Synchronisationsaktionen hat dies in fDTL einige Nachteile, wie anhand entsprechender Beispiele später gezeigt wird, da jede entfernte Formel immer vom Beginn des Laufes des entfernten Prozesses an ausgewertet wird.

Im Folgenden werden zuerst Syntax und Semantik von fDTL definiert sowie deren Vorteile und Nachteile erläutert. In den darauf folgenden Abschnitten werden Scheduler und Synchronisationsaktionen in die Logik eingebaut sowie ein Modell für das System erstellt, um die verbliebenen Probleme aus ptDTL zu entfernen.

3.2.1 fDTL-Syntax

Definition 3.1 (fDTL-Syntax)**.** *Sei $a \in AP^r$ eine atomare Proposition des Prozesses r, auf den das innerste @ verweist, das a umschließt. Seien weiter p und q Prozesse. Die Menge der fDTL-Formeln ist induktiv über die folgende Grammatik definiert:*

$$\psi ::= @_p \varphi$$

$$\begin{array}{rccccccccccc} \varphi ::= & \mathit{true} & | & a & | & \varphi \vee \varphi & | & \bigcirc \varphi & | & \varphi \mathcal{U} \varphi & | & \Diamond \varphi & | \\ & \mathit{false} & | & \neg a & | & \varphi \wedge \varphi & | & & & \varphi \mathcal{R} \varphi & | & \Box \varphi & | \\ & \neg \varphi & | & @_q \varphi & & & & & & & & & \end{array}$$

Die Syntax wurde nur um den @-Operator erweitert und entspricht sonst der von LTL. Wie bei ptDTL ist das ψ dafür da, um ein @ um die gesamte Formel zu garantieren, damit jede Formel eindeutig einem Prozess zugeordnet werden kann.

3.2.2 fDTL-Semantik

Die fDTL-Semantik basiert auf der dreiwertigen Semantik von LTL über endlichen Präfixen von unendlichen Worten, LTL_3.

Definition 3.2 (fDTL-Semantik)**.** *Seien p und q Prozesse, AP^p eine Menge von atomaren Propositionen auf dem Prozess p und $a \in AP^p$ eine atomare Proposition sowie $N^p \subseteq AP^p$ die Menge der Propositionen für die Sende- und Empfangszeitpunkte der Nachrichten von Prozess p. Weiter sei $\Sigma^p = 2^{AP^p}$ das Alphabet von p, $\xi^p = 2^{AP^p \setminus N^p}$ das Alphabet ohne Nachrichten, $w^p \in \Sigma^{p+}$ ein endlicher Präfix des Laufes und $i \in \mathbb{N}$. Die Semantik von fDTL ist für eine Formel über den Prozess p durch die Auswertungsfunktion $[\![\cdot]\!]_{fDTL} : \Sigma^{p+} \times fDTL \to \mathbb{B}_3$ wie folgt definiert:*

$$[\![w^p \models @_p \varphi]\!]_{fDTL} = \begin{cases} \top & \text{wenn } \forall u \in \xi^\omega : [\![w^p u, 0 \models \varphi]\!]_{fDTL_\omega} = \top \\ \bot & \text{wenn } \forall u \in \xi^\omega : [\![w^p u, 0 \models \varphi]\!]_{fDTL_\omega} = \bot \\ ? & \text{sonst} \end{cases}$$

wobei $[\![\cdot]\!]_{fDTL_\omega} : {\Sigma^p}^\omega \times \mathbb{N} \times fDTL \to \mathbb{B}_3$ wie folgt definiert ist:

$$[\![w,i \models true]\!]_{fDTL_\omega} = \top$$

$$[\![w,i \models a]\!]_{fDTL_\omega} = \begin{cases} \top & wenn\ a \in w_i \\ \bot & sonst \end{cases}$$

$$[\![w,i \models \neg\varphi]\!]_{fDTL_\omega} = \overline{[\![w,i \models \varphi]\!]_{fDTL_\omega}}$$

$$[\![w,i \models \varphi \vee \psi]\!]_{fDTL_\omega} = [\![w,i \models \varphi]\!]_{fDTL_\omega} \sqcup [\![w,i \models \psi]\!]_{fDTL_\omega}$$

$$[\![w,i \models \bigcirc\varphi]\!]_{fDTL_\omega} = \begin{cases} \top & wenn\ [\![w,i+1 \models \varphi]\!]_{fDTL_\omega} = \top \\ \bot & wenn\ [\![w,i+1 \models \varphi]\!]_{fDTL_\omega} = \bot \\ ? & sonst \end{cases}$$

$$[\![w,i \models \varphi\,\mathcal{U}\,\psi]\!]_{fDTL_\omega} = \begin{cases} \top & wenn\ \exists j \geq i : [\![w,j \models \psi]\!]_{fDTL_\omega} = \top \wedge \\ & \quad \forall i \leq h < j : [\![w,h \models \varphi]\!]_{fDTL_\omega} = \top \\ \bot & wenn\ \forall j \geq i : [\![w,j \models \psi]\!]_{fDTL_\omega} = \bot \vee \\ & \quad \exists i \leq h < j : [\![w,h \models \varphi]\!]_{fDTL_\omega} = \bot \\ ? & sonst \end{cases}$$

$$[\![w,i \models @_q\varphi]\!]_{fDTL_\omega} = \begin{cases} \downarrow_k^q (@_q\varphi) & wenn\ \exists j \geq 0 : \downarrow_k^q \in w_j \wedge \downarrow_k^q (@_q\varphi) \neq ? \\ ? & sonst \end{cases}$$

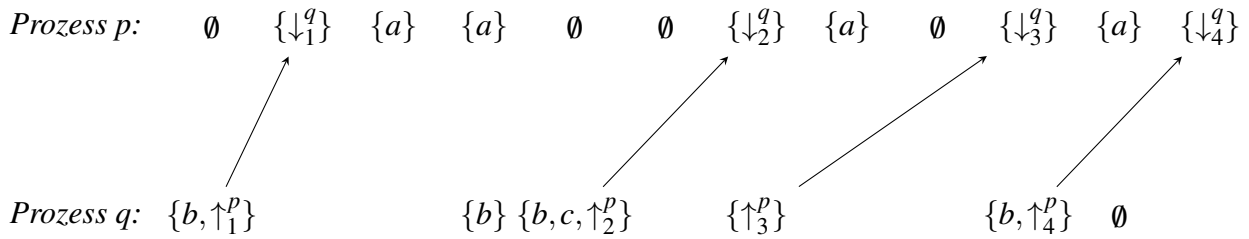
Es gelten dieselben Äquivalenzen wie in LTL.

Die fDTL-Semantik wurde auf andere Weise definiert als LTL_3. Insbesondere unterscheidet sich die zugrunde liegende Logik. Der Grund dafür ist der neu hinzugekommene @-Operator. Da beim Nutzen der dreiwertigen Semantik auch die entfernten Formeln auf dem entfernten Prozessen über die dreiwertige Semantik ausgewertet werden, stehen auch in den Nachrichten für eine Formel drei mögliche Werte. Würde man fDTL nun basierend auf einer zweiwertigen Semantik definieren, wie LTL_3, so müsste die zweiwertige Semantik die Wahrheitswerte aus den Nachrichten verarbeiten, ohne dabei Informationen zu verlieren. Dies ist aber nicht möglich, da man immer Informationen verliert, wenn man von drei möglichen Werten auf zwei reduziert. Aus diesem Grund wurde direkt eine dreiwertige Semantik angegeben.

Des Weiteren wurden auch die unendlichen Verlängerungen über einem anderen Alphabet definiert als der endliche Präfix selber, anders als bei LTL_3. Der Grund dafür ist, dass die unendlichen Verlängerungen keine Nachrichten enthalten dürfen, da sonst die Semantik Wahrheitswerte für entfernte Formeln herausfinden könnte, für die eigentlich noch keine Nachrichten angekommen sind. Ein Beispiel dafür wäre die Formel $@_p(\Diamond a \vee @_q\text{true})$. Würde es in der unendlichen Verlängerung des endlichen Präfixes Nachrichten geben, so würde in all diesen Nachrichten für $@_q$true der Wert $\top$ stehen. Dadurch würde sich für die Formel immer $\top$ ergeben, obwohl in dem endlichen Präfix eventuell noch keine Nachricht empfangen wurde.

Beispiel 3.3 (fDTL)**.** *Es wird ein ähnliches Beispiel wie das Beispiel 2.17 auf Seite 20 betrachtet und dieselben Prozesse und Läufe der Prozesse genutzt. In diesem Beispiel werden allerdings fDTL-Formeln anstelle von ptDTL-Formeln genutzt. Die Formeln sind grundsätzlich dieselben, außer, dass alle Vergangenheits-Operatoren durch Zukunfts-Operatoren ersetzt wurden, wodurch sich aber eine andere Semantik ergibt.*

Sei $\varphi = @_p\Box(a \rightarrow @_q\Box b)$ die auf dem Monitor m_p von Prozess p und $\psi = @_q(\Diamond c)$ die auf dem Monitor $m_{q,1}$ von Prozess q zu überwachende Formel. Die entfernte Formel $@_q\Box b$, die eine Teilformel von ψ ist, wird von dem Monitor $m_{q,2}$ ausgewertet. Der aus Beispiel 2.17 auf Seite 20 bekannte Lauf der beiden Prozesse wird hier noch einmal zur Übersicht dargestellt:



Zu prüfen, ob ψ erfüllt ist, ist wieder einfach. Die Formel sagt aus, dass irgendwann mal ein c auftreten muss. Das heißt, es muss einen Punkt in dem Lauf geben, ab dem dies gilt. Ist ein c aufgetreten, so wird die Formel für den Rest des Laufes zu $\top$ ausgewertet. Für die ersten beiden Schritte von q wertet $m_{q,1}$ die Formel ψ zu ? aus, da noch kein c gesehen wurde. Ab dem dritten Schritt ergibt sich dann $\top$, da dort ein c auftritt und es daher ab diesem Zeitpunkt immer einen Zeitpunkt gibt, in dem ein c auftrat.

Für φ wird wieder zunächst die Formel $@_q\Box b$ auf $m_{q,2}$ betrachtet. Diese Formel wird bis zu dem dritten Schritt, $\{b, c, \uparrow_2^p\}$, zu ? ausgewertet, da bisher für alle Schritte b galt. Danach ergibt sich für die Formel $\bot$, denn im vierten Schritt tritt kein b auf und daher gab es ab diesem Zeitpunkt immer ein Schritt, in dem b nicht galt. φ wird in den ersten 9 Schritten von Prozess p zu ? ausgewertet, da Monitor m_p wieder vor Schritt 8 nicht mitbekommt, dass $@_q\Box b$ auf Monitor $m_{q,2}$ zu $\bot$ ausgewertet wird. Dies ist dieselbe, durch Verzögerung der Nachrichten entstehende, Ungenauigkeit wie in Beispiel 2.17 auf Seite 20. In Schritt 10 erfährt Monitor m_p dann aufgrund des Empfangszeitpunktes $\downarrow_3^q$, dass $@_q\Box b$ zu $\bot$ ausgewertet wurde, da $@_q\Box b$ auf $m_{q,2}$ den Wert $\bot$ ergab, als die Nachricht mit $\uparrow_3^q$ verschickt wurde. Aus diesem Grund ergibt sich für φ in Schritt 11 auch $\bot$ und Schritt 12 ändert daran nichts mehr.

Wie man an diesem Beispiel sieht, funktioniert fDTL im Grundsatz ähnlich wie ptDTL. Daraus folgt natürlich auch, dass einige der Probleme, die es bei ptDTL schon gab, auch bei fDTL auftreten. Auf diese Probleme wird später in diesem Kapitel genauer eingegangen.

3.2.3 Analyse

In diesem Abschnitt wird die Semantik von fDTL analysiert. Dabei wird zuerst eine Form, die @-disjunktive Normalform (@-DNF), angegeben, in die alle fDTL-Formeln äquivalent umgewandelt werden können. Als nächstes wird fDTL mit LTL_3 verglichen, um bestimmte Äquivalenzen

zwischen den Logiken zu zeigen. Schließlich wird die Mächtigkeit von fDTL und ptDTL verglichen.

Um zu zeigen, dass alle fDTL-Formeln in äquivalente Formeln mit einer bestimmten Form umgewandelt werden können, wird zuerst die @-DNF definiert.

Definition 3.4 (@-DNF)**.** *Eine fDTL-Formel $@_p\varphi$ ist in @-DNF, wenn*

$$\varphi = \bigvee_{i=0}^{n} \bigwedge_{j=0}^{m} \varphi_{i,j}$$

mit $n,m \in \mathbb{N}$ und $\varphi_{i,j} \in \{\psi \mid \psi \in fDTL \vee \neg\psi \in fDTL \vee \psi \in LTL\}$ gilt.

Eine Formel in @-DNF ist also eine boolesche Verknüpfung von fDTL-Formeln, Formeln, die negiert fDTL-Formeln sind und LTL-Formeln. Das heißt, die $\varphi_{i,j}$ besitzen entweder die Form $@_q\psi$ oder $\neg@_q\psi$ oder überhaupt keine Teilformeln der Form $@_q\psi$.

Theorem 3.5 (@-DNF)**.** *Für jede fDTL-Formel $@_p\varphi$ gibt es eine äquivalente Formel $@_p\varphi'$ in @-DNF.*

Beweis. Um dies zu zeigen muss eine Umwandlung von einer beliebigen fDTL-Formel in eine äquivalente fDTL-Formel in @-DNF gefunden werden. Dafür werden zuerst Äquivalenzen auf fDTL-Formeln betrachtet und für die Operatoren einzeln gezeigt, dass es eine geforderte Umwandlung gibt.

Bei der Definition des @-Operators in fDTL wird nach dem Gleichheitszeichen das i nicht mehr verwendet, sondern jede Stelle des Wortes betrachtet. Das heißt, die aktuelle Stelle des Wortes spielt keine Rolle für den Wert einer entfernten Formel. Aus diesem Grund hat es keinen Effekt auf den Wahrheitswert einer entfernten Formel, ob sie von einem temporalen Operator umgeben ist oder nicht. Dies wird im Folgenden genutzt, um entweder die entfernte Formel vor den temporalen Operator zu ziehen oder den temporale Operator ganz zu entfernen, wodurch wieder eine Formel in @-DNF entsteht. Zuerst wird das Next betrachtet:

Es gilt nun $@_q\varphi \in$ fDTL. Es folgt

$$\begin{aligned}
[\![w,i \models \bigcirc @_q\varphi]\!]_{\text{fDTL}_\omega} &= \begin{cases} \top & \text{wenn } [\![w,i+1 \models @_q\varphi]\!]_{\text{fDTL}_\omega} = \top \\ \bot & \text{wenn } [\![w,i+1 \models @_q\varphi]\!]_{\text{fDTL}_\omega} = \bot \\ ? & \text{sonst} \end{cases} \\
&= \begin{cases} \top & \text{wenn } [\![w,i \models @_q\varphi]\!]_{\text{fDTL}_\omega} = \top \\ \bot & \text{wenn } [\![w,i \models @_q\varphi]\!]_{\text{fDTL}_\omega} = \bot \\ ? & \text{sonst} \end{cases} \\
&= [\![w,i \models @_q\varphi]\!]_{\text{fDTL}_\omega}
\end{aligned}$$

Dadurch folgt, dass ein Next-Operator um eine entfernte Formel weggelassen werden kann.

Als nächstes wird gezeigt, dass eine Formel der Form $\bigcirc\psi$, wobei ψ in @-DNF ist, in eine Formel in @-DNF umgewandelt werden kann.

$$\bigcirc\psi = \bigcirc \bigvee_{i=0}^{n} \bigwedge_{j=0}^{m} \psi_{i,j} = \bigvee_{i=0}^{n} \bigwedge_{j=0}^{m} \bigcirc \psi_{i,j}$$

Da ψ in @-DNF ist, kann nun bei allen $\psi_{i,j} \in \text{fDTL}$ das $\bigcirc$ weggelassen werden, wie vorher gezeigt. Gilt $\neg\psi_{i,j} \in \text{fDTL}$ für ein $\psi_{i,j}$, so wird die Negation zuerst vor den Next-Operator gezogen und dann der Next-Operator entfernt. Dadurch entsteht wieder eine Formel in @-DNF.

Sei im Folgenden $F = \{\varphi \mid @_p\varphi \in \text{fDTL}\}$.

Als nächstes wird gezeigt, dass auch aus einem Until-Operator mit zwei Teilformeln, die in @-DNF sind, wieder eine @-DNF hergestellt werden kann. Zuerst werden zwei Eigenschaften des Until-Operators gezeigt. Seien dafür $\psi_i \in \text{fDTL} \cup \text{LTL}$, $0 \leq i \leq n, n \in \mathbb{N}$, $\varphi \in F \cup \text{fDTL}$, $@_q\psi' \in \text{fDTL}$ und $\psi = \bigwedge_{i=0}^{n} \psi_i \wedge \psi_@$ mit $\psi_@ = @_q\psi'$ oder $\psi_@ = \neg@_q\psi'$. Es gilt:

$$\begin{aligned}
[\![w,i \models \varphi \,\mathcal{U}\, \psi]\!]_{\text{fDTL}_\omega} = \top &\iff \exists j \geq i : [\![w,j \models \psi]\!]_{\text{fDTL}_\omega} = \top \wedge \forall i \leq h < j : [\![w,h \models \varphi]\!]_{\text{fDTL}_\omega} = \top \\
&\iff \exists j \geq i : [\![w,j \models \psi_@]\!]_{\text{fDTL}_\omega} = \top \wedge [\![w,j \models \textstyle\bigwedge_{i=0}^{n} \psi_i]\!]_{\text{fDTL}_\omega} = \top \wedge \\
&\quad\ \forall i \leq h < j : [\![w,h \models \varphi]\!]_{\text{fDTL}_\omega} = \top \\
&\iff [\![w,i \models \psi_@]\!]_{\text{fDTL}_\omega} = \top \wedge \exists j \geq i : [\![w,j \models \textstyle\bigwedge_{i=0}^{n} \psi_i]\!]_{\text{fDTL}_\omega} = \top \wedge \\
&\quad\ \forall i \leq h < j : [\![w,h \models \varphi]\!]_{\text{fDTL}_\omega} = \top \\
&\iff \left[\!\!\left[w,i \models \psi_@ \wedge \varphi \,\mathcal{U} \bigwedge_{i=0}^{n} \psi_i \right]\!\!\right]_{\text{fDTL}_\omega} = \top
\end{aligned}$$

Der Beweis für

$$[\![w,i \models \varphi \,\mathcal{U}\, \psi]\!]_{\text{fDTL}_\omega} = \bot \iff \left[\!\!\left[w,i \models \psi_@ \wedge \varphi \,\mathcal{U} \bigwedge_{i=0}^{n} \psi_i \right]\!\!\right]_{\text{fDTL}_\omega} = \bot$$

folgt analog.

Für die zweite Eigenschaft sei nun $\psi = \bigvee_{i=0}^{n} \psi_i \vee \psi_@$ und $\varphi \in F$. Es folgt

$$
\begin{aligned}
[\![w,i \models \varphi \mathcal{U}\, \psi]\!]_{\mathrm{fDTL}_\omega} = \top &\iff \exists j \geq i : [\![w,j \models \varphi]\!]_{\mathrm{fDTL}_\omega} = \top \wedge \forall i \leq h < j : [\![w,h \models \psi]\!]_{\mathrm{fDTL}_\omega} = \top \\
&\iff \exists j \geq i : [\![w,j \models \varphi]\!]_{\mathrm{fDTL}_\omega} = \top \wedge \forall i \leq h < j : [\![w,h \models \\
&\quad \psi_@]\!]_{\mathrm{fDTL}_\omega} = \top \vee [\![w,h \models \textstyle\bigvee_{i=0}^{n} \psi_i]\!]_{\mathrm{fDTL}_\omega} = \top \\
&\iff \exists j \geq i : [\![w,j \models \varphi]\!]_{\mathrm{fDTL}_\omega} = \top \wedge ([\![w,i \models \psi_@]\!]_{\mathrm{fDTL}_\omega} = \top \vee \\
&\quad \forall i \leq h < j : [\![w,h \models \textstyle\bigvee_{i=0}^{n} \psi_i]\!]_{\mathrm{fDTL}_\omega} = \top) \\
&\iff (\exists j \geq i : [\![w,j \models \varphi]\!]_{\mathrm{fDTL}_\omega} = \top \wedge [\![w,i \models \psi_@]\!]_{\mathrm{fDTL}_\omega} = \top) \vee \\
&\quad (\exists j \geq i : [\![w,j \models \varphi]\!]_{\mathrm{fDTL}_\omega} = \top \wedge \forall i \leq h < j : [\![w,h \models \\
&\quad \textstyle\bigvee_{i=0}^{n} \psi_i]\!]_{\mathrm{fDTL}_\omega} = \top) \\
&\iff \left[\!\!\left[w,i \models (\psi_@ \wedge (\mathrm{true}\,\mathcal{U}\, \varphi)) \vee \left(\bigvee_{i=0}^{n} \psi_i \mathcal{U}\, \varphi \right) \right]\!\!\right]_{\mathrm{fDTL}_\omega} = \top
\end{aligned}
$$

Der Beweis für

$$
[\![w,i \models \varphi \mathcal{U}\, \psi]\!]_{\mathrm{fDTL}_\omega} = \bot \iff \left[\!\!\left[w,i \models (\psi_@ \wedge (\mathrm{true}\,\mathcal{U}\, \varphi)) \vee \left(\bigvee_{i=0}^{n} \psi_i \mathcal{U}\, \varphi \right) \right]\!\!\right]_{\mathrm{fDTL}_\omega} = \bot
$$

folgt wiederum analog.

Sei im Folgenden $N = \{\varphi \mid \varphi \in \mathrm{fDTL} \vee \neg\varphi \in \mathrm{fDTL}\}$ die Menge aller fDTL- und negierten fDTL-Formeln.

Mit Hilfe der eben gezeigten Eigenschaften kann nun gezeigt werden, dass $\varphi \mathcal{U}\, \psi$ in @-DNF gebracht werden kann, wenn φ und ψ in @-DNF sind. Zuerst wird ψ betrachtet:

$$
\varphi \mathcal{U}\, \psi = \varphi \mathcal{U} \bigvee_{i=0}^{n} \bigwedge_{j=0}^{m} \psi_{i,j} = \bigvee_{i=0}^{n} \varphi \mathcal{U} \bigwedge_{j=0}^{m} \psi_{i,j}
$$

Nach der ersten gezeigten Eigenschaft für den Until-Operator können alle $\psi_{i,j} \in N$ aus $\bigwedge_{j=0}^{m} \psi_{i,j}$ vor den jeweiligen Until-Operator gezogen werden. Als nächstes wird φ betrachtet. φ wird zuerst in eine konjunktive Normalform (KNF) umgewandelt und dann folgt

$$
\varphi \mathcal{U}\, \psi = \left(\bigwedge_{i=0}^{n} \bigvee_{j=0}^{m} \varphi_{i,j} \right) \mathcal{U}\, \psi = \bigwedge_{i=0}^{n} \left(\bigvee_{j=0}^{m} \varphi_{i,j} \mathcal{U}\, \psi \right) = \bigwedge_{i=0}^{n} \left(\bigvee_{j=0}^{m} \varphi_{i,j} \mathcal{U}\, \psi \right)
$$

Nach der zweiten gezeigten Eigenschaft für den Until-Operator können alle $\varphi_{i,j} \in N$ aus $\bigvee_{j=0}^{m} \varphi_{i,j}$ vor den jeweiligen Until-Operator gezogen werden. Formt man die Formel, die man dadurch erhält, in DNF um, so ergibt sich wieder eine Formel in @-DNF.

Damit wurde gezeigt, dass aus jedem Until-Operator mit zwei Teilformeln in @-DNF eine äquivalente Formel in @-DNF erzeugt werden kann.

Sei nun $@_p\varphi$ eine beliebige fDTL-Formel. Um diese Formel in @-DNF umzuformen, werden zuerst alle booleschen Verknüpfungen in der Formel in eine DNF umgewandelt. Danach werden von innen nach außen rekursiv alle Teilformeln wie vorher beschrieben in @-DNF transformiert. Während der Transformation kann es passieren, dass eine gerade umgewandelte Teilformel ψ von einer Negation umgeben ist. Ist dies der Fall, so wird $\neg\psi$ in eine DNF transformiert, damit sich wieder eine @-DNF ergibt. Sind danach nicht alle booleschen Verknüpfungen in DNF, so werden die, für die es nötig ist, wieder in DNF umgeformt. Es ergibt sich insgesamt dann eine, zu $@_p\varphi$ äquivalente, Formel $@_p\varphi'$ in @-DNF. □

Als nächstes soll die Äquivalenz zwischen fDTL und LTL_3 für lokale fDTL-Formeln, daher Formeln der Form $@_p\varphi$ in denen es in φ keinen @-Operator gibt, gezeigt werden. Ohne einen @-Operator in φ ist φ eine LTL-Formel.

Dafür wird zuerst gezeigt, dass sich in fDTL_ω für eine Formel φ kein ? ergeben kann, wenn es in φ keinen @-Operator gibt.

Lemma 3.6 (fDTL_ω ohne @-Operator)**.** *Für eine Formel φ ohne @-Operator kann sich mit der fDTL_ω-Semantik kein ? ergeben.*

Beweis. Es muss für die Operatoren $\neg$, $\vee$, $\bigcirc$ und $\mathcal{U}$ sowie die atomare Formel true und eine atomare Proposition a gezeigt werden, dass sich nie ? ergeben kann, wenn es den @-Operator nicht gibt. Die restlichen Operatoren ergeben sich durch die üblichen Äquivalenzen.

Seien φ und ψ LTL-Formeln. Sei weiter $w \in \Sigma^\omega$ ein beliebiges, unendliches Wort.

- Atomare Formeln

  Betrachtet man die atomare Formel true und eine atomare Proposition a, so folgt direkt aus der Definition der Semantik, dass sich kein ? ergeben kann.

- Negation

  Angenommen für alle $v \in \Sigma^\omega$ und alle $i \in \mathbb{N}$ gilt $[\![v, i \models \varphi]\!]_{\text{fDTL}_\omega} \in \{\top, \bot\}$. Für $\neg\varphi$ gilt dann

  $$[\![w, i \models \neg\varphi]\!]_{\text{fDTL}_\omega} = \overline{[\![w, i \models \varphi]\!]_{\text{fDTL}_\omega}}$$

  und damit folgt auch $[\![w, i \models \neg\varphi]\!]_{\text{fDTL}_\omega} \in \{\top, \bot\}$.

- Oder

  Angenommen für alle $v \in \Sigma^\omega$ und alle $i \in \mathbb{N}$ gilt $[\![v, i \models \varphi]\!]_{\text{fDTL}_\omega}, [\![v, i \models \psi]\!]_{\text{fDTL}_\omega} \in \{\top, \bot\}$. Betrachtet man $[\![w \models \varphi \vee \psi]\!]_{\text{fDTL}_\omega}$, so gilt

  $$[\![w \models \varphi \vee \psi]\!]_{\text{fDTL}_\omega} = [\![w \models \varphi]\!]_{\text{fDTL}_\omega} \sqcup [\![w \models \psi]\!]_{\text{fDTL}_\omega}$$

  Da die einzelnen Ergebnisse nur $\top$ oder $\bot$ sein können, folgt dies auch für eine Verknüpfung mit $\sqcup$. Damit gilt auch $[\![w \models \varphi \vee \psi]\!]_{\text{fDTL}_\omega} \in \{\top, \bot\}$.

- Next-Operator

  Angenommen für alle $v \in \Sigma^\omega$ und alle $i \in \mathbb{N}$ gilt $[\![v, i \models \varphi]\!]_{\text{fDTL}_\omega} \in \{\top, \bot\}$. Für $\bigcirc\varphi$ gilt dann

$$[\![w, i \models \bigcirc\varphi]\!]_{\text{fDTL}_\omega} = \begin{cases} \top & \text{wenn } [\![w, i+1 \models \varphi]\!]_{\text{fDTL}_\omega} = \top \\ \bot & \text{wenn } [\![w, i+1 \models \varphi]\!]_{\text{fDTL}_\omega} = \bot \\ ? & \text{sonst.} \end{cases}$$

  Nach der vorherigen Annahme decken die beiden ersten Fälle alle möglichen Fälle ab und damit gilt auch $[\![w, i \models \bigcirc\varphi]\!]_{\text{fDTL}_\omega} \in \{\top, \bot\}$.

- Until-Operator

  Angenommen für alle $v \in \Sigma^\omega$ und alle $i \in \mathbb{N}$ gilt $[\![v, i \models \varphi]\!]_{\text{fDTL}_\omega}, [\![v, i \models \psi]\!]_{\text{fDTL}_\omega} \in \{\top, \bot\}$. Für $\varphi \mathcal{U} \psi$ gilt dann

$$[\![w, i \models \varphi \mathcal{U} \psi]\!]_{\text{fDTL}_\omega} = \begin{cases} \top & \text{wenn } \exists j \geq i : [\![w, j \models \psi]\!]_{\text{fDTL}_\omega} = \top \wedge \\ & \qquad \forall i \leq h < j : [\![w, h \models \varphi]\!]_{\text{fDTL}_\omega} = \top \\ \bot & \text{wenn } \forall j \geq i : [\![w, j \models \psi]\!]_{\text{fDTL}_\omega} = \bot \vee \\ & \qquad \exists i \leq h < j : [\![w, h \models \varphi]\!]_{\text{fDTL}_\omega} = \bot \\ ? & \text{sonst} \end{cases}$$

  Da sowohl der erste als auch der zweite Fall nur $\top$ und $\bot$ Werte abfragen, muss nur gezeigt werden, dass der eine die Negation des anderen ist, um zu zeigen, dass der dritte Fall nicht eintritt.

  Es gilt

$$\begin{aligned} &\neg(\exists j \geq i : [\![w, j \models \psi]\!]_{\text{fDTL}_\omega} = \top \wedge \forall i \leq h < j : [\![w, h \models \varphi]\!]_{\text{fDTL}_\omega} = \top) \\ \equiv &\forall j \geq i : [\![w, j \models \neg\psi]\!]_{\text{fDTL}_\omega} = \top \vee \exists i \leq h < j : [\![w, h \models \neg\varphi]\!]_{\text{fDTL}_\omega} = \top \\ \equiv &\forall j \geq i : [\![w, j \models \psi]\!]_{\text{fDTL}_\omega} = \bot \vee \exists i \leq h < j : [\![w, h \models \varphi]\!]_{\text{fDTL}_\omega} = \bot. \end{aligned}$$

  Daraus folgt, dass die ersten beiden Fälle zueinander dual sind und damit gilt unter den Annahmen auch $[\![w, i \models \varphi \mathcal{U} \psi]\!]_{\text{fDTL}_\omega} \in \{\top, \bot\}$.

Es wurde für alle Operatoren gezeigt, dass, wenn alle Teilformeln eines Operators nur zu $\top$ oder $\bot$ ausgewertet werden können, sich auch für den Operator selber nur $\top$ oder $\bot$ ergeben kann. Da die innersten Teilformeln atomare Formeln sind und diese nur zu $\top$ oder $\bot$ ausgewertet werden können, ergibt sich auch für die Gesamtformel immer nur $\top$ oder $\bot$. □

Als nächstes wird nun die Äquivalenz von lokalen fDTL-Formeln, also fDTL-Formeln ohne entfernte Teilformel, und LTL_3-Formeln gezeigt. Dafür werden die Ergebnisse des vorherigen Lemmas benutzt, um die Äquivalenz von fDTL_ω und LTL in diesem Szenario zu zeigen, aus der die gewünschte Äquivalenz folgt.

Theorem 3.7 (Lokale fDTL-Formel)**.** *Sei p ein Prozess, Σ das Alphabet von Prozess p, $w \in \Sigma^+$ und $@_p\psi_1$ eine fDTL-Formel, wobei ψ_1 keinen @-Operator enthält. Dann gilt $[\![w \models @_p\psi_1]\!]_{fDTL} = [\![w \models \psi_1]\!]_{LTL_3}$.*

Beweis. Zu zeigen ist

$$[\![w \models \psi_1]\!]_{\text{LTL}_3} = \begin{cases} \top & \text{wenn } \forall u \in \Sigma^\omega : [\![wu \models \psi_1]\!]_{\text{LTL}} = \top \\ \bot & \text{wenn } \forall u \in \Sigma^\omega : [\![wu \models \psi_1]\!]_{\text{LTL}} = \bot \\ ? & \text{sonst} \end{cases}$$
$$= \begin{cases} \top & \text{wenn } \forall u \in \Sigma^\omega : [\![wu, 0 \models \psi_1]\!]_{\text{fDTL}_\omega} = \top \\ \bot & \text{wenn } \forall u \in \Sigma^\omega : [\![wu, 0 \models \psi_1]\!]_{\text{fDTL}_\omega} = \bot \\ ? & \text{sonst} \end{cases}$$
$$= [\![w \models @_p\psi_1]\!]_{\text{fDTL}},$$

wenn ψ_1 keinen @-Operator enthält. Um dies zu zeigen muss $\forall v \in \Sigma^\omega : [\![v \models \psi_1]\!]_{\text{LTL}} = [\![v, 0 \models \psi_1]\!]_{\text{fDTL}_\omega}$ gezeigt werden.

Sei im Folgenden $w \in \Sigma^\omega$ der gesamte Lauf von Prozess p, a eine atomare Proposition des Prozesses und $\varphi, \psi \in \text{LTL}$. Für den Beweis werden die atomaren Formeln true und a sowie die Operatoren $\neg$, $\vee$, $\bigcirc$ und $\mathcal{U}$ betrachtet. Im weiteren Verlauf werden die Ergebnisse für fDTL_ω mit Formeln ohne @-Operator aus Lemma 3.6 auf Seite 33 genutzt.

- Wahrheitswert true

  Für true gilt

$$[\![w \models \text{true}]\!]_{\text{LTL}} = \begin{cases} \top & \text{wenn } w \models \text{true} \\ \bot & \text{sonst.} \end{cases} = \top = [\![w, 0 \models \text{true}]\!]_{\text{fDTL}_\omega}$$

- Proposition a

Für eine atomare Proposition a gilt

$$\begin{aligned}[\![w \models a]\!]_{\mathrm{LTL}} &= \begin{cases} \top & \text{wenn } w \models a \\ \bot & \text{sonst.} \end{cases} \\ &= \begin{cases} \top & \text{wenn } a \in w_0 \\ \bot & \text{sonst.} \end{cases} \\ &= [\![w, 0 \models a]\!]_{\mathrm{fDTL}_\omega}\end{aligned}$$

- Negation

  Für die Negation gilt in LTL

$$\begin{aligned}[\![w \models \neg\varphi]\!]_{\mathrm{LTL}} &= \begin{cases} \top & \text{wenn } w \models \neg\varphi \\ \bot & \text{sonst.} \end{cases} \\ &= \begin{cases} \top & \text{wenn } w \models \neg\varphi \\ \bot & \text{wenn } w \not\models \neg\varphi \end{cases} \\ &= \begin{cases} \top & \text{wenn } w \not\models \varphi \\ \bot & \text{wenn } w \models \varphi \end{cases} \\ &= \begin{cases} \overline{\top} & \text{wenn } w \models \varphi \\ \overline{\bot} & \text{wenn } w \not\models \varphi \end{cases} \\ &= \overline{[\![w \models \varphi]\!]_{\mathrm{LTL}}}\end{aligned}$$

  Und in fDTL_ω gilt

$$[\![w, i \models \neg\varphi]\!]_{\mathrm{fDTL}_\omega} = \overline{[\![w, i \models \varphi]\!]_{\mathrm{fDTL}_\omega}}$$

  Damit wurde gezeigt, dass die Funktion der Negation in LTL äquivalent zu der in fDTL_ω ist.

- Oder

  Für die Verorderung gilt in LTL

$$\begin{aligned}[\![w \models \varphi \vee \psi]\!]_{\mathrm{LTL}} &= \begin{cases} \top & \text{wenn } w \models \varphi \vee \psi \\ \bot & \text{sonst} \end{cases} \\ &= \begin{cases} \top & \text{wenn } w \models \varphi \vee w \models \psi \\ \bot & \text{sonst} \end{cases} \\ &= \begin{cases} \top & \text{wenn } [\![w \models \varphi]\!]_{\mathrm{LTL}} = \top \vee [\![w \models \psi]\!]_{\mathrm{LTL}} = \top \\ \bot & \text{sonst} \end{cases} \\ &= [\![w \models \varphi]\!]_{\mathrm{LTL}} \sqcup [\![w \models \psi]\!]_{\mathrm{LTL}}\end{aligned}$$

Das Verhalten der Veroderung in LTL_3 ist damit dasselbe wie in fDTL_ω.

- Next-Operator

  Für den Next-Operator gilt

$$\begin{aligned}[\![w \models \bigcirc\varphi]\!]_{\mathrm{LTL}} &= \begin{cases} \top & \text{wenn } w \models \bigcirc\varphi \\ \bot & \text{sonst} \end{cases} \\ &= \begin{cases} \top & \text{wenn } w_{\geq 1} \models \varphi \\ \bot & \text{sonst} \end{cases} \\ &= [\![w_{\geq 1} \models \varphi]\!]_{\mathrm{LTL}}\end{aligned}$$

  Nach Lemma 3.6 auf Seite 33, weil dadurch der dritte Fall entfällt, gilt für fDTL

$$\begin{aligned}[\![w, 0 \models \bigcirc\varphi]\!]_{\mathrm{fDTL}_\omega} &= \begin{cases} \top & \text{wenn } [\![w, 0+1 \models \varphi]\!]_{\mathrm{fDTL}_\omega} = \top \\ \bot & \text{sonst} \end{cases} \\ &= [\![w, 1 \models \varphi]\!]_{\mathrm{fDTL}\omega}\end{aligned}$$

  Daher ist auch die Funktion des Next-Operators dieselbe.

- Until-Operator

  Nach Lemma 3.6 auf Seite 33, weil dadurch der dritte Fall in fDTL entfällt, gilt für den Until-Operator

$$\begin{aligned}[\![w \models \varphi \mathcal{U} \psi]\!]_{\mathrm{LTL}} &= \begin{cases} \top & \text{wenn } w \models \varphi \mathcal{U} \psi \\ \bot & \text{sonst} \end{cases} \\ &= \begin{cases} \top & \text{wenn } \exists i : w_{\geq i} \models \psi \wedge \forall j < i : w_{\geq j} \models \varphi \\ \bot & \text{sonst} \end{cases} \\ &= \begin{cases} \top & \text{wenn } \exists i : [\![w_{\geq i} \models \psi]\!]_{\mathrm{LTL}} = \top \wedge \forall j < i : [\![w_{\geq j} \models \varphi]\!]_{\mathrm{LTL}} = \top \\ \bot & \text{sonst} \end{cases} \\ &= \begin{cases} \top & \text{wenn } \exists i : [\![w, i \models \psi]\!]_{\mathrm{fDTL}_\omega} = \top \wedge \forall j < i : [\![w, j \models \varphi]\!]_{\mathrm{fDTL}_\omega} = \top \\ \bot & \text{sonst} \end{cases} \\ &= [\![w, 0 \models \varphi \mathcal{U} \psi]\!]_{\mathrm{fDTL}_\omega}\end{aligned}$$

  Dabei folgt das Gleichheitszeichen in der vorletzten Zeile, weil für alle anderen Operatoren bereits gezeigt wurde, dass sie sich in fDTL_ω und LTL für Formeln ohne @-Operator äquivalent verhalten.

Wie gezeigt wurde verhalten sich alle Operatoren und atomaren Formeln in LTL und fDTL_ω äquivalent, wenn eine Formel keinen @-Operator enthält. Damit folgt das zu Zeigende. □

Als letztes wird noch die Mächtigkeit von fDTL und ptDTL-Formeln, die noch mit einem Globally- oder einen Finally-Operator umgeben sind, betrachtet, da ansonsten mit ptDTL keine Eigenschaften monitorbar waren. Dabei wird der @-Operator nicht genauer betrachtet, da dieser in fDTL und ptDTL grundsätzlich anders funktioniert. Im Gegensatz zu ptDTL, wo für eine entfernte Formel immer der gesamte Lauf von hinten mit einer zweiwertigen Semantik ohne Impartiality betrachtet wird, wird in fDTL der gesamte Lauf von vorne mit einer dreiwertigen Semantik mit Impartiality und Anticipation betrachtet. Außerdem wartet der @-Operator auf Wahrheitswerte in den Nachrichten, die impartial sind und betrachtet daher unabhängig von der aktuell betrachteten Stelle des Wortes das gesamte Wort. Die Werte einer entfernten Formel in ptDTL sind damit deutlich stärker abhängig von dem Versandzeitpunkt der Nachrichten als in fDTL.

Theorem 3.8 (Mächtigkeit von fDTL bezüglich ptDTL)**.** *fDTL hat eine größere Mächtigkeit als ptDTL-Formeln, die mit einem Globally- oder Finally-Operator umgeben sind, wenn der @-Operator nicht genauer betrachtet wird.*

Beweis. Im Folgenden wird mit ptDTL die Logik bezeichnet, die dem eigentlichen ptDTL entspricht, wobei alle Formeln mit einem Globally- oder Finally-Operator umgeben sind. Für den Beweis muss zum Einen ptDTL $\subseteq$ fDTL gezeigt werden und zum Anderen eine fDTL Eigenschaft gefunden werden, die nicht durch ptDTL ausdrückbar ist.

Zuerst wird ptDTL $\subseteq$ fDTL gezeigt:

In [Zuc86] wurde in Theorem 6.1 auf Seite 35 gezeigt, dass Logiken, die nur die Zukunfts-Operatoren enthalten, genau so mächtig sind wie Logiken, welche die Vergangenheits- und die Zukunfts-Operatoren enthalten. Daraus folgt, dass die Vergangenheits-Operatoren über endlichen Worten maximal so mächtig sind, wie die Zukunfts-Operatoren. Daher gilt ptDTL $\subseteq$ fDTL.

Als nächstes wird gezeigt, dass es eine fDTL-Eigenschaft gibt, die nicht durch ptDTL ausdrückbar ist:

Um dies zu zeigen wird die Obligation-Eigenschaft $\varphi = @_p \Box a \wedge \Diamond b$ mit den atomaren Propositionen a und b und einem Prozess p betrachtet. Offensichtlich ist dies eine fDTL-Formel. Es gibt nun zwei Fälle:

- Für ptDTL-Formeln, die mit einem Globally-Operator umgeben sind:

  In diesem Fall müsste vor allem noch $\Diamond b$ dargestellt werden. Dies ist aber nicht möglich. Beim ersten Zeichen müsste unter anderem schon entschieden werden, ob $\Diamond b$ aktuell gilt oder nicht. Würde sich dort dafür entschieden werden, dass es nicht gilt, so ist durch das Globally die gesamte Formel nicht erfüllt. Sollte sich entschieden werden, dass es gilt, so würde dies bei jedem weiteren Zeichen fortgesetzt werden und sich daher insgesamt ergeben, dass die Formel erfüllt ist, egal, ob $\Diamond b$ erfüllt wird oder nicht.

- Für ptDTL-Formeln, die mit einem Finally-Operator umgeben sind:

  In diesem Fall ist es aus demselben Grund wie in dem vorherigen Fall nicht möglich $\Box a$ darzustellen, da der Globally- und der Finally-Operator dual zueinander sind.

Damit ist φ nicht durch ptDTL ausdrückbar und daher folgt insgesamt ptDTL $\subset$ fDTL, wenn das @ nicht genauer betrachtet wird. □

Wie gezeigt wurde sind aber die lokalen Operatoren in fDTL mächtiger als die in ptDTL, wodurch grundsätzlich mehr Eigenschaften spezifiziert werden können.

3.2.4 Vorteile

Da die fDTL-Semantik dreiwertig ist birgt die Logik einige Vorteile gegenüber ptDTL. Diese werden im Folgenden genauer betrachtet, da sie sich auch auf die Logik fSDTL auswirken, die am Ende dieses Kapitels definiert wird.

Ein Vorteil ist, dass die fDTL-Semantik Impartiality besitzt. Für lokal auswertbare Formeln ohne @-Operator folgt dies direkt aus Theorem 3.7 auf Seite 35, denn danach ist fDTL für lokal auswertbare Formeln äquivalent zu LTL_3. Da LTL_3 Impartiality besitzt, gilt dies in diesem Fall auch für fDTL. Für Teilformeln, die auf entfernten Prozessen ausgewertet werden, gilt Impartiality durch das lokale Auswerten der Teilformeln mit der fDTL-Semantik auf den entfernten Prozessen. Sollte eine Formel auf einem entfernten Prozess zu $\top$ oder $\bot$ ausgewertet werden, so steht dies auch als Wert in jeder später versandten Nachricht. Da die Werte in den Nachrichten auf dem empfangenen Prozess von dem @-Operator direkt übernommen werden und diese Werte impartial sind, sind auch die Werte von entfernt ausgewertete Teilformeln auf einem Prozess impartial. Dadurch bleibt auch die Gesamtformel impartial.

Mit fDTL sind auch mehr Eigenschaften monitorbar als mit ptDTL. Neben Safety- und Guarantee-Eigenschaften, die auch mit ptDTL monitorbar waren, lässt sich auch noch die gesamte Klasse Obligation sowie Teile von den Klassen Response und Persistence monitoren. Dies ist aufgrund der Impartiality und der dreiwertigen Semantik möglich wie in [FFM12] untersucht und in Unterabschnitt 2.4.1 auf Seite 14 kurz erläutert.

Auch Anticipation ist bei fDTL gegeben, zumindest lokal. Für lokal auswertbare Formeln ohne @-Operator folgt dies wieder direkt aus Theorem 3.7 auf Seite 35. Da LTL_3 Anticipation besitzt, gilt dies in diesem Fall auch für fDTL. Auch in Formeln mit @-Operator ist dies kein Problem, da die Werte des @-Operators impartial sind und in der fDTL-Semantik bei der Auswertung ein Erfüllbarkeitstest durchgeführt wird. Global ist Anticipation nicht möglich, da es ein asynchrones, verteiltes System ist, welches auf Nachrichten basiert. Da die Nachrichten beliebig lange brauchen können, bis sie empfangen werden und auch nicht direkt nach Auftreten eines Wahrheitswertes abgeschickt werden, ist es unmöglich, dass ein Prozess von einer Formel, die entfernt ausgewertet wird, sofort weiß, wann diese einen endgültigen Wahrheitswert erreicht.

Des Weiteren ist auch die Mächtigkeit von fDTL höher als die von ptDTL, wenn die ptDTL-Formeln von einem Globally- oder Finally-Operator umgeben sind und wenn man den @-Operator nicht genauer betrachtet. Dies folgt aus Theorem 3.8 auf der vorherigen Seite. Der direkte Vergleich mit der Mächtigkeit von ptDTL selber macht keinen Sinn, weil mit ptDTL keine Eigenschaften monitorbar sind.

3.2.5 Weitere Anforderungen

Durch die Nutzung von LTL als Grundlage sowie die Definition des @-Operators ergibt sich eine weitere Anforderung, die für ptDTL keine Rolle spielte.

Die Ergebnisse der entfernten Formeln sind impartial und ändern sich in der Definition von fDTL auch nicht mehr, wenn sich einmal ein Wert ergeben hat, der impartial ist. Wie in Theorem 3.5 auf Seite 30 gezeigt, kann ein @ aus jedem temporalen Operator in fDTL herausgezogen werden, da temporale Operatoren keinen Effekt mehr auf das @ haben. Es wäre daher sinnvoll, wenn die Auswertung einer entfernten Formel, deren Wahrheitswert zu einem Zeitpunkt benötigt wird, auch möglichst nah an diesem Zeitpunkt auf dem entfernten Prozess beginnt. Dadurch würde nicht mehr das Ergebnis der Auswertung der entfernten Formel vom Beginn des Laufes des entfernten Prozesses an für alle Zeitpunkte gelten. Außerdem passt dieses Verhalten auch mehr zu dem Verhalten der anderen Operatoren.

Ein Beispiel dafür wäre die LTL-Formel $\square(a \rightarrow \square b)$, wobei a und b Propositionen sind. Bei dieser Formel wird die Teilformel $\square b$ ausgewertet, nachdem ein a gesehen wurde, da die Formel aussagt, dass nach jedem a immer b gelten muss. Dies ist nicht möglich, wenn man die Formel mit fDTL verteilt auswerten will. Bei $@_p\square(a \rightarrow @_q\square b)$ würde es nur bedeuten, dass, wenn auf Prozess p ein a auftritt, auf Prozess q immer b gelten und gegolten haben muss, da $@_q\square b$ auf Prozess q immer von ganz vorne ausgewertet wird. Aus diesem Grund ist es egal, wann das a gesehen wird. $@_p\square(a \rightarrow \square @_q b)$ wäre auch keine Lösung dafür, da diese Formel nur abfragt, ob im ersten Schritt von Prozess q die Proposition b galt, wenn auf p ein a auftrat. Der Globally-Operator hätte in diesem Fall also überhaupt keine Auswirkung. Es ist demnach wünschenswert eine Möglichkeit zu haben, Formeln, die von einem entfernten Monitor ausgewertet werden, von einem bestimmten Zeitpunkt an auswerten zu lassen, um die vorher genannten Anforderungen zu erfüllen. Daraus folgt auch eine erhöhte Mächtigkeit der Logik folgen, wie später gezeigt wird.

Des Weiteren gibt es noch Anforderungen, die von fDTL nicht erfüllt werden. Es gibt noch immer kein Modell für eine Ausführung des Systems. Dies wird in den nächsten beiden Abschnitten durch die Beschreibung eines Systemmodells und der Einführung von Schedulern verbessert. Nach den Abschnitten, die das benötigte Modell des Systems beschreiben, werden Synchronisationsaktionen definiert, mit denen auch die, in diesem Abschnitt, neu hinzugekommene Anforderung erfüllt werden kann.

3.3 Systemmodell

In diesem Abschnitt wird das Modell für das Gesamtsystem beschrieben, welches im weiteren Verlauf als Grundlage dient. Dafür wird zuerst ein Modell definiert, welches das Verhalten eines Prozesses beschreibt. Als nächstes folgt ein Modell für die gesamte Architektur des Systems. In diesem wird zum Einen der Aufbau des Systems, daher die Verteilung der Prozesse und deren Monitore, und zum Anderen die Kommunikation zwischen den einzelnen Teilnehmern, also das Versenden und Empfangen von Nachrichten und Austauschen von Wahrheitswerten, dargestellt.

3.3.1 Prozess als Transitionssystem

Die Beschreibung der Prozesse in Unterabschnitt 2.5.3 auf Seite 18 war nur darauf ausgelegt die Prozesse so zu beschreiben, wie sie in [SVAR04] für ptDTL dargestellt wurden und anzugeben, was alles zu einem Prozess gehört. Dabei wurde angenommen, dass sämtliches Wissen über den Lauf des Prozesses schon vorher vorhanden ist. In dieser Definition soll ein Modell für das Verhalten von Prozessen definiert werden, so dass der Lauf durch einzelnes Ausführen von Schritten des Prozesses aufgebaut werden kann. Die Bezeichnungen werden dabei aus Unterabschnitt 2.5.3 auf Seite 18 übernommen.

Im Folgenden werden Prozesse durch ein Transitionssystem modelliert, durch dessen Zustände angegeben wird, welche Propositionen des Prozesses aktuell gelten. Durch eine Transition von einem Zustand zu einem anderen werden die aktuell geltenden Propositionen geändert.

Definition 3.9 (Prozess)**.** *Sei p ein Prozess mit einer exklusiven Menge von atomaren Propositionen AP^p, in der auch die Empfangs- und Sendezeitpunkte von Nachrichten enthalten sind, und einem Alphabet $\Sigma^p = 2^{AP^p}$. p wird durch ein Transitionssystem $T^p = (V^p, E^p, F^p)$ modelliert, wobei*

- *(V^p, E^p) ein gerichteter Graph und*
- *$F^p : V^p \to \Sigma^p$ eine Funktion ist.*

Durch diese Art von Modell wird das Verhalten der Prozesse eindeutig modelliert. Durch die Funktion F werden durch die Zustände die jeweils geltenden Propositionen sowie die Nachrichten, die empfangen oder gesendet werden, angegeben. In dem Transitionssystem kann ein Schritt in einer Ausführung des Prozesses dargestellt werden und durch unendlich viele Schritte in dem Transitionssystem wird ein gesamter Lauf beschrieben, indem die besuchten Zustände in der Reihenfolge des Besuchs als Buchstaben des Wortes mit Hilfe der Funktion F des Transitionssystems interpretiert werden.

Der Vorteil dieser Modellierung ist, dass sie zusammen mit einem Scheduler verwendet werden kann, um eine Ausführung des gesamten Systems zu erstellen. Dieses Verfahren und die Scheduler werden in dem Abschnitt nach dem Architekturmodell definiert.

3.3.2 Architekturmodell

Im Folgenden wird das Architekturmodell für das Gesamtsystem beschrieben. Dieses spiegelt die Verteilung von Prozessen und Monitoren sowie die Kommunikation zwischen den einzelnen Teilnehmern wider.

In diesem Modell besitzt das System eine beliebige Anzahl an Prozessen, von denen jeder mit jedem kommunizieren kann, indem untereinander Nachrichten ausgetauscht werden. Diese Nachrichten sind entweder reine Datenmengen, die neben den herkömmlichen Daten auch Wahrheitswerte enthalten, die der empfangene Prozess von dem Sendenden für entfernte Formeln benötigt,

oder auch Nachrichten, die zur Synchronisation benutzt werden, so genannte Synchronisationsaktionen, die in Abschnitt 3.5 auf Seite 45 definiert werden.

Des Weiteren besitzt jeder Prozess eine beliebige Anzahl an Monitoren. Jeder Monitor kann nur mit dem Prozess kommunizieren, zu dem er gehört und kümmert sich immer um die Auswertung genau einer Formel. Das heißt, ein Monitor versendet und empfängt keine Nachrichten, noch löst er den Versand von Nachrichten auf seinem zugehörigen Prozess aus. Alle Nachrichten, die ein Prozess an andere Prozesse versendet, sind unabhängig von den Monitoren. Ein Monitor sendet einem Prozess nur Wahrheitswerte und ein Prozess sendet einem Monitor zum Einen die Propositionen, die bei ihm gelten, und zum Anderen die Wahrheitswerte entfernter Formeln. Die Wahrheitswerte, die der Prozess über entfernte Formeln von anderen Prozessen erfahren hat, können für einen Monitor für die Auswertung seiner Formel wichtig sein. Die Wahrheitswerte, die ein Monitor durch die Auswertung seiner Formel erhält, sind entweder für das gesamte System oder für Auswertungen der Formeln entfernter Monitore wichtig, weshalb der Prozess diese benötigt, um sie in Nachrichten an andere Prozesse zu verschicken.

Wenn ein Monitor einen Wahrheitswert einer entfernten Formeln benötigt, so fragt er seinen Prozess, was dieser über den gesuchten Wert weiß. Der Prozess übermittelt den angefragten Wahrheitswert dann an den Monitor, sofern es einen gibt, und dieser wertet mit diesem Wahrheitswert seine Formel aus. Will ein Prozess eine Nachricht versenden, so fragt er die aktuellen Wahrheitswerte seiner Monitore ab. Diese hängt er an die Nachricht an, damit der empfangene Prozess von den Wahrheitswerten erfährt, um diese seinen Monitoren weiter zu geben, wenn dies benötigt wird. Empfängt ein Prozess eine Nachricht, so sucht er sich die benötigten Werte heraus und speichert sie ab.

Ein Beispiel für die Architektur wurde in Abbildung 3.1 auf der nächsten Seite dargestellt.

3.4 Scheduler

Das Ziel ist es nun, die übrigen Nachteile von ptDTL zu entfernen sowie die, in Unterabschnitt 3.2.5 auf Seite 40 genannte, hinzugekommene Anforderung zu erfüllen. Dazu werden zunächst Scheduler eingeführt, um mit der Modellierung der Prozesse als Transitionssysteme ein Modell für eine Ausführung zu erhalten.

Ein Scheduler wird eine Ausführungsreihenfolge für die verschiedenen Prozesse angeben, daher beschreiben, wann welcher Prozess in einer Ausführung des Systems einen Schritt durchführt.

Definition 3.10 (Scheduler)**.** *Sei P die Menge aller Prozesse. Ein Scheduler ist eine Ausführungsreihenfolge der Prozesse $s \in P^\omega$.*

Für einen Prozess $p \in P$ ergibt sich die Menge aller Stellen, an denen p einen Schritt macht, s^p, durch

$$s^p = \{i \in \mathbb{N} \mid s_i = p\}$$

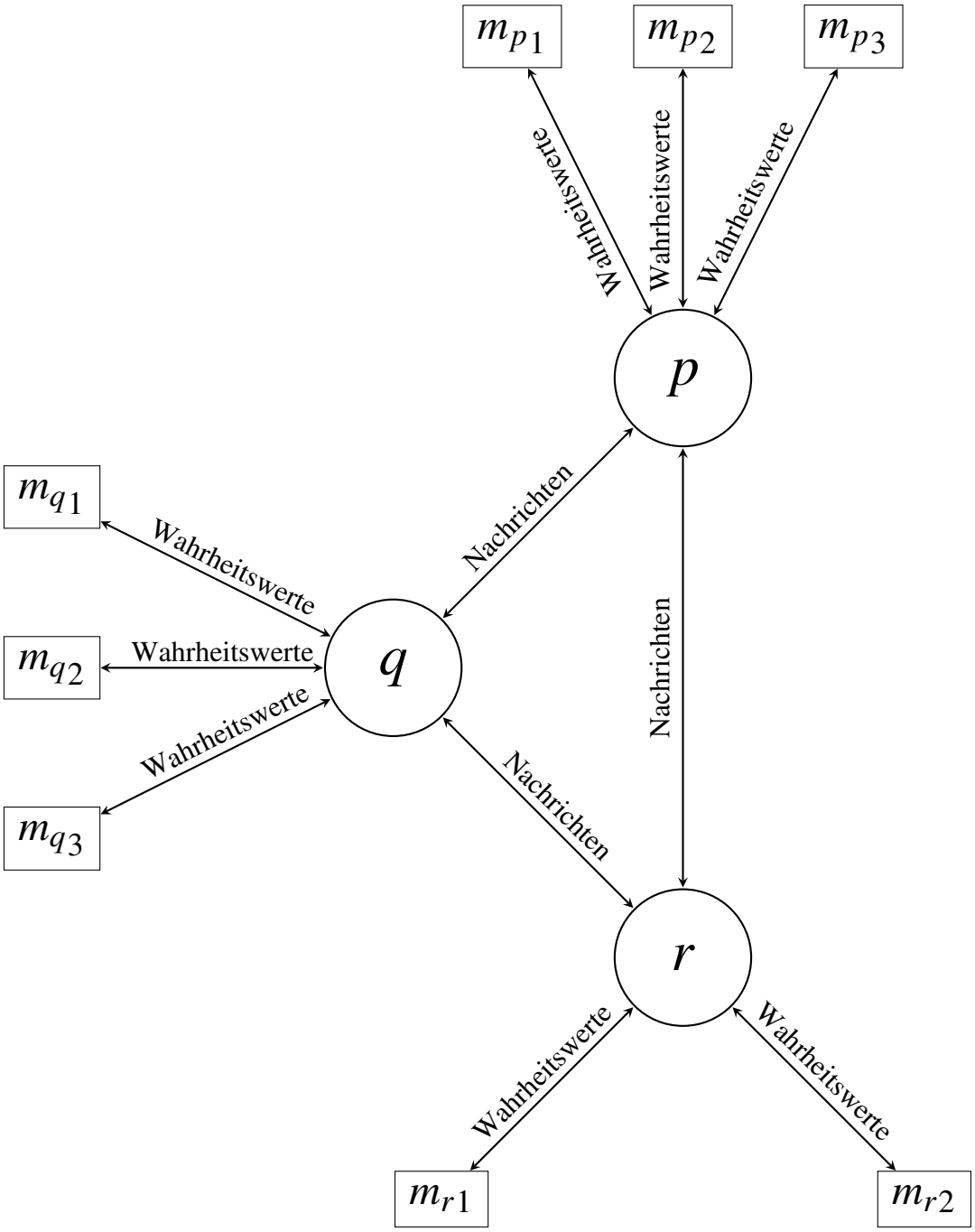


Abbildung 3.1: Das Architekturmodell für Prozesse (Kreise), Monitore (Rechtecke) sowie die Kommunikation in dem System. In diesem Fall gibt es drei Prozesse, p, q und r, sowie acht Monitore, von denen jeweils drei zu Prozess p und q gehören und zwei zu Prozess r. Die Kanten stellen die Kommunikation dar. Wie man sieht können die drei Prozesse mittels Nachrichten untereinander kommunizieren und an diese Nachrichten werden Wahrheitswerte der Monitore des versendenden Prozesses angehängt. Die Kommunikation von einem Prozess und einem Monitor beschränkt sich auf den Austausch von Wahrheitswerten. Außerdem sendet der Prozess die, auf ihm geltenden, Propositionen an seine Monitore. Dies wurde für bessere Übersicht in dem Bild weggelassen.

Ein Scheduler bestimmt durch seine Ausführungsreihenfolge s, wann welcher Prozess einen Schritt durchführt. Dadurch repräsentiert ein Scheduler eine mögliche zeitliche Abfolge der Schritte der Prozesse, die durch die Asynchronität des Systems während einer Ausführung entstehen kann.

In der Form, wie eine Ausführungsreihenfolge eines Schedulers dargestellt wird, kann sie allerdings nicht ausgewertet werden, da sie nur eine Folge aus Prozessen ist. Aus diesem Grund wird als nächstes die Ausführungsreihenfolge erst in eine Folge von Zuständen und dann mittels der F-Funktionen der Transitionssysteme, welche die Zustände auf die Menge der, in den Zuständen geltenden, Propositionen abbildet, in ein Wort aus Mengen von Propositionen der Prozesse umgewandelt.

Definition 3.11 (Ausführung)**.** *Sei P die Menge aller Prozesse, für alle $p \in P$ $next_p : V^p \to V^p$ eine Funktion, die von einem Zustand in T^p auf einen Folgezustand abbildet, $q_{0,p}$ der Startzustand von (V^p, E^p) und s eine Ausführungsreihenfolge eines Schedulers.*

Eine mögliche Zustandsfolge z für s ergibt sich durch

$$z_i = \begin{cases} q_{0,p} & i \in s^p \wedge \nexists j < i : j \in s^p \\ next_p(z_j) & i \in s^p \wedge \exists j < i : j \in s^p \wedge \nexists j < h < i : h \in s^p \end{cases}$$

Eine Ausführung w für s ergibt sich dann durch

$$w_i = F^p(z_i) \text{ für } i \in s^p$$

Eine Ausführung ist damit ein konkreter Lauf eines Systems. Die Ausführung gibt genau an, wann bei welchem Prozess welche Propositionen und Nachrichten galten und in welcher Reihenfolge die Prozesse ausgeführt wurden. Zusätzlich geben die Mengen s^p für alle Prozesse an, welche der Schritte in dem Lauf des Systems zu ihnen gehören, wodurch sich der Lauf eines einzelnen Prozesses rekonstruieren lässt. Dadurch können Formeln über einer Ausführung ausgewertet werden.

Es sind allerdings nicht alle Ausführungen eine sinnvolle Ausführung der Prozesse. Aus diesem Grund werden in der folgenden Definition die „erlaubten“ Ausführungen eingeschränkt.

Definition 3.12 (Erlaubte Ausführungen bezüglich Nachrichten)**.** *Sei P die Menge aller Prozesse und $q \in P$ ein Prozess. Eine Ausführung w für eine Ausführungsreihenfolge s heißt erlaubt bezüglich Nachrichten, wenn*

$$\forall p \in P : \forall i \in s^p : \downarrow_j^q \in w_i \to \exists 0 \leq h < i : h \in s^q \wedge \uparrow_j^p \in w_h$$

gilt.

Durch diese Definition sind Ausführungen nur dann bezüglich Nachrichten erlaubt, wenn vor jedem Empfang einer Nachricht auch eine entsprechende Nachricht gesendet wurde.

Sobald in fDTL Scheduler integriert wurden, wird eine Formel nicht mehr über einem Wort ausgewertet, sondern über der Ausführung einer Ausführungsreihenfolge eines Schedulers. Die Frage ist daher, ob eine Ausführung für eine Ausführungsreihenfolge eine Formel erfüllt.

3.5 Synchronisationsaktionen

Reale asynchrone, verteilte Systeme haben im Allgemeinen eine Möglichkeit, dass sich die einzelnen Akteure (Prozesse) untereinander synchronisieren können. Dies ist nötig, da die Prozesse zusammenarbeiten sollen und daher kein Prozess beliebig fortschreiten darf, während ein anderer noch sehr wenige Schritte durchgeführt hat. So eine Synchronisation wird mit Nachrichten durchgeführt, welche von einem Prozess an einen anderen, wartenden, gesendet werden, wenn der Prozess einen bestimmten Marker überschritten hat. Diese Nachricht symbolisiert dem wartenden Prozess, dass er fortfahren darf.

Diese Synchronisationen sind natürlich auch für die theoretische Betrachtung für die Menge der erlaubten Ausführungen interessant, damit diese noch weniger Ausführungen enthält, die keiner möglichen Ausführungsreihenfolge der Prozesse entsprechen. Die Synchronisationen werden in dem System von den einzelnen Prozessen ausgelöst. Aus diesem Grund müssen diese als Synchronisationsaktionen in den Transitionssystemen der Prozesse wie andere Propositionen gehandhabt werden, weshalb sie, wie auch Sende- und Empfangspunkte von Nachrichten, zur Menge der Propositionen eines Prozesses gehören. Aus diesem Grund treten sie auch, wie alle anderen Propositionen, in den Läufen der Prozesse auf.

Eine weitere wünschenswerte Eigenschaft von Synchronisationsaktionen wäre, dass ab diesen Formeln neu ausgewertet werden können. Der Prozess benachrichtigt bei Überschreitung von bestimmten Synchronisationspunkten seine Monitore, damit diese dann ihre Formeln parallel vom Beginn des Laufes des Prozesses sowie von dem Synchronisationspunkt an auswerten können. Dieser stellt sozusagen den Anfang eines neuen Laufes dar. Die entstehenden Wahrheitswerte bei der Auswertung einer Formel von Beginn des Laufes und ab den Synchronisationspunkten werden dann alle an versandte Nachrichten angehängt und die Monitore des empfangenen Prozesses suchen sich die benötigten heraus, indem sie herausfinden, welche Synchronisationspunkte ihr Prozess bereits überschritten hat. Mit diesem Verfahren kann die neue Anforderung aus Unterabschnitt 3.2.5 auf Seite 40 erfüllt werden. Die Definition für dieses Verhalten wird in dem Abschnitt über fSDTL angegeben.

Im Folgenden werden Synchronisationsaktionen definiert, mit denen sich zwei Prozesse synchronisieren können. Danach werden erlaubte Ausführungen bezüglich Synchronisationsaktionen definiert, so, dass eine Ausführung nur noch dann erlaubt ist, wenn in jedem Schritt nur Prozesse ausgeführt werden, die aktuell nicht auf einen anderen warten müssen.

Eine Synchronisationsaktion ist eine Synchronisierung zweier Prozesse durch eine Nachricht, die an einem Sendepunkt abgesendet wird und an oder vor einem Wartepunkt auf einem anderen Prozess ankommt, an dem dieser andere Prozess wartet und diesem signalisiert, dass er fortschreiten darf.

Definition 3.13 (Synchronisationsaktion)**.** *Seien im Folgenden p und q Prozesse und $i \in \mathbb{N} \setminus \{0\}$ die ID einer Synchronisierungsaktion.*

Ein Wartepunkt $\lambda_{\downarrow_i}^{p}$ von Prozess q ist ein Zeitpunkt, ab dem Prozess q warten muss, bis er die entsprechende Synchronisierungsnachricht von Prozess p erhalten hat. Ein Sendepunkt $\lambda_{\uparrow_i}^{q}$ von

Prozess p ist der Zeitpunkt, an dem p die Synchronisierungsnachricht an Prozess q sendet, damit er über den Wartepunkt $\lambda_{\downarrow_i}^p$ hinaus fortfahren darf. Eine solche Synchronisierungsaktion wird auch einseitige Synchronisierungsaktion genannt.

Ein Synchronisierungspunkt $\lambda_{\updownarrow_i}^p$ auf Prozess q ist sowohl ein Wartepunkt $\lambda_{\downarrow_i}^p$ als auch ein Sendepunkt $\lambda_{\uparrow_i}^p$. Entsprechend gibt es immer auch einen Synchronisierungspunkt $\lambda_{\updownarrow_i}^q$ auf Prozess p. An einem Synchronisierungspunkt warten beide Prozesse auf die jeweilige Synchronisationsnachricht des anderen Prozesses. Eine solche Synchronisierungsaktion wird auch beidseitige Synchronisierungsaktion genannt.

Die entsprechenden Synchronisationsaktionen gehören im Folgenden zu der Menge von Propositionen AP^p eines Prozesses p.

Synchronisationsaktionen befinden sich wie beschrieben in der Menge der Propositionen eines Prozesses, wie Nachrichten auch. Das heißt, dass sie in einer Ausführung für einen Scheduler in den Mengen auftreten und sich daher auch in den entstehenden Läufen der Prozesse befinden.

Durch die Synchronisationsaktionen werden die erlaubten Ausführungen weiter eingeschränkt. Ein Prozess darf nicht über einen Wartepunkt hinaus ausgeführt werden, solange der entsprechend andere noch nicht den Sendepunkt erreicht hat. Die erlaubten Ausführungen bezüglich Synchronisationsaktionen werden in der folgenden Definition beschrieben.

Definition 3.14 (Erlaubte Ausführungen bezüglich Synchronisationsaktionen)**.** *Sei P die Menge aller Prozesse und $p,q \in P$ zwei Prozesse. Eine Ausführung w für eine Ausführungsreihenfolge s eines Schedulers heißt erlaubt bezüglich Synchronisationsaktionen, wenn gilt:*

$$\forall i \in s^p : (\exists \lambda_{\downarrow_j}^q \in w_i \rightarrow \forall k > i \in s^p : \exists\, 0 \leq l < k : l \in s^q \wedge \lambda_{\uparrow_j}^p \in w_l)$$

Nach dieser Definition ist eine Ausführung nur dann bezüglich Synchronisationsaktionen erlaubt, wenn es für jeden Wartepunkt $\lambda_{\downarrow_j}^q$ eines Prozesses p in einem Schritt der Ausführung, an dem auf Prozess q gewartet werden muss, vor dem nächsten Schritt des Prozesses p einen Sendepunkt $\lambda_{\uparrow_j}^p$ in einem Schritt des Prozesses q gibt.

Jetzt existieren zwei Definitionen für erlaubte Ausführungen, einmal bezüglich Nachrichten und einmal bezüglich Synchronisationsaktionen. In der nächsten Definition werden diese beiden zusammengefasst.

Definition 3.15 (Erlaubte Ausführungen)**.** *Eine Ausführung heißt erlaubt, wenn sie sowohl bezüglich Nachrichten als auch bezüglich Synchronisationsaktionen erlaubt ist.*

Die letzte Definition beschreibt die Menge der erlaubten Ausführungen. Nur Elemente dieser Menge werden im Weiteren betrachtet, da nicht erlaubte Ausführungen in realen Systemen nicht auftreten können und es daher nicht wichtig ist, diese zu betrachten.

3.6 Future Synchronized Distributed Temporal Logic

In diesem Abschnitt wird die Logik fDTL um Synchronisationsaktionen und Scheduler zur future Synchronized Distributed Temporal Logic (fSDTL) erweitert. Dadurch sollen die letzten, noch unerfüllten, Anforderungen erfüllt werden. fSDTL basiert auf fDTL und setzt die dort vorgestellten Ideen fort.

Aufgrund der Synchronisationsaktionen wird es nun möglich sein, Formeln, die durch ein @ auf einem entfernten Monitor ausgewertet werden, ab jedem Sendepunkt erneut auszuwerten. Dafür werden die Monitore des Prozesses, auf denen entfernte Formeln ausgewertet werden, diese parallel ab dem Beginn des Laufes des Prozesses sowie ab jedem seiner Sendepunkte erneut auswerten. In den Nachrichten an den Prozess, dessen Monitore die Formeln benötigen, werden dann alle diese Wahrheitswerte mitgeschickt und die Monitore des Prozesses können sich die benötigten heraussuchen. In fDTL war dies noch nicht möglich, denn dort gab es noch keine Synchronisationsaktionen. Deshalb hätte dort aktiv durch die Monitore veranlasst werden müssen, dass eine Nachricht gesendet wird, um die entfernten Monitore anzustoßen, damit sie ihre Formel erneut auswerten. Jetzt kann ein Sendepunkt als Auswertungsanfang genommen werden, welcher direkt auf dem entfernten Prozess entsteht und dadurch wird keine Nachricht benötigt. Die Monitore des Prozesses, welcher Nachrichten von dem entfernten Prozess empfängt, suchen in dem Lauf ihres Prozesses nach dem neusten Wartepunkt mit dem entfernten Prozess vor dem Zeitpunkt, ab dem die entfernte Formel ausgewertet werden soll. In den folgenden empfangenen Nachrichten von dem entfernten Prozess werden dann die Wahrheitswerte der Formel ab dem, zu dem gefundenen Wartepunkt korrespondierenden, Sendepunkt betrachtet. Dadurch werden nicht mehr alle entfernt ausgewerteten Formeln dauerhaft von Anfang des Laufes des entfernten Prozesses an ausgewertet, wie es bei fDTL der Fall war. Der Vorteil daran ist, dass dann auch entfernte Formeln ähnlich funktionieren, wie man es aufgrund der Funktion der lokalen Formeln erwartet. Außerdem wird die Mächtigkeit der Logik erhöht.

Die Semantik wird für eine erlaubte Ausführung für einen Scheduler definiert und abhängig von der Ausführung ergibt sich dann ein bestimmter Wahrheitswert. Alle nicht erlaubten Ausführungen werden automatisch zu $\bot$ ausgewertet.

Zuerst wird allerdings noch die Definition der Nachrichten erweitert, damit die Nachrichten mit Schedulern und dessen Ausführungen funktionieren und die benötigten Wahrheitswerte ab jedem Sendepunkt enthalten.

Definition 3.16 (Scheduler-Nachrichten)**.** *Eine Scheduler-Nachricht ist eine für Scheduler angepasste Nachricht und enthält zusätzlich zu den Wahrheitswerten der entfernen Formeln jeweils noch die Wahrheitswerte ab einem Sendepunkt bis zum Absendezeitpunkt der Nachricht für die entfernten Formeln. Sei TL die temporale Logik, die verwendet wird, und $\mathbb{B}_x$ der Verband, über den die Semantik von TL definiert ist. Eine Scheduler-Nachricht ist eine Funktion $TL \times \mathbb{N} \to \mathbb{B}_x$.*

Seien $p,q \in P$ Prozesse, s eine Ausführungsreihenfolge eines Schedulers und w eine Ausführung von s. Für eine Formel φ der temporalen Logik TL, die von Monitor m_p des Prozesses p benötigt

und von Monitor m_q des Prozesses q ausgewertet wird, ein $b \in \mathbb{B}_x$ und $\uparrow_i^p \in w_k, k \in s^q$ gilt:

$$\uparrow_i^p (\varphi, 0) = b \wedge \downarrow_i^q (\varphi, 0) = b \wedge [\![w_{0..k} \models \varphi]\!]_{TL} = b$$

Für die Werte ab den Sendepunkten gilt

$$\uparrow_i^p (\varphi, m) = b \wedge \downarrow_i^q (\varphi, m) = b \wedge (\exists j \leq k : j \in s^q \wedge \lambda_{\uparrow_m^p} \in w_j \wedge [\![w_{j..k} \models \varphi]\!]_{TL} = b),$$

wobei $m > 0$ gilt und $\lambda_{\uparrow_m^p}$ der m-te Sendepunkt von Prozess q an Prozess p ist.

Im Gegensatz zu den normalen Nachrichten bilden Scheduler-Nachrichten von einer Formel und einer natürlichen Zahl auf einen Wahrheitswert ab. Die natürliche Zahl ist dabei die ID des Sendepunktes, der betrachtet werden soll. Der Teil der Definition mit den Parametern φ und 0 entspricht den Werten einer Scheduler-Nachricht, die auch in einer normalen Nachricht stehen würden. Die Werte für die Parameter φ und m mit $m > 0$ sind neu und werden bestimmt, indem der entsprechende Sendepunkt auf dem entfernten Prozess mit ID m gesucht und die Formel von der Stelle mit dem Sendepunkt bis zu dem Versand der Nachricht ausgewertet wird. Gibt es den gesuchten Sendepunkt noch nicht bis zum Versand der Nachricht, so gibt es keinen Wert für φ, m.

Im Folgenden werden in der Definition der Semantik die eben definierten Scheduler-Nachrichten benutzt. Solange der Kontext eindeutig ist werden sie auch einfach nur als „Nachrichten“ bezeichnet.

3.6.1 fSDTL

Die Syntax von fSDTL entspricht exakt derer von fDTL.

Die dreiwertige Semantik entspricht einer abgeänderten Version der Definition der Semantik für fDTL. Statt über den Läufen der Prozesse ist diese dreiwertige Semantik über einen endlichen Präfix einer Ausführung eines Schedulers definiert. Weiter wird, wie vorher erwähnt, die Definition des @-Operators geändert, um Formeln immer ab der letzten Synchronisationsaktion auszuwerten.

Für alle lokal auswertbaren Teilformeln einer Formel, daher alle die, die nicht von einem @ umschlossen werden, auf einem Prozess p, müssen nur die Stellen des Wortes der Ausführung des Schedulers betrachtet werden, an denen der Prozess p einen Schritt gemacht hat. Diese Stellen ergeben den Lauf von p und entsprechen s^p für eine Ausführungsreihenfolge s. Für die Wahrheitswerte von Formeln, die von entfernten Monitoren ausgewertet werden, muss die neuste, empfangene Nachricht von dem entfernten Prozess betrachtet werden.

Definition 3.17 (fSDTL-Semantik)**.** *Sei P die Menge aller Prozesse, $p, q \in P$ Prozesse, S die Menge aller Ausführungsreihenfolge aller Scheduler und $s \in S$, W_{pref} die Menge aller endlichen Präfixe von erlaubten Ausführungen von Ausführungsreihenfolgen, $w \in W_{pref}$ ein endlicher Präfix der Ausführung von s, W die Menge aller erlaubten Ausführungen von Ausführungsreihenfolgen und W' die Menge aller Fortsetzungen ohne Nachrichten von w, sodass für alle $w' \in W'$ ww' eine erlaubte*

Ausführung ist. Sei weiter a eine atomare Proposition aus einer endliche Menge AP^p von atomaren Propositionen des Prozesses p und $i \in \mathbb{N}$. Für die Definition der Semantik wird die Hilfsfunktion $last_{\lambda_\downarrow} : W \times \mathbb{N} \times P \times P \to \mathbb{N}$ mit

$$last_{\lambda_\downarrow}(v,i,p,q) = \begin{cases} m & wenn\ \exists j < i : j \in s^p \wedge \lambda_{\downarrow m}^q \in v_j \wedge \nexists i \geq h \geq j : h \in s^p \wedge \lambda_{\downarrow n}^q \in v_h \\ 0 & sonst \end{cases}$$

genutzt, welche den Index des letzten Wartepunktes von Prozess p auf Prozess q vor der Position i in v zurückgibt.

Die Semantik von fSDTL ist für eine Formel über den Prozess p durch die Auswertungsfunktion $[\![.]\!]_{fSDTL} : W_{pref} \times fSDTL \to \mathbb{B}_3$ wie folgt definiert:

$$[\![w \models @_p \varphi]\!]_{fSDTL} = \begin{cases} \top & wenn\ \forall u \in W' : [\![wu, 0 \models \varphi]\!]_{fSDTL_\omega} = \top \\ \bot & wenn\ \forall u \in W' : [\![wu, 0 \models \varphi]\!]_{fSDTL_\omega} = \bot \\ ? & sonst \end{cases}$$

wobei $[\![.]\!]_{fSDTL_\omega} : W \times \mathbb{N} \times fSDTL \to \mathbb{B}_3$ wie folgt definiert ist:

$$[\![w, i \models true]\!]_{fSDTL_\omega} = \top$$

$$[\![w, i \models a]\!]_{fSDTL_\omega} = \begin{cases} \top & wenn\ \exists j \geq i : j \in s^p \wedge a \in w_j \wedge \nexists i \leq h < j : h \in s^p \\ \bot & sonst \end{cases}$$

$$[\![w, i \models \neg\varphi]\!]_{fSDTL_\omega} = \overline{[\![w, i \models \varphi]\!]_{fSDTL_\omega}}$$

$$[\![w, i \models \varphi \vee \psi]\!]_{fSDTL_\omega} = [\![w, i \models \varphi]\!]_{fSDTL_\omega} \sqcup [\![w, i \models \psi]\!]_{fSDTL_\omega}$$

$$[\![w, i \models \bigcirc\varphi]\!]_{fSDTL_\omega} = \begin{cases} \top & wenn\ \exists j > i : j \in s^p \wedge \exists! i \leq h < j : h \in s^p \wedge \\ & \quad [\![w, j \models \varphi]\!]_{fSDTL_\omega} = \top \\ \bot & wenn\ \exists j > i : j \in s^p \wedge \exists! i \leq h < j : h \in s^p \wedge \\ & \quad [\![w, j \models \varphi]\!]_{fSDTL_\omega} = \bot \\ ? & sonst \end{cases}$$

$$[\![w,i \models \varphi \mathcal{U} \psi]\!]_{fSDTL_\omega} = \begin{cases} \top & \text{wenn } \exists j \geq i : [\![w,j \models \psi]\!]_{fSDTL_\omega} = \top \wedge \\ & \quad \forall i \leq h < j : [\![w,h \models \varphi]\!]_{fSDTL_\omega} = \top \\ \bot & \text{wenn } \forall j \geq i : [\![w,j \models \psi]\!]_{fSDTL_\omega} = \bot \vee \\ & \quad \exists i \leq h < j : [\![w,h \models \varphi]\!]_{fSDTL_\omega} = \bot \\ ? & \text{sonst} \end{cases}$$

$$[\![w,i \models @_q \varphi]\!]_{fSDTL_\omega} = \begin{cases} \downarrow_k^q (@_p \varphi, m) & \text{wenn } last_{\lambda_\downarrow}(w,i,p,q) = m \wedge \exists j \geq 0 : \\ & \quad \downarrow_k^q \in w_j \wedge \downarrow_k^q (@_p \varphi, m) \neq ? \\ ? & \text{sonst} \end{cases}$$

wobei $\exists!$ *für „es existiert genau ein“ steht.*

Es gelten dieselben Äquivalenzen wie in LTL und fDTL.

Für die Definition der Semantik wird zu Beginn eine Hilfsfunktion definiert, welche allein dazu dient, die Definition des @-Operators übersichtlich zu gestalten. Die Funktion $last_{\lambda_\downarrow}(v,i,p,q)$ sucht für eine Ausführung v einer Ausführungsreihenfolge, ein $i \in \mathbb{N}$ und zwei Prozesse p und q den letzten Wartepunkt von Prozess p auf Prozess q vor der i-ten Stelle der Ausführung v heraus und gibt dessen ID zurück. Wird keiner gefunden, so wird 0 zurückgegeben. Der Sinn dabei ist, dass man, wenn ein Wartepunkt vor der i-ten Stelle gefunden wird, weiß, dass ein entsprechender Sendepunkt auf einem anderen Prozess erreicht oder überschritten worden sein muss. Dies liegt an der Definition der Synchronisationsaktionen, welche so definiert wurden, dass ein Wartepunkt nur überschritten werden darf, wenn auf dem entsprechenden anderen Prozess der korrespondierende Sendepunkt erreicht wurde. Aus diesem Grund bietet es sich an, die Formeln ab den Sendepunkten erneut auszuwerten, da durch den, mit der Funktion $last_{\lambda_\downarrow}$ gefundenen Wartepunkt, bekannt ist, welcher Sendepunkt auf dem entfernten Prozess schon überschritten worden sein muss und damit auch, welcher Wert aus den Nachrichten abgefragt werden muss.

Wie vorher beschrieben, wertet die Semantik die Formeln nun über einem endlichen Präfix einer Ausführung einer Ausführungsreihenfolge aus. Die unendlichen Verlängerungen wurden, wie auch bei fDTL, wieder ohne Nachrichten definiert. Bei einer Auswertung einer Formel werden, für eine Formel auf einem Prozess p, immer die Stellen der Ausführung betrachtet, an denen p jeweils einen Schritt macht. Die Überprüfung, ob an einer Stelle ein bestimmter Prozess einen Schritt gemacht hat, wird so durchgeführt, dass möglichst wenige Operatoren, nämlich nur die Propositionen und der Next-Operator ($\bigcirc$), betroffen sind.

Wie erwähnt, wurde der @-Operator umdefiniert. Für die Definition des @-Operators wird nun die vorher erklärte Hilfsfunktion genutzt. Dabei gibt es dieselben beiden Fälle wie bei der fDTL-Semantik, nur, dass bei dem ersten Fall noch zusätzlich mit der Hilfsfunktion nach dem letzten überschrittenen Wartepunkt, bei dem auf den entsprechenden Prozess gewartet wurde, gesucht wird. Entsprechend des gefundenen Wartepunktes wird ein Wert aus der neusten Nachricht von dem entfernten Prozess abgefragt. Ist noch keine Nachricht von dem entfernten Prozess angekommen oder gibt es noch keinen Wert für den gefundenen Wartepunkt in der Nachricht, so tritt der dritte Fall ein und es wird ? ausgegeben, da noch nichts über den Wahrheitswert bekannt ist.

Durch diese Definition des @-Operators können die entfernten Formeln nun von verschiedenen Punkten in dem Lauf des entfernten Prozesses an ausgewertet werden, je nachdem, wann nach dem Wahrheitswert einer entfernten Formel auf dem eigentlichen Prozess gefragt wird und welche Wartepunkte davor lagen.

Beispiel 3.18 (fSDTL)**.** *In diesem Beispiel sollen die Funktion und die Aufgaben des Schedulers und der Synchronisationsaktionen sowie die Auswertung einer fSDTL-Formel etwas verdeutlicht werden.*

Seien p und q Prozesse und m_p und m_q zugehörige Monitore und T^p und T^q die Transitionssysteme, die das Verhalten der Prozesse modellieren, $AP^p = \{a,b\} \cup X_p$ und $AP^q = \{c,d\} \cup X_q$ die Mengen der möglichen Propositionen der Prozesse, wobei X_p und X_q die entsprechenden Mengen der Nachrichten und Synchronisationsaktionen sind. Es soll nun die Formel $\varphi = @_p(\Box(a \rightarrow @_q(c \mathcal{U} d)))$ überwacht werden.

Wir betrachten dafür einen Scheduler mit der Ausführungsreihenfolge

$$s = q\ q\ p\ p\ p\ q\ q\ p\ p\ p\ \ldots$$

Als nächstes wird durch das Ausführen der Transitionssysteme T^p und T^q der Prozesse in der Reihenfolge, die s vorgibt, eine erlaubte Ausführung erzeugt. In diesem Beispiel wird davon ausgegangen, dass das System aktuell 10 Schritte gemacht hat und für diese 10 Schritte der Wahrheitswert überprüft werden soll. Das heißt, es entsteht durch das Ausführen der Transitionssysteme ein 10 elementiger, endlicher Präfix einer Ausführung w. Nehmen wir an, das Ausführen der Transitionssysteme ergab folgenden endlichen Präfix einer Ausführung:

$$w = \{c, \overset{1}{\uparrow_1^p}\}_q, \{d, \overset{2}{\lambda_{\uparrow 1}^p}\}_q, \{\overset{3}{a}\}_p, \{a, \overset{4}{\downarrow_1^q}\}_p, \{\overset{5}{\lambda_{\downarrow 1}^q}\}_p, \{c, \overset{6}{\lambda_{\uparrow 2}^p}\}_q, \{\overset{7}{\uparrow_2^p}\}_q, \{\overset{8}{\downarrow_2^q}\}_p, \{a, \overset{9}{\lambda_{\downarrow 2}^q}\}_p, \{\overset{10}{a}\}_p$$

Die Nummern bezeichnen den jeweiligen Schritt, um später besser auf diesen verweisen zu können. Über diesem endlichen Präfix einer Ausführung kann nun eindeutig mittels fSDTL ein Wahrheitswert für die Formel bestimmt werden.

Wie man sieht ist w ein erlaubter endlicher Präfix einer Ausführung, denn vor jedem Empfang kommt das Senden einer Nachricht und kein Prozess überschreitet einen Wartepunkt, bevor der jeweilige andere nicht den Sendepunkt erreicht hat.

Betrachten wir nun zuerst die Teilformel $@_q(c \mathcal{U} d)$, die entfernt von dem Monitor m_q ausgewertet wird. In Schritt 1 ergibt diese ? und für die Nachricht $\uparrow_1^p$ gilt damit $\uparrow_1^p (@_q(c \mathcal{U} d), 0) = ?$. In Schritt 2 gilt dann $\top$, sowohl wenn die Formel vom Anfang des Laufes an ausgewertet wird, als auch wenn sie vom ersten Sendepunkt an ausgewertet wird. In Schritt sechs ist der zweite Sendepunkt und durch das c gilt für $c \mathcal{U} d$ ab dem zweiten Sendepunkt ?. Im siebten Schritt gilt für die Formel

ab dem zweiten Sendepunkt dann $\bot$ und in der zweiten Nachricht steht insgesamt

$$\uparrow_2^p (@_q(c\,\mathcal{U}\,d), 0) = \top$$
$$\uparrow_2^p (@_q(c\,\mathcal{U}\,d), 1) = \top$$
$$\uparrow_2^p (@_q(c\,\mathcal{U}\,d), 2) = \bot$$

Als nächstes wird mit den vorherigen Erkenntnissen die gesamte Formel φ betrachtet, die von m_p ausgewertet wird. Direkt im ersten Schritt von Prozess p, Schritt 3, gilt ein a. Da noch keine Nachricht von Prozess q angekommen ist ergibt sich erstmal ?. In Schritt 4 kommt zwar eine Nachricht an, doch es gilt $\uparrow_1^p (@_q(c\,\mathcal{U}\,d), 0) = ?$ und daher auch insgesamt ?. In Schritt 5 gibt es einen Wartepunkt, der aber noch nicht überschritten ist und damit noch keine Auswirkung hat. In Schritt 8 kommt die zweite Nachricht von Prozess q an. Da in dieser $\uparrow_2^p (@_q(c\,\mathcal{U}\,d), 0) = \top$ steht, ergibt sich ab dort für die beiden as aus den Schritten 3 und 4 für die Implikation $\top$ und damit insgesamt zu diesem Zeitpunkt ?. In Schritt 9 tritt wieder ein a auf. Da dieses nach dem ersten Wartepunkt aus Schritt 5 auftritt, muss nun aus der Nachricht $\uparrow_2^p$ der Wert für die Formel $@_q(c\,\mathcal{U}\,d)$ ab dem ersten Sendepunkt betrachtet werden. Da $\uparrow_2^p (@_q(c\,\mathcal{U}\,d), 1) = \top$ gilt, ergibt auch für dieses a die Implikation $\top$. In Schritt 10 tritt dann wieder ein a auf, allerdings war in Schritt 9 ein weiterer Wartepunkt. Deshalb wird nun $\uparrow_2^p (@_q(c\,\mathcal{U}\,d), 2)$ betrachtet. Da in der Nachricht an dieser Stelle $\bot$ steht, ergibt sich auch für die Implikation $\bot$. Somit gilt $[\![w \models @_p(\Box(a \rightarrow @_q(c\,\mathcal{U}\,d)))]\!]_{fSDTL} = \bot$.

Im nächsten Teil dieser Arbeit wird fSDTL analysiert und mit anderen Logiken verglichen, um zu zeigen, was mit der neuen Logik erreicht wurde.

3.7 Analysen und Vergleiche

In diesem Abschnitt sollen die verschiedenen Eigenschaften von fSDTL und deren Auswirkungen erläutert werden. Unter anderem werden die Scheduler und Transitionssysteme, die Mächtigkeit, Impartiality und Anticipation und die Synchronisationsmöglichkeiten betrachtet. Dabei wird gezeigt, dass die Logik über einem gewünschten Modell definiert ist, alle geforderten Anforderungen erfüllt und wie sie sich zu anderen Logiken, wie LTL und einer Logik für synchrone, verteilte Systeme aus [BF12] verhält.

3.7.1 Modell für eine Ausführung

Zuerst wird das fSDTL zugrunde liegende Modell einer Ausführung des Systems betrachtet. Wie in den Anforderungen erläutert wurde, ist ein solches Modell wichtig.

Das gewünschte Modell besteht aus der Modellierung der Prozesse als Transitionssysteme, dem Architekturmodell und den Schedulern, aus denen dann eine Ausführung resultiert. Die Transitionssysteme beschreiben das Verhalten der Prozesse und geben auch deren Nachrichtenkommunikation und Synchronisierungsaktionen an. Eine Ausführungsreihenfolge eines Schedulers gibt

genau eine mögliche zeitliche Abfolge der Schritte der Prozesse an. Durch das Ausführen der Transitionssysteme der Prozesse in der Reihenfolge, die die Ausführungsreihenfolge angibt, erhält man die Ausführung. Diese Ausführung gibt den Lauf des Gesamtsystems an. Für eine derartige Darstellung einer Ausführung des Systems kann mit fSDTL ein eindeutiger Wahrheitswert für eine Formel gefunden werden. Insgesamt wurde also ein Modell für das System gefunden, welches alle wichtigen Aspekte eindeutig modelliert.

3.7.2 Synchronisation

Als nächstes werden die Möglichkeiten zur Synchronisation von fSDTL betrachtet. Im Gegensatz zu ptDTL und fDTL gibt es nun Synchronisationsaktionen.

Dadurch müssen Prozesse auf andere warten und ihre Monitore haben ein gewisses Wissen darüber, wie weit die anderen Prozesse sind, da ein Prozess einen Wartepunkt nicht überschreiten darf bis der entsprechende Sendepunkt von dem anderen Prozess erreicht wurde und dadurch die Synchronisationsnachricht ankommt.

Außerdem ermöglichen die Synchronisationsaktionen, dass entfernte Formeln ab diesen erneut ausgewertet werden können, was die neue Anforderung aus Unterabschnitt 3.2.5 auf Seite 40 erfüllt. Sendepunkte entstehen auf dem entfernten Prozess und können daher nicht verzögert bei dem entfernten Prozess ankommen wie Nachrichten, die von einem Prozess an den entfernten Prozess gesendet werden, damit dieser mit einer neuen Auswertung der Formel beginnt. Aus diesem Grund sind Sendepunkte gute Möglichkeiten, um Formeln neu auszuwerten. Dieses Verfahren ist natürlich nicht optimal, da es im besten Fall so wäre, dass beispielsweise bei einer Formel $@_p\Box(a \rightarrow @_q\Box b)$ das $\Box b$ genau ab dem Zeitpunkt auf Prozess q ausgewertet werden müsste, an dem auf Prozess p das a gegolten hat. Dies ist aber in einem asynchronen, verteilten System nicht möglich, zum Einen aufgrund der Nachrichten, wie es vorher schon einmal beschrieben wurde und zum Anderen, weil ein Sendepunkt eines entfernten Prozesses, nachdem ein Prozess den entsprechenden Wartepunkt überschritten hat, im Allgemeinen schon in der Vergangenheit liegt.

3.7.3 Verhalten für lokale Formeln

In diesem Abschnitt wird gezeigt, dass das Ergebnis der Auswertung von lokalen fSDTL-Formeln, also fSDTL-Formeln ohne entfernte Teilformel, mit der fSDTL-Semantik wie auch bei fDTL äquivalent zu dem Ergebnis der Auswertung mit der LTL_3-Semantik ist. Dafür wird in dem Beweis gezeigt, dass die $fDTL_\omega$- und die $fSDTL_\omega$-Semantik für lokale Formeln äquivalent sind.

Theorem 3.19 (Lokale fSDTL-Formel)**.** *Sei p ein Prozess, w ein endlicher Präfix einer erlaubten Ausführung einer Ausführungsreihenfolge eines Schedulers, w^p der endliche Präfix des Laufes von p, der sich aus w ergibt und $@_p\psi_1$ eine fSDTL-Formel, wobei ψ_1 keinen @-Operator enthält. Dann gilt $[\![w \models @_p\psi_1]\!]_{fSDTL} = [\![w^p \models \psi_1]\!]_{LTL_3}$.*

Beweis. Da die Formel $@_p\psi_1$ keinen @-Operator in dem ψ_1 enthält, wird der Beweis durchgeführt, indem gezeigt wird, dass fSDTL$_\omega$ äquivalent zu fDTL$_\omega$ ist, wenn ψ_1 keinen @-Operator enthält. Da in Theorem 3.7 auf Seite 35 gezeigt wurde, dass $[\![w \models @_p\psi_1]\!]_{\text{fDTL}} = [\![w \models \psi_1]\!]_{\text{LTL}_3}$ gilt, wenn ψ_1 keinen @-Operator enthält, indem $\forall v \in \Sigma^\omega : [\![v \models \psi_1]\!]_{\text{LTL}} = [\![v,0 \models \psi_1]\!]_{\text{fDTL}_\omega}$ gezeigt wurde, folgt daher dann auch $[\![w \models @_p\psi_1]\!]_{\text{fSDTL}} = [\![w^p \models \psi_1]\!]_{\text{LTL}_3}$, wenn ψ_1 keinen @-Operator enthält.

Sei AP^p die Menge atomarer Propositionen von Prozess p und $w^p \in \Sigma^{p\omega}$ dessen gesamter Lauf. Sei w die Ausführung des gesamten Systems und $s^p \subseteq \mathbb{N}$ wie gewohnt die Menge der Stellen, an denen p in w einen Schritt macht.

Betrachtet man die atomaren Formel true und die Operatoren $\neg$, $\vee$ und $\mathcal{U}$, so sieht man, dass diese in der Semantik von fDTL$_\omega$ und in der Semantik von fSDTL$_\omega$ gleich definiert sind. Zu betrachten sind also noch die atomaren Propositionen sowie der Next-Operator ($\bigcirc$).

Sei im Folgenden die Funktion $min_x : 2^{\mathbb{N}} \to \mathbb{N}$ wie folgt definiert:

$$min_x(M) = \begin{cases} -1 & \text{wenn } x < 0 \\ min(M,x) & \text{sonst} \end{cases}$$

Wobei für $min : 2^{\mathbb{N}} \times \mathbb{Z} \to \mathbb{N}$ gilt:

$$min(M,x) = n \Leftrightarrow n \in M \wedge \exists M' \subseteq M : |M'| = x \wedge \forall m \in M' : m < n \wedge \forall m' \in M \setminus (M' \cup \{n\}) : n < m'$$

$min_x(M)$ gibt aus einer Menge M natürlicher Zahlen das Element zurück, für das es genau x kleinere Elemente in der Menge gibt.

Mit Hilfe dieser Funktion wird nun gezeigt, wie weit die Ausführung im Verhältnis zu dem Lauf von w^p sein kann. Durch diese Erkenntnisse können die Definitionen der Operatoren in fSDTL auf die in fDTL reduziert werden.

Sei nun $k \in \mathbb{N}$ die nächste zu betrachtende Stelle von dem Lauf w^p. Für i, die nächste, zu betrachtende Stelle der Ausführung w muss daher $min_{k-1}(s^p) < i \leq min_k(s^p)$ gelten. Der Grund dafür ist, dass, wenn als nächstes die Stelle k von w^p betrachtet werden muss, das Gesamtsystem schon über die Stelle $min_{k-1}(s^p)$ von w, welche der Stelle $k-1$ von w^p entspricht, hinaus sein muss, aber $min_k(s^p)$, welches der Stelle k von w^p entspricht, noch nicht überschritten haben kann. Deswegen kann aus $\exists j \geq i : j \in s^p \wedge \nexists i \leq h < j : h \in s^p$ aus der Definition der Propositionen in fSDTL nur ein j mit $w_j = w^p_k$ folgen. Damit folgt für eine atomare Proposition $a \in AP^p$:

$$\begin{aligned} [\![w,i \models a]\!]_{\text{fSDTL}_\omega} &= \begin{cases} \top & \text{wenn } \exists j \geq i : j \in s^p \wedge a \in w_j \wedge \nexists i \leq h < j : h \in s^p \\ \bot & \text{sonst} \end{cases} \\ &= \begin{cases} \top & \text{wenn } a \in w^p_k \\ \bot & \text{sonst} \end{cases} \\ &= [\![w^p,k \models a]\!]_{\text{fDTL}_\omega} \end{aligned}$$

Mit derselben Begründung wie vorher kann aus $\exists j > i : j \in s^p \wedge \exists! i \leq h < j : h \in s^p$ aus der Definition des Next-Operators aus der Definition der fSDTL-Semantik nur ein j mit $w_j = w^p_{k+1}$ folgen. Das liegt daran, dass es genau ein h vor j geben soll, das auch in s^p ist. Daher wäre $w_h = w^p_k$ und j wäre dann die Stelle des nächsten Schrittes von p in w, also die Stelle $k+1$ von w^p. Für den Next-Operator folgt dann

$$
\begin{aligned}
[\![w_i \models \bigcirc \varphi]\!]_{\text{fSDTL}_\omega} &= \begin{cases} \top & \text{wenn } \exists j > i : j \in s^p \wedge \exists! i \leq h < j : h \in s^p \wedge \\ & \qquad [\![w_j \models \varphi]\!]_{\text{fSDTL}_\omega} = \top \\ \bot & \text{wenn } \exists j > i : j \in s^p \wedge \exists! i \leq h < j : h \in s^p \wedge \\ & \qquad [\![w_j \models \varphi]\!]_{\text{fSDTL}_\omega} = \bot \\ ? & \text{sonst} \end{cases} \\
&= \begin{cases} \top & \text{wenn } [\![w^p_{k+1} \models \varphi]\!]_{\text{fDTL}_\omega} = \top \\ \bot & \text{wenn } [\![w^p_{k+1} \models \varphi]\!]_{\text{fDTL}_\omega} = \bot \\ ? & \text{sonst} \end{cases} \\
&= [\![w^p_{k+1} \models \bigcirc \varphi]\!]_{\text{fDTL}_\omega}
\end{aligned}
$$

□

3.7.4 Eigenschaften der Semantik

Da die fDTL-Semantik impartial ist, folgt bei fSDTL die Impartiality für Formeln ohne entfernte Teilformeln aus Theorem 3.7 auf Seite 35, denn danach ist fSDTL für diese Formeln äquivalent zu fDTL. Die fSDTL-Semantik besitzt aber auch für Formeln mit entfernten Teilformeln Impartiality, denn eine entfernte Teilformel kann in fSDTL durch die Synchronisationsaktionen wie eine Proposition angesehen werden, die sich erst verspätet für eine endgültigen Wert entscheidet. Aus diesem Grund ist die fSDTL-Semantik impartial.

Auch Anticipation ist in der fSDTL-Semantik vorhanden, allerdings wie bei fDTL wieder nur lokal. Für lokal auswertbare Formeln folgt die Anticipation wie auch die Impartiality aus Theorem 3.7 auf Seite 35, da sie auch in fDTL vorhanden ist. Auch in Formeln mit entfernten Teilformeln ist Anticipation vorhanden, weil entfernte Teilformeln sich ähnlich wie Propositionen verhalten und endgültige Wahrheitswerte aus den Nachrichten annehmen, sobald diese bekannt sind. Global ist Anticipation auch in fSDTL, wie in Unterabschnitt 3.2.4 auf Seite 39 beschrieben, nicht möglich.

Mit ptDTL sind nur Safety- oder Guarantee-Eigenschaften monitorbar gewesen. Dies wurde in fSDTL deutlich verbessert. Die Logik besitzt nun auf den einzelnen Prozessen die Mächtigkeit von LTL_3, wie in Theorem 3.19 auf Seite 53 gezeigt wurde, sowie eine dreiwertige Semantik mit Impartiality. Wie in Unterabschnitt 2.4.1 auf Seite 14 beschrieben sind damit die Eigenschaften aus Safety und Guarantee zugleich sowie die gesamte Klasse Obligation und Teile von Response und Persistence monitorbar.

Dadurch ist es nun möglich in dem überwachten System Eigenschaften zu testen, die vorher nicht überwacht werden konnten.

3.7.5 Mächtigkeit

In diesem Abschnitt folgt eine Betrachtung der Mächtigkeit von fSDTL. Es wird zuerst die Mächtigkeit im Verhältnis zu fDTL betrachtet. Dabei ist es vor allem wichtig zu zeigen, dass jede fDTL-Formel auch als fSDTL-Formel dargestellt werden kann, denn dadurch wird gezeigt, dass das Hinzufügen der Synchronisationsaktionen nur positive Effekte hat und nicht zu einem Verlust an Mächtigkeit führt. Als zweites wird dann noch die Mächtigkeit von ptDTL und fSDTL verglichen.

Für den Beweis, dass fSDTL mächtiger ist als fDTL, ergibt sich aber das Problem, dass der @-Operator in fSDTL Synchronisationsaktionen benutzt und in fDTL nicht, wodurch für fSDTL eine Möglichkeit gefunden werden muss, dass der @-Operator genutzt wird und die Synchronisationsaktionen nicht beachtet werden. Stattdessen muss immer das Ergebnis der Auswertung der entfernten Formel vom Beginn des Laufes des entfernten Prozesses an genommen werden. Dies ist genau dann der Fall, wenn eine entfernte Teilformeln in fSDTL von keinem temporalen Operator umgeben ist, was in dem folgenden Beweis ausgenutzt wird.

Theorem 3.20 (Mächtigkeit von fSDTL bezüglich fDTL)**.** *fSDTL ist mächtiger als fDTL, es gilt daher fDTL* $\subset$ *fSDTL.*

Beweis. Um fDTL $\subseteq$ fSDTL zu zeigen, wird zuerst gezeigt, dass alle Operatoren außer der @-Operator in fSDTL_ω und fDTL_ω äquivalent sind. Danach wird gezeigt, dass es für jede fDTL-Formel eine äquivalente fSDTL-Formel gibt.

Seien dazu im Folgenden p und q Prozesse aus der Menge P aller Prozesse und $a \in AP^p$ eine atomare Proposition des Prozesses p. Die atomaren Formel true und die Operatoren $\neg$, $\vee$ und $\mathcal{U}$ sind in den Semantiken von fSDTL_ω und fDTL_ω gleich definiert. Für eine atomare Proposition a und den Next-Operator folgt die Äquivalenz aus Theorem 3.19 auf Seite 53.

Sei $@_p\varphi$ eine beliebige fDTL-Formel. Nun gibt es zwei Fälle:

- φ enthält keinen @-Operator:

  In diesem Fall ist $@_p\varphi$ bereits eine fSDTL-Formel und die Äquivalenz folgt aus Theorem 3.19 auf Seite 53.

- φ enthält Teilformeln der Form $@_q\psi$:

  In diesem Fall kann $@_p\varphi$ sowie alle entfernten Teilformeln von $@_p\varphi$ nach Theorem 3.5 auf Seite 30 in @-DNF umgewandelt werden, so dass eine Formel $@_p\varphi'$ entsteht. Dadurch wird erreicht, dass es kein @ innerhalb eines temporalen Operators in $@_p\varphi'$ oder in einer entfernten Teilformeln der Formel gibt.

$@_p\varphi'$ ist über der fDTL-Semantik und der fSDTL-Semantik äquivalent, da die Funktion des @-Operators in fDTL und fSDTL identisch ist, wenn das @ vom ersten Zeichen an ausgewertet wird, da dort noch keine Wartepunkte überschritten worden sein können. Daher wird auch in fSDTL immer der Wahrheitswert der Auswertung der entfernten Formel von Beginn des Laufes des entfernten Prozesses an abgefragt.

Dadurch wurde gezeigt, dass jede fDTL-Formel in eine äquivalente Formel umgewandelt werden kann, die in fSDTL äquivalent ausgewertet wird. Daher sind in fSDTL alle Eigenschaften darstellbar, die in fDTL ausgedrückt werden können und daher gilt fDTL $\subseteq$ fSDTL.

fDTL $\subset$ fSDTL wird gezeigt, indem eine Eigenschaft angegeben wird, die durch fSDTL besser angenähert werden kann als durch fDTL.

Sei die betrachtete Eigenschaft „Immer, wenn auf Prozess p die Proposition a gilt, so muss ab diesem Zeitpunkt auf dem Prozess q so lange die Proposition b gelten, bis einmal die Proposition c gilt.“. Das Wichtige daran ist das „ab diesem Zeitpunkt“. Dies kann in beiden Logiken nicht genau abgebildet werden, weil das System asynchron ist. In fSDTL wird dies angenähert, indem die entfernte Teilformel auf Prozess q von jedem Sendepunkt an erneut ausgewertet wird. Wenn dann ein a auf Prozess p auftritt, so wird abhängig von dem letzten Wartepunkt der Wahrheitswert ab einem Sendepunkt abgefragt. Dadurch wird die entfernte Formel „ungefähr ab diesem Zeitpunkt“ auf Prozess q ausgewertet. In fSDTL ist dies die Formel $@_p\Box(a \rightarrow @_q(b\,\mathcal{U}\,c))$. In fDTL kann das „ab diesem Zeitpunkt“ nicht dargestellt werden, weil alle entfernten Formeln immer vom Beginn des Laufes des entfernten Prozesses an ausgewertet werden. Aus diesem Grund wird die entfernte Formel nie neu ausgewertet. □

Als nächstes wird bewiesen, dass fSDTL eine größere Mächtigkeit hat als ptDTL. Dies folgt aus dem vorherigen Theorem.

Theorem 3.21 (Mächtigkeit von fSDTL bezüglich ptDTL)**.** *fSDTL ist mächtiger als ptDTL, es gilt daher ptDTL $\subset$ fSDTL, wenn der @-Operator nicht genauer betrachtet wird.*

Beweis. In Theorem 3.8 auf Seite 38 wurde gezeigt, dass fDTL eine größere Mächtigkeit besitzt als ptDTL, wenn der @-Operator nicht genauer betrachtet wird. Da nun in Theorem 3.20 auf der vorherigen Seite gezeigt wurde, dass fDTL $\subset$ fSDTL gilt, folgt damit auch ptDTL $\subset$ fSDTL, wenn der @-Operator nicht genauer betrachtet wird. □

Der @-Operator kann für die Betrachtung der Mächtigkeit nicht genauer analysiert werden. Zwar besitzt dieser in fSDTL nun Synchronisationsaktionen und die entfernten Formeln können daher auch auf den entfernten Prozessen von einem späteren Punkt an ausgewertet werden, allerdings funktioniert er doch grundsätzlich anders als in ptDTL. Im Gegensatz zu ptDTL, wo für eine entfernte Formel immer der gesamte Lauf von hinten mit einer zweiwertigen Semantik ohne Impartiality betrachtet wird, wird in fSDTL der gesamte, entfernte Lauf von vorne und ab Synchronisationsaktionen mit einer dreiwertigen Semantik mit Impartiality und Anticipation betrachtet und der @-Operator wartet auf Wahrheitswerte in den Nachrichten, die impartial sind. Die Werte einer entfernten Formel in ptDTL sind damit deutlich stärker abhängig von dem Versandzeitpunkt der

Nachrichten als in fSDTL. Wie gezeigt wurde sind aber die lokalen Operatoren in fSDTL mächtiger als in ptDTL, wodurch für die lokalen Teile mehr Eigenschaften spezifiziert werden können.

3.7.6 Vergleich zu anderen Logiken

Während der Betrachtung wurde fSDTL mit drei anderen Logiken verglichen, nämlich mit LTL_3, ptDTL und fDTL. Wie gezeigt wurde entsprechen lokale fSDTL-Formeln LTL_3 und fSDTL ist ohne genauere Betrachtung des @-Operators mächtiger und besitzt bessere Eigenschaften als ptDTL. Außerdem ist fSDTL mächtiger als fDTL.

Als letztes soll fSDTL noch mit einer Logik für Monitoring in einem synchronen, verteilten System verglichen werden. In [BF12] wurde auf Basis von LTL_3 eine Auswertungsfunktion für Formeln gegeben, die sich in einem synchronen, verteilten System auf mehrere Monitore beziehen. Das heißt, eine Formel enthält im Allgemeinen Propositionen mehrerer Prozesse. Dabei bekommt jeder Monitor jede Formel und wertet in jedem synchronen Schritt das aus, was ihm in der Formel möglich ist. Danach senden alle ihre, teilweise ausgewerteten, Formeln weiter an Monitore anderer Prozesse und es wird wieder ausgewertet, was möglich ist. Wenn eine Formel an einen anderen Monitor gesendet wird werden Previous-Operatoren in die Formeln eingefügt, so, dass der nächste Monitor weiß, dass etwas in dem vorherigen Schritt gelten sollte. Dies geschieht so lange, bis ein eindeutiger Wert einer Formel gefunden wurde.

Vergleicht man diese Art der Auswertung mit der aus fSDTL, so wird schnell klar, dass sie aufgrund des synchronen Systems deutlich mächtiger ist. Es kann exakt beschrieben werden, sowohl lokal als auch entfernt, in welchem Schritt eines Prozesses eine Formel gelten sollte, da in einem synchronen System ein Next- bzw. ein Previous-Operator ausdrückt, dass in dem nächsten bzw. vorherigen Schritt des Gesamtsystems etwas gelten soll. Dadurch sind Eigenschaften eines Systems noch viel genauer spezifizierbar als mit fSDTL und dessen Synchronisationsaktionen.

4 Automatenmodell

In diesem Kapitel wird ein Automatenmodell für die im letzten Kapitel definierte Logik fSDTL entwickelt. Weiter wird eine Umwandlung für eine fSDTL-Formel in dieses Automatenmodell erstellt.

Sowohl die Automaten als auch die Umwandlung basieren auf den Ideen für die Monitorgenerierung für LTL_3 aus [BLS11], da fSDTL Ähnlichkeiten zu LTL_3 aufweist. Im Gegensatz zu LTL_3 gibt es in fSDTL zusätzlich den @-Operator sowie die dazugehörigen Synchronisationsaktionen und Nachrichten, welche zusätzlich in dem Automatenmodell abgebildet werden müssen. Die Automaten werden über einem endlichen Präfix einer Ausführung ausgeführt und ergeben dann den Wert, der sich auch bei einer Auswertung mit fSDTL ergeben würde. Da fSDTL auf einer dreiwertigen Logik über unendlichen Worten basiert, sind auch die Automaten dreiwertig, besitzen also drei verschiedene Arten von Zuständen.

Des Weiteren muss das Automatenmodell für eine Formel das Verhalten mehrerer Prozesse gleichzeitig simulieren, nicht nur das eines Prozesses. Dies kommt durch das verteilte System und den deswegen vorhandenen @-Operator, durch den in einer einzigen fSDTL-Formel mehrere Prozesse involviert sein können. Neben der Gesamtformel, die von einem Monitor eines Prozesses ausgewertet wird, müssen die entfernten Teilformeln von anderen Monitoren auf den entsprechenden Prozessen ausgewertet werden. Aus diesem Grund müssen auch diese anderen Monitore durch das Automatenmodell simuliert werden, um die Wahrheitswerte in den Nachrichten zu erhalten.

Das heißt, dass das Automatenmodell aus mehreren Automaten besteht, von denen jeder ein Monitor auf einem Prozess ist. Ein Automat macht genau dann einen Schritt, wenn der zugehörige Prozess in dem Präfix der Ausführung einen Schritt macht.

Zuerst werden dafür die verschiedenen, für die Umwandlung benötigten, Automatentypen definiert.

4.1 Endlich und unendlich große Automaten

In diesem Abschnitt werden die Automatentypen definiert, die als Zwischenschritte für die Umwandlung der Formel in das gewünschte Modell benötigt werden. Diese sind, bis auf die Mächtigkeit der Menge der Zustände, identisch mit denen, die in [BLS11] verwendet werden.

Automaten mit einer unendlich großen Zustandsmenge und Menge von akzeptierenden Zuständen sind echt mächtiger als Automaten mit endlich großen Mengen. Die Mächtigkeit dieser Automaten

wird allerdings nicht ausgeschöpft, denn die, für das Automatenmodell in der Umwandlung entstehenden, Automaten besitzen eine ganz spezielle Struktur aus identischen, endlichen Elementen, wie am Ende der Umwandlung gezeigt wird. Des Weiteren wird in Abschnitt 4.4 auf Seite 93 gezeigt, dass die entstehenden Automaten auf endliche Größe beschränkt werden können.

Als erstes werden Automaten über endlichen Worten betrachtet. Dazu gehören deterministische und nicht-deterministische Automaten, die im Folgenden definiert werden. Die gleichen Automatentypen mit endlich vielen und mit unendlich vielen Zuständen werden jeweils zusammen in einer Definition behandelt.

Definition 4.1 (Deterministischer (un-)endlicher Automat)**.** *Ein deterministischer (un-)endlicher Automat (determinstic (in-)finite automaton, DFA (DIA)) $A = (\Sigma, Q, q_0, \delta, F)$ ist ein 5-Tupel mit*

- *einem unendlichen Alphabet Σ,*
- *einer (un-)endlichen Menge von Zuständen Q,*
- *einem Startzustand $q_0 \in Q$,*
- *einer partiellen Transitionsfunktion $\delta : Q \times \Sigma \to Q$ und*
- *einer (un-)endlichen Menge von akzeptierenden Zuständen $F \subseteq Q$.*

Sei $w \in \Sigma^*$. Der Lauf eines DFAs bzw. eines DIAs $A = (\Sigma, Q, q_0, \delta, F)$ bei Eingabe des endlichen Wortes w ist eine Funktion $\rho : \{0, 1, \ldots, |w|\} \to Q$, so dass

- $\rho(0) = q_0$ *und*
- $\forall i \in \{0, 1, \ldots, |w| - 1\} : \rho(i+1) = \delta(\rho(i), w_i)$.

Sei $\mathcal{L}(A)$ die akzeptierte Sprache von A. Es gilt $w \in \mathcal{L}(A)$ gdw. $\rho(|w|) \in F$.

Als nächstes wird die Definition der nicht-deterministischen, (un-)endlichen Automaten gegeben. Diese ist ähnlich wie die der deterministischen, (un-)endlichen Automaten.

Definition 4.2 (Nicht-deterministischer (un-)endlicher Automat)**.** *Ein nicht-deterministischer (un-)endlicher Automat (non-determinstic (in-)finite automaton, NFA (NIA) $A = (\Sigma, Q, Q_0, \delta, F)$ ist ein 5-Tupel mit*

- *einem unendlichen Alphabet Σ,*
- *einer (un-)endlichen Menge von Zuständen Q,*
- *einer endlichen Menge von Startzuständen $Q_0 \subseteq Q$,*
- *einer partiellen Transitionsfunktion $\delta : Q \times \Sigma \to 2^Q$ und*
- *einer (un-)endlichen Menge von akzeptierenden Zuständen $F \subseteq Q$.*

Sei $w \in \Sigma^*$. Der Lauf eines NFAs bzw. eines NIAs $A = (\Sigma, Q, Q_0, \delta, F)$ bei Eingabe des endlichen Wortes w ist eine Funktion $\rho : \{0, 1, \ldots, |w|\} \to 2^Q$, so dass

- $\rho(0) = Q_0$ *und*
- $\forall i \in \{0,1,\ldots,|w|-1\} : \rho(i+1) = \bigcup_{q \in \rho(i)} \delta(q, w_i)$.

Sei $\mathcal{L}(A)$ die akzeptierte Sprache von A. Es gilt $w \in \mathcal{L}(A)$ gdw. $\exists q \in \rho(|w|) : q \in F$.

Neben den Automaten auf endlichen Worten werden auch noch Automaten auf unendlichen Worten für die Umwandlung benötigt. Für diese werden als nächstes Büchi-Automaten definiert. Dabei werden zuerst nicht-deterministische und dann alternierende Büchi-Automaten betrachtet.

Definition 4.3 (Nicht-deterministischer Büchi-Automat)**.** *Ein (unendlicher) nicht-deterministischer Büchi-Automat ((Infinite) non-determinstic Büchi-automaton, (I)NBA) $A = (\Sigma, Q, Q_0, \delta, F)$ ist ein 5-Tupel mit*

- *einem unendlichen Alphabet Σ,*
- *einer (un-)endlichen Menge von Zuständen Q,*
- *einer endlichen Menge von Startzuständen $Q_0 \subseteq Q$,*
- *einer partiellen Transitionsfunktion $\delta : Q \times \Sigma \to 2^Q$ und*
- *einer (un-)endlichen Menge von akzeptierenden Zuständen $F \subseteq Q$.*

Sei $w \in \Sigma^\omega$. Der Lauf eines (I)NBAs $A = (\Sigma, Q, Q_0, \delta, F)$ bei Eingabe des unendlichen Wortes w ist eine Funktion $\rho : \mathbb{N} \to Q$, so dass

- $\rho(0) = Q_0$ *und*
- $\forall i \in \mathbb{N} : \rho(i+1) = \bigcup_{q \in \rho(i)} \delta(\rho(i), w_i)$.

Sei $\mathcal{L}(A)$ die akzeptierte Sprache von A. Es gilt $w \in \mathcal{L}(A)$ gdw. $|\{k \in \mathbb{N} \mid \rho(k) \cap F \neq \emptyset\}| = \infty$.

Die Akzeptanzbedingung wurde anders aufgeschrieben, als es normalerweise für endliche große NBAs der Fall ist. Die Akzeptanzbedingung sagt aus, dass unendlich viele akzeptierende Zustände durchlaufen werden müssen. Da ein endlich großer NBA nur endlich viele Zustände besitzt, ist es äquivalent, ob unendlich viele akzeptierende Zustände oder ein akzeptierender Zustand unendlich oft, was im Allgemeinen die Formulierung der Akzeptanzbedingung für endlich große NBAs ist, betreten werden soll. Für unendlich große NBAs, also INBAs, ist dies hingegen ein großer Unterschied, da es dort, zum Beispiel, eine Kette aus unendlich vielen, akzeptierenden Zuständen geben kann. Aus diesem Grund ist die hier benutzte Akzeptanzbedingung für unendlich große INBAs mächtiger als die, welche normalerweise für NBAs genutzt wird. Diese Akzeptanzbedingung wurde gewählt, weil sie für die Umwandlung so benötigt wird.

Nach den nicht-deterministischen Büchi-Automaten folgt nun noch die Definition für die alternierenden Büchi-Automaten. Zuerst wird aber die Menge der positiven booleschen Kombinationen definiert, da diese für die Transitionsfunktion und die Startzustände der alternierenden Automaten benötigt wird.

Definition 4.4 (Menge der positiven booleschen Kombinationen)**.** *Sei Q eine Menge aus booleschen Variablen. Die Menge der positiven booleschen Kombinationen $\mathbb{B}^+(Q)$ der Menge Q ist dann die kleinste Menge, für die gilt:*

- $Q \subseteq \mathbb{B}^+(Q)$,
- $x, y \in \mathbb{B}^+(Q) \Rightarrow x \vee y, x \wedge y \in \mathbb{B}^+(Q)$ *und*
- $\mathit{true}, \mathit{false} \in \mathbb{B}^+(Q)$.

Eine Menge von booleschen Variablen $S \subseteq Q$ ist ein Modell für eine Formel $\varphi \in \mathbb{B}^+(Q)$, geschrieben $S \models \varphi$, wenn alle $s \in S$ als true und alle $s \in Q \setminus S$ als false angenommen werden und φ dann in der Aussagenlogik zu true äquivalent ist. Eine Menge von booleschen Variablen $S \subseteq Q$ ist ein minimales Modell für eine Formel $\varphi \in \mathbb{B}^+(Q)$, geschrieben $S \models\!\!\!\models \varphi$, wenn $S \models \varphi$ gilt und es keine Menge von booleschen Variablen $S' \subset S$ gibt, für die $S' \models \varphi$ gilt.

Die Menge der positiven booleschen Kombinationen enthält also zum Einen die Elemente der Menge, über der sie gebildet wird und zum Anderen alle Verundungen und Veroderungen zweier eigener Elemente.

Definition 4.5 (Alternierender Büchi-Automat)**.** *Ein (unendlicher) alternierender Büchi-Automat ((Infinite) alternating Büchi-automaton, (I)ABA) $A = (\Sigma, Q, Q_0, \delta, F)$ ist ein 5-Tupel mit*

- *einem unendlichen Alphabet Σ,*
- *einer (un-)endlichen Menge von Zuständen Q,*
- *einer positiven booleschen Formel von Startzuständen $Q_0 \in \mathbb{B}^+(Q)$,*
- *einer alternierenden Transitionsfunktion $\delta : Q \times \Sigma \to \mathbb{B}^+(Q)$ und*
- *einer (un-)endlichen Menge von akzeptierenden Zuständen $F \subseteq Q$.*

Sei $w \in \Sigma^\omega$. Der Lauf eines (I)ABAs $A = (\Sigma, Q, q_0, \delta, F)$ bei Eingabe des unendlichen Wortes w ist ein Q-beschrifteter gerichteter azyklischer Graph (V, E), so dass es eine Beschriftung $l : V \to Q$ und eine Funktion $h : \mathbb{N} \to V$, welche alle Knoten auf einer Höhe zurückgibt, existieren, die folgende Eigenschaften erfüllen:

- $\{l(v) \mid v \in h(0)\} \models\!\!\!\models Q_0$,
- $E \subseteq \bigcup_{i \in \mathbb{N}} (h(i) \times h(i+1))$,
- $\forall v' \in V : h^{-1}(v') \geq 1 \to \{v \mid v \in V \wedge (v, v') \in E\} \neq \emptyset$,
- $\forall v, v' \in V : v \neq v' \wedge l(v) = l(v') \Rightarrow h^{-1}(v) \neq h^{-1}(v')$ *und*
- $\forall v \in V : \{l(v') \mid (v, v') \in E\} \models\!\!\!\models \delta(l(v), w_{h^{-1}(v)})$.

Sei $\mathcal{L}(A)$ die akzeptierte Sprache des (I)ABAs A. Es gilt $w \in \mathcal{L}(A)$ gdw. es einen Lauf (V, E) von A für das Wort w gibt, in dem es in jedem unendlichen Pfad des Laufes unendlich viele Knoten $v \in V$ mit $l(v) \in F$ gibt.

Die Definition des Laufes eines (I)ABAs beschreibt einen Graphen, der durch die erste Bedingung in einem minimalen Modell für die Startzustandsformel des (I)ABAs beginnt. Die anderen vier Bedingungen beschreiben, dass es für jedes Zeichen das gelesen wird eine Ebene in dem Graphen gibt. In dieser befinden sich die aktuellen Zustände, die in der Vereinigung jeweils eines minimalen Modelles für die positiven booleschen Kombinationen sind, die sich durch das Lesen des Zeichens aus den Zuständen der vorherigen Ebene ergeben. Da das Wort unendlich lang ist sind auch die Pfade entweder unendlich lang oder enden nach endlicher Zeit, indem eine entstehende boolesche Kombination äquivalent zu true ist.

4.2 Das Modell

In diesem Abschnitt sollen nun die Automaten definiert werden, welche das angestrebte Automatenmodell für eine fSDTL-Formel bilden.

Aufgrund der vorher beschriebenen Anforderungen an das Modell, bietet es sich an, das Modell einer Moore- oder einer Mealy-Maschine als Grundlage zu nehmen. In dieser Arbeit werden Moore-Maschinen benutzt. Das gesamte Automatenmodell enthält dann für jeden benötigten Monitor eine Moore-Maschine und in den Zuständen der Moore-Maschinen sich die aktuellen Wahrheitswerte als aktuelle Ausgabe.

Als nächstes ergibt sich das Problem, dass jede Moore-Maschine neben den Operatoren, die es in LTL gibt, auch @-Operatoren auswerten muss. Dessen Wahrheitswerte sind allerdings nicht anhand der Schritte des Prozesses erkennbar, denn dort sind nur die Empfangszeitpunkte der Nachrichten vermerkt, nicht deren Inhalte. Das heißt, dass diese auf irgendeine Art und Weise aus dem Speicher des, zu dem Monitor gehörenden, Prozesses ausgelesen werden müssen. Wie dies genau funktioniert wird im nächsten Abschnitt in der Transitionsfunktion der Automaten beschrieben. Ein @-Operator in einer Formel kann ähnlich wie eine Proposition von dem Automaten behandelt werden und stellt daher keine weiteren Anforderungen an das Modell.

Außerdem müssen auch die Synchronisationsaktionen berücksichtigt werden. Das Auftreten dieser muss sich in dem Automaten gemerkt werden, damit man weiß, welcher Wert für ein @ benötigt wird und um anzugeben, ab welchem Sendepunkt ein Wahrheitswert in einer Nachricht ausgewertet wurde. Das heißt, es muss für jede Kombination aus aufgetretenen Sende- und Wartepunkten, von denen es in einem System im Allgemeinen unendlich viele gibt, ein neuer Teil einer oder eine neue Moore-Maschine erstellt werden, in dem oder der sich die Synchronisationsaktion gemerkt wird. Für das Merken von Wartepunkten werden neue, zu den alten Teilen strukturell identische, Teile der Moore-Maschinen entstehen. Der Einzige unterschied in den verschiedenen Teilen ist dann, dass dort aus den Nachrichten ein anderer Wert abgefragt wird. Das heißt, dass die Moore-Maschinen, aus denen das Automatenmodell besteht, im Allgemeinen unendlich viele Zustände haben. Um sich die Sendepunkte zu merken werden mehrere, strukturell identische, Moore-Maschinen erstellt, die erst verspätet mit der Auswertung der Formel beginnen.

Das Versenden sowie das Empfangen von Nachrichten wird nicht von dem Monitor verwaltet, sondern von dem Prozess selber. Daher muss dies in dem Automatenmodell nicht durch Automaten

dargestellt werden. Es muss allerdings beschrieben werden, wie die Wahrheitswerte in die und aus den zu versendenden Nachrichten kommen. Dabei wird einfach angenommen, dass eine Nachricht versandt wird, wenn dies in der Ausführung des Systems auftritt. Wie die aktuellen, beschriebenen Wahrheitswerte in die Nachricht geschrieben werden, wird noch genauer erläutert. Beim Empfangen schreibt der Prozess dann die aktuellen Wahrheitswerte in seinen Speicher und diese werden in der Transitionsfunktion der Automaten durch eine bedingte Transition abgefragt.

Jetzt wird das gewünschte Automatenmodell für eine lokale Betrachtung der Auswertung einer fSDTL-Formel als eine Abwandlung einer Moore-Maschine definiert, welches die nötigen Eigenschaften besitzt, um die Auswertung einer fSDTL-Formel darzustellen.

Definition 4.6 (Unendliche deterministische Moore-Maschine)**.** *Eine unendliche deterministische Moore-Maschine (ISM, (Infinite State Machine)) $M = (\Sigma, Q, q_0, \delta, \Gamma, \lambda)$ ist ein 6-Tupel mit*

- *einem unendlichen Eingabealphabet Σ,*
- *einer unendlichen Menge von Zuständen Q,*
- *einem Startzustand $q_0 \in Q$,*
- *einer Transitionsfunktion $\delta : Q \times \Sigma \to Q$,*
- *einem endlichen Ausgabealphabet Γ und*
- *einer Ausgabefunktion $\lambda : Q \to \Gamma$.*

Sei $w \in \Sigma^*$. Der Lauf einer ISM $M = (\Sigma, Q, q_0, \delta, \Gamma, \lambda)$ bei Eingabe des endlichen Wortes w ist eine Funktion $\rho : \{0, 1, \ldots, |w|\} \to Q$, so dass

- *$\rho(0) = q_0$ und*
- *$\forall i \in \{0, 1, \ldots, |w| - 1\} : \rho(i+1) = \delta(\rho(i), w_i)$.*

Die Ausgabe des Laufes ist dann $\lambda(\rho(|w|))$.

Dieses Modell beinhaltet die nötigen Elemente, um die Auswertung von fSDTL-Formeln darzustellen. Es arbeitet auf endlichen Worten und hat unendlich viele Zustände um die Synchronisationsaktionen modellieren zu können. Außerdem hat es eine Ausgabe, die in jedem Schritt des Laufes den, sich bei fSDTL für diesen Präfix ergebenden, Wert ausgibt. Bis auf die Mächtigkeit des Zustandsmenge entspricht eine ISM einer normalen FSM.

Ein Problem gibt es allerdings noch. In einer ISM kann nur ein möglicher Ausgabewert betrachtet werden. Aus diesem Grund braucht man für jede spätere Auswertung der Formel ab einem Sendepunkt eine neue ISM, welche diesen Wahrheitswert darstellt. Diese ISMs erzeugen dann die Wahrheitswerte, welche in den Nachrichten an andere Prozesse verschickt werden. Das heißt, für eine lokale Auswertung einer fSDTL-Formel ergeben sich unendlich viele ISMs. Nun ergibt sich ein weiteres Problem, nämlich, dass für die lokale Auswertung einer Formel die Wahrheitswerte für die entfernten Teilformeln benötigt werden. Diese Formel müssen als neue fSDTL-Formel betrachtet und separat auf einem Monitor eines anderen Prozess ausgewertet werden. Das heißt,

es wird ein Konstrukt als Automatenmodell benötigt, welches endlich viele Tupel, eines für eine Gesamtformel und weitere für jeden @-Operator, das in dieser vorkommt, mit jeweils unendlich vielen ISMs enthält. Des Weiteren muss in dem Modell noch beschrieben werden, wie Nachrichten versendet und empfangen werden.

Ein Modell, welches diese Anforderungen erfüllt, wird in der folgenden Definition gegeben.

Definition 4.7 (Verteilte unendliche deterministische Moore-Maschine)**.** *Eine verteilte unendliche deterministische Moore-Maschine (DISM, Distributed Infinite State Machine) D ist ein Tupel*

$$D = (M_1, M_2, \ldots, M_n), n \in \mathbb{N}$$

von ∞-Tupeln M_i mit

$$M_i = (A_i, S_{i_1}, S_{i_2}, S_{i_3}, \ldots),$$

wobei A_i eine ISM ist und für die S_{ij} gilt

$$S_{ij} = S_i^{p_1}{}_j, S_i^{p_2}{}_j, \ldots, S_i^{p_m}{}_j.$$

Dabei sind alle $S_i^{p_h}{}_j$ ISMs, $p_1, p_2, \ldots, p_m$ Prozesse aus der Menge der Prozesse P sind und es gilt $|P| = m$. Weiter gilt $A_i = (\Sigma_{A_i}, Q_{A_i}, q_{0A_i}, \delta_{A_i}, \Gamma_{A_i}, \lambda_{A_i})$ und $S_i^{p_h}{}_j = (\Sigma_{S_i^{p_h}{}_j}, Q_{S_i^{p_h}{}_j}, q_{0S_i^{p_h}{}_j}, \delta_{S_i^{p_h}{}_j}, \Gamma_{S_i^{p_h}{}_j}, \lambda_{S_i^{p_h}{}_j})$ für alle $i, j, h \in \mathbb{N}, 1 \leq i \leq n, 1 \leq j, 1 \leq h \leq m$.

Es ergibt sich außerdem für D

- *ein Gesamtalphabet $\Sigma_D = (\bigcup_{1 \leq i \leq n} \Sigma_{A_i}) \cup (\bigcup_{1 \leq i \leq n} \bigcup_{1 \leq j} \bigcup_{1 \leq h \leq |P|} \Sigma_{S_i^{p_h}{}_j})$,*
- *eine Transitionsfunktion $\delta_D : K \times \Sigma_D \to K$ mit*

  $$K = Q_{M_1} \times Q_{M_2} \times \cdots \times Q_{M_n}$$

  wobei für Q_{Mi} gilt

  $$Q_{Mi} = (Q_{A_i} \times Q_{S_{i_1}^{p_1}} \times Q_{S_{i_1}^{p_2}} \times \cdots \times Q_{S_{i_1}^{p_m}} \times Q_{S_{i_2}^{p_1}} \times Q_{S_{i_2}^{p_2}} \times \ldots Q_{S_{i_2}^{p_m}} \times Q_{S_{i_3}^{p_1}} \ldots).$$

  Ein Element aus K enthält für jedes Tupel M_i genau ein Tupel, welches von jedem Automaten aus diesem Tupel einen Zustand enthält und daher eine mögliche aktuelle Konfiguration von D darstellt. K ist damit die Menge aller möglichen Konfigurationen von D. Außerdem ergibt sich für D

- *eine Ausgabefunktion $\lambda_D = \lambda_{A_1}$.*

Es wird als nächstes das Verhalten der Transitionsfunktion δ_D definiert. Seien im Folgenden $w \in \Sigma_D^x$, $0 < s < x$ und $c_{A_i}, c_{S_i^{p_h}{}_j}, 1 \leq i \leq n, 1 \leq j, 1 \leq h \leq |P|$ die Zustände, in denen sich die ISMs A_i beziehungsweise $S_i^{p_h}{}_j$ nach s Schritten aktuell befinden. Sei weiter

$$c_i = (c_{A_i}, c_{S_{i_1}^{p_1}}, c_{S_{i_1}^{p_2}}, \ldots, c_{S_{i_1}^{p_m}}, c_{S_{i_2}^{p_1}}, c_{S_{i_2}^{p_2}}, \ldots, c_{S_{i_2}^{p_m}}, c_{S_{i_3}^{p_1}}, \ldots).$$

Nimmt man alle c_i zusammen, so ergibt sich die aktuelle Konfiguration für D nach s Schritten. Es gilt für δ_D:

$$\delta_D((c_1,c_2,\ldots,c_n),w_{s+1}) = (\Delta(c_1,w_{s+1}),\Delta(c_2,w_{s+1}),\ldots,\Delta(c_n,w_{s+1}))$$

mit der Funktion $\Delta : \bigcup_{1\leq i\leq n} Q_{Mi} \times \Sigma_D \to \bigcup_{1\leq i\leq n} Q_{Mi}$, wobei die Q_{Mi} wie für δ_D beschrieben aufgebaut sind, die wie folgt definiert ist:

$$\Delta(c_i,\sigma) = (\Delta'(c_{A_i},\sigma),\Delta'(c_{S_{i_1}^{p_1}},\sigma),\Delta'(c_{S_{i_1}^{p_2}},\sigma),\ldots,\Delta'(c_{S_{i_1}^{p_m}},\sigma),\Delta'(c_{S_{i_2}^{p_1}},\sigma),\Delta'(c_{S_{i_2}^{p_2}},\sigma),\ldots)$$

Für $\Delta' : Q_D \times \Sigma_D \to Q_D$, wobei

$$Q = \left(\bigcup_{1\leq i\leq n} Q_{A_i}\right) \cup \left(\bigcup_{1\leq i\leq n}\bigcup_{1\leq j}\bigcup_{1\leq h\leq |P|} Q_{S_{ij}^{p_h}}\right)$$

die Menge aller Zustände aller Automaten der Tupel aus D ist, gilt mit der Menge aller Automaten aus D,

$$Aut = \left(\bigcup_{1\leq i\leq n} \{A_i\}\right) \cup \left(\bigcup_{1\leq i\leq n}\bigcup_{1\leq j}\bigcup_{1\leq h\leq |P|} \{S_{ij}^{p_h}\}\right)$$

folgendes:

$$\Delta'(c,\sigma) = \begin{cases} \delta_A(c,\sigma) & \text{wenn } A \in Aut \wedge c \in Q_A \wedge \sigma \in \Sigma_A \\ c & \text{sonst.} \end{cases}$$

Die Funktion Δ' überprüft also, ob das gelesene Zeichen und der Zustand c aus demselben Automaten sind. Ist dies der Fall, so führt Δ' einen Schritt mit der Transitionsfunktion des Automaten durch, wenn es nicht der Fall war, so wird der alte Zustand c beibehalten. Insgesamt nimmt δ_D also die aktuellen Zustände aller ISMs aus D sowie eine Eingabe, prüft für jeden Zustand, ob er und die Eingabe aus demselben Automaten stammen, und führt bei Übereinstimmung einen Schritt in dem Automaten durch.

Jetzt folgt die Betrachtung eines Laufes einer DISM D. Sei $w \in \Sigma^*$. Der Lauf einer DISM D bei Eingabe des endlichen Wortes w ist eine Funktion $\rho : \{0,1,\ldots,|w|\} \to K$, wobei K wie bei δ_D definiert ist, so dass

- $\rho(0) = (q_{0M_1}, q_{0M_2}, \ldots, q_{0M_n})$ *mit*

  $q_{0M_i} = (q_{0A_i} \times q_{0S_{i_1}^{p_1}} \times q_{0S_{i_1}^{p_2}} \times \cdots \times q_{0S_{i_1}^{p_m}} \times q_{0S_{i_2}^{p_1}} \times q_{0S_{i_2}^{p_2}} \times \ldots q_{0S_{i_2}^{p_m}} \times q_{0S_{i_3}^{p_1}} \ldots)$ *und*

- $\forall i \in \{0,1,\ldots,|w|-1\} : \rho(i+1) = \delta_D(\rho(i), w_i)$.

Sei q das erste Element des ersten Tupels aus $\rho(|w|)$ und damit der aktuelle Zustand von A_i. Die Ausgabe des Laufes ist dann $\lambda(q)$.

Als letztes wird noch der Versand und Empfang von Nachrichten betrachtet. Seien $c_{A_i}, c_{S_i{}_j^{p_h}}, 1 \leq i \leq n, 1 \leq j, 1 \leq h \leq |P|$ *die aktuellen Zustände der Automaten zu dem Zeitpunkt, an dem ein Prozess* $p \in P$ *mit einem Alphabet* Σ^p *eine Nachricht* $\uparrow_x^q$ *an einen Prozess* $q \in P$ *versenden will. Sei weiter* $Form : Aut \rightarrow fSDTL$ *eine Funktion, welche die Formel zurückgibt, aus der ein Automat entstanden ist. Aut ist dabei die Menge aller Automaten in den Tupeln aus D, wie vorher beschrieben. Des Weiteren ist* $Send : Aut \rightarrow \mathbb{N}$ *mit*

$$Send(A) = \begin{cases} 0 & wenn\ \exists i : A = A_i \\ j & wenn\ \exists i,h : A = S_i{}_j^{p_h} \end{cases}$$

eine Funktion, die zurückgibt, ab welchem Sendepunkt der Automat die Formel auswertet. Zusätzlich zu den normalen Daten in der Nachricht werden noch die Elemente aus v angehängt, wobei für v gilt

$$v = \{(\gamma, p, Send(A), Form(A)) \mid A \in Aut \wedge \gamma = \lambda_A(c_A) \wedge \Sigma_A = \Sigma^p\}.$$

$\Sigma_A = \Sigma^p$ *stellt dabei sicher, dass der Automat ein Monitor von Prozess p ist. Die Tupel aus v besitzen vier Elemente.* γ *ist dabei die Ausgabe einer ISM und beschreibt den aktuellen Wahrheitswert der Formel ab dem Sendepunkt, ab dem die ISM die Formel auswertet. Die anderen drei Elemente dienen zur Identifikation des Wertes* γ*. p ist der Prozess, zu dem die ISM gehört,* $Send(A)$ *ist der Sendepunkt, ab dem die ISM die Formel auswertet und* $Form(A)$ *ist die Formel, die ausgewertet wurde.*

Empfängt ein Prozess $p \in P$ *eine Nachricht* $\downarrow_x^q$ *von einem Prozess* $q \in P$*, so schreibt dieser die Wahrheitswerte aus der Nachricht, die in den Tupeln aus v stehen, an die geeigneten Stellen in seinem Speicher, die durch die anderen drei Elemente des Tupels beschrieben werden. Die einzelnen ISMs, die zu dem Prozess p gehören, daher alle ISMs I, für die gilt* $I \in Aut \wedge \Sigma_I = \Sigma^p$*, fragen dann diese Werte ab, um bestimmte Transitionen auszuführen.*

Eine DISM ist eine Menge von ∞-Tupeln aus ISMs, welche die benötigten Monitore darstellen. Alle ISMs können für den Versand von Nachrichten benötigt werden, da die Formeln, deren Auswertung sie repräsentieren, von anderen ISMs benötigt werden können. Die erste Maschine des ersten Tupels, in der Definition ist dies A_1, ist die Maschine, welche die Ausgabe der gesamten DISM erzeugt. A_1 sollte daher die gesamte Formel repräsentieren, für die das Automatenmodell erstellt wurde. Weiter besitzt eine DISM noch ein Alphabet, welches sich aus den Alphabeten der einzelnen ISMs ergibt sowie eine Transitions- und eine Ausgabefunktion. Die Transitionsfunktion betrachtet dabei die aktuellen Zustände aller ISMs und bestimmt deren Folgezustände, sofern der gelesene Buchstabe in deren Alphabet ist. Sonst wird der aktueller Zustand der ISM beibehalten. Die Ausgabefunktion verhält sich wie die Ausgabefunktion der Maschine, welche die Ausgabe der DISM repräsentiert, da auch nur der aktuelle Zustand dieser Maschine für die Ausgabe betrachtet wird.

4.3 Umwandlung

In diesem Abschnitt wird die Umwandlung von einer fSDTL-Formel in eine DISM angegeben. Dabei entstehen mehrere Tupel, für jeden @-Operator in der Gesamtformel eines, die jeweils aus unendlich vielen ISMs mit jeweils unendlich vielen Zustände bestehen. Diese Tupel aus ISMs werden dann zu einer DISM zusammengefügt.

Zunächst wird die Umwandlung von einer LTL_3-Formel in eine FSM beschrieben, welche die Grundlage für die Umwandlung von einer fSDTL-Formel in eine DISM bildet.

4.3.1 Umwandlung für LTL_3

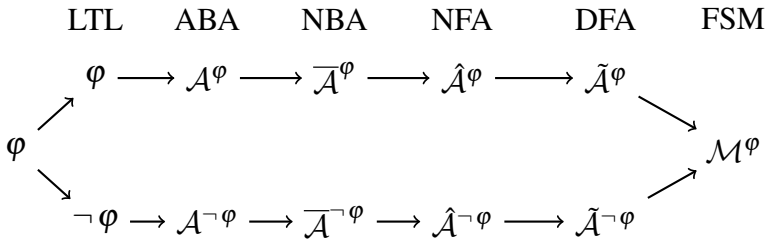


Abbildung 4.1: Der Ablauf der Umwandlung einer Formel φ in eine FSM. Zuerst wird aus der Formel ein ABA $\mathcal{A}^{\varphi}$ erstellt und dieser dann durch eine Potenzmengenkonstruktion in einen NBA $\overline{\mathcal{A}}^{\varphi}$ umgewandelt. Damit die FSM am Ende anticipatory ist, wird danach ein Leerheitstest auf dem NBA ausgeführt und der entstehende Automat als NFA $\hat{\mathcal{A}}^{\varphi}$ interpretiert. Dieser wird wiederum mit einer Potenzmengenkonstruktion in einen DFA $\tilde{\mathcal{A}}^{\varphi}$ umgewandelt. Dieselbe Prozedur wird für die negierte Formel durchgeführt, wodurch sich ein DFA $\tilde{\mathcal{A}}^{\neg\varphi}$ ergibt. Aus den beiden DFAs kann dann die FSM $\mathcal{M}^{\varphi}$ erstellt werden, die den Monitor darstellt.

Die Umwandlung für fSDTL basiert auf der Monitorkonstruktion aus [BLS11], die für LTL_3 entwickelt wurde. Diese ist in Abbildung 4.1 dargestellt. Zuerst wird aus der Formel ein alternierender Büchi-Automat (ABA) erstellt und dieser dann in einen nicht-deterministischen Büchi-Automaten (NBA) umgewandelt. Dies wird durch eine Potenzmengenkonstruktion erreicht, wobei sich in den Zuständen des entstehenden NBAs die Zustände, in denen sich der ABA zu einem Zeitpunkt gleichzeitig befindet, gemerkt werden. Auf dem NBA wird dann von jedem Zustand aus ein Leerheitstest durchgeführt. Dabei wird von jedem Zustand aus untersucht, ob es noch ein Wort gibt, das akzeptiert wird. Alle Zustände, von denen aus noch akzeptiert werden kann, werden dann als akzeptierend markiert. Der entstehende Automat wird als ein nicht-deterministischer endlicher Automat (NFA) interpretiert. Durch den Leerheitstest auf den Zuständen des NBAs wird die Anticipation sichergestellt, da dadurch nur die Zustände nicht akzeptierend sind, von denen aus auf keinen Fall mehr akzeptiert werden kann. Der NFA wird danach wiederum durch eine Potenzmengenkonstruktion in einen deterministischen endlichen Automaten (DFA) umgewandelt. Dabei merkt sich der DFA in seinen Zuständen, in welchen Zuständen der NFA zu einem Zeitpunkt

nicht-deterministisch sein kann. Selbiges wird für die Negation der Formeln getan, wodurch sich am Ende der DFA für die negierte Formel ergibt. Durch den Weg über die positive Formel erkennt man durch den Leerheitstest Zustände, die nicht mehr akzeptieren können und durch den Weg über die negierte Formel Zustände, die nicht mehr verwerfen können. Durch das Zusammenführen der beiden entstehenden DFAs erhält man eine FSM, die für die Formel, aus der die DFAs konstruiert wurden, genau dann $\top$ bzw. $\bot$ ausgibt, wenn dies auch für die Auswertung der Formel mit der LTL_3-Semantik der Fall wäre.

4.3.2 Umwandlung für fSDTL

Diese Umwandlung für LTL_3 wird nun für fSDTL angepasst. Das heißt, die entstehenden Automaten haben unendlich viele Zustände und werden auch aus Formeln generiert, die entfernte Formeln als Teilformeln besitzen. Der @-Operator ähnelt dabei einer Proposition, die abhängig von den aktuellen Werten in den neusten Nachrichten zu einem Wahrheitswert ausgewertet wird. Die entfernten Formeln werden dann von weiteren Automaten ausgewertet, die dadurch die Werte für die Nachrichten generieren. Die Umwandlung transformiert nach der Anpassung nur eine Formel in ein Tupel aus ISMs, daher müssen für die Umwandlung einer gesamten fSDTL-Formel mehrere Teilformeln, nämlich noch alle, die von einem @ umgeben sind, jeweils mit derselben Umwandlung transformiert werden. Das heißt, es entsteht ein Tupel mit unendlich vielen Monitoren für die Gesamtformel und weitere Tupel mit unendlich vielen Monitoren für die entfernten Teilformeln. Diese verschiedenen, unendlichen Größen werden allerdings in Abschnitt 4.4 auf Seite 93 in verschiedenen Betrachtungen wieder auf endliche Größen beschränkt. Des Weiteren besitzen die entstehenden, unendlich großen Automaten eine sehr spezielle Struktur aus identischen, endlichen Elementen, wie am Ende der Umwandlung gezeigt wird.

In Abbildung 4.2 auf der nächsten Seite ist der Verlauf der angepassten Umwandlung auf dieselbe Art wie in Unterabschnitt 4.3.1 auf der vorherigen Seite dargestellt. Wie man sieht ist der Verlauf identisch. Anders ist, dass in jedem Schritt jeweils statt nur einem Automaten immer ein Tupel mit unendlich vielen entsteht. Wie die Übergänge von einem Schritt zu einem anderen im Detail funktionieren wird im weiteren Verlauf dieses Abschnittes erläutert.

Im Folgenden wird nur die Transformation des positiven Weges der Umwandlung in einen DIA gezeigt, da der Weg über die negierte Formel äquivalent funktioniert.

4.3.3 fSDTL-Formel zu IABAs

Zuerst wird die Negationsnormalform (NNF) für fSDTL-Formeln definiert, da diese im Folgenden benötigt wird. Des Weiteren wird gezeigt, dass sich jede fSDTL-Formel in NNF umwandeln lässt.

Definition 4.8 (Negationsnormalform)**.** *Eine Formel φ mit $@_p\varphi \in$ fSDTL ist in Negationsnormalform (NNF), wenn Negationen $(\neg)$ in φ außerhalb von entfernten Teilformeln nur vor Propositionen oder entfernten Teilformeln vorkommen.*

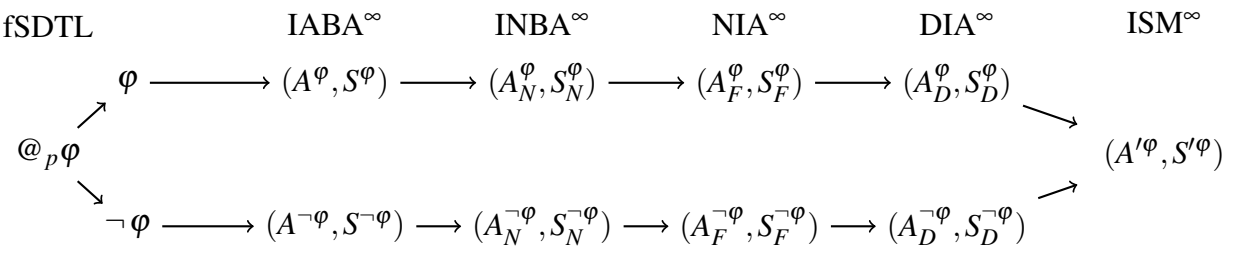


Abbildung 4.2: Der Ablauf der Umwandlung einer Formel fSDTL-Formel $@_p\varphi$ in ein Tupel aus ISMs. Das ∞ steht jeweils dafür, dass es immer unendlich große Tupel aus Automaten des jeweiligen Typs sind. Das Element A der Tupel ist immer ein einzelner Automat, der die Auswertung der Formel darstellt und das S steht immer für unendlich viele Automaten, welche die Auswertung der Formel ab den verschiedenen Sendepunkten repräsentieren. Zuerst wird aus der Formel ein Tupel aus IABAs erstellt und jeder IABA dann durch die bekannte Potenzmengenkonstruktion in einen INBA umgewandelt. Damit die ISMs am Ende anticipatory sind, wird wieder ein Leerheitstest auf den INBAs ausgeführt und die entstehenden Automaten als NIAs interpretiert. Diese werden wiederum mit der bekannten Potenzmengenkonstruktion in DIAs umgewandelt. Dieselbe Prozedur wird für die negierte Formel durchgeführt. Aus den beiden Tupeln aus DIAs kann dann ein Tupel aus ISMs erstellt werden.

Lemma 4.9 (Negationsnormalform von fSDTL-Formeln)**.** *Für jede Formel φ mit $@_p\varphi \in$ fSDTL existiert eine äquivalente Formel ψ mit $@_p\psi \in$ fSDTL in NNF.*

Beweis. In einer Formel φ mit $@_p\varphi \in$ fSDTL werden für Negationen außerhalb von entfernten Teilformeln so lange die De-Morgan Regeln für Aussagenlogik und temporale Logiken angewendet, bis eine äquivalente Formel ψ mit $@_p\psi \in$ fSDTL in NNF entsteht. □

Im weiteren Verlauf dieses Unterabschnittes folgt nun die Umwandlung von einer fSDTL-Formel in ein ∞-Tupel aus IABAs.

Sei P die Menge aller Prozesse, $p \in P$ ein Prozess und $@_p\varphi$ die fSDTL-Formel, wobei φ in NNF ist, über einem Alphabet $\Sigma = 2^{AP}$, wobei AP die Menge von atomaren Propositionen des Prozesses p ist, die in ein Automatenmodell umgewandelt werden soll. Als erstes wird φ in ein ∞-Tupel (A^φ, S^φ) aus einem Formel-IABA A^φ und unendlich vielen Sendepunkt-IABAs aus S mit

$$S^\varphi = S^\varphi{}_1^1, S^\varphi{}_1^2, \ldots, S^\varphi{}_1^{|P|-1}, S^\varphi{}_2^1, S^\varphi{}_2^2, \ldots, S^\varphi{}_2^{|P|-1}, S^\varphi{}_3^1, S^\varphi{}_3^2, \ldots, S^\varphi{}_3^{|P|-1}, \ldots$$

umgewandelt. Dabei ist der Formel-IABA $A^\varphi = (\Sigma, Q^\varphi, q_0^\varphi, \delta^\varphi, F^\varphi)$ der Automaten, der die Ausgabe für die Auswertung von φ, also der gesamten Formel, bereitstellt sowie den Wahrheitswert für eine Nachricht, der sich bei der Auswertung von φ vom Beginn des Laufes an ergibt. Die Sendepunkt-IABAs aus S^φ sind die Automaten, welche die Wahrheitswerte für eine Nachricht ab

den verschiedenen Sendepunkten mit den anderen Prozessen ausgeben. Die Elemente eines ${S^\varphi}_i^j$ aus S^φ werden mit ${S^\varphi}_i^j = ({\Sigma^\varphi}_i^j, {Q^\varphi}_i^j, {q_0^\varphi}_i^j, {\delta^\varphi}_i^j, {F^\varphi}_i^j)$ bezeichnet.

Im Folgenden wird zuerst die Erstellung von A^φ beschrieben und danach die Anpassung, um die IABAs aus S^φ zu erhalten. Nachdem die IABAs erstellt wurden, werden alle auf dieselbe Weise weiter in eine ISM transformiert.

Sei $X = \{\psi' \mid @_r\psi' \in \text{fSDTL}\}$, $\psi_1, \psi_2 \in X$ in NNF, a eine atomare Proposition und $q \in P$ ein Prozess. Die Menge aller Teilformeln einer Formel in NNF ist dann durch $sub : X \to 2^X$ wie folgt beschrieben:

$$\begin{aligned}
sub(\text{true}) &= \emptyset \\
sub(\text{false}) &= \emptyset \\
sub(a) &= \{a\} \\
sub(\neg a) &= \{\neg a\} \\
sub(\psi_1 \vee \psi_2) &= \{\psi_1 \vee \psi_2\} \cup sub(\psi_1) \cup sub(\psi_2) \\
sub(\psi_1 \wedge \psi_2) &= \{\psi_1 \wedge \psi_2\} \cup sub(\psi_1) \cup sub(\psi_2) \\
sub(\bigcirc \psi_1) &= \{\bigcirc \psi_1\} \cup sub(\psi_1) \\
sub(\psi_1 \,\mathcal{U}\, \psi_2) &= \{\psi_1 \,\mathcal{U}\, \psi_2\} \cup sub(\psi_1) \cup sub(\psi_2) \\
sub(\psi_1 \,\mathcal{R}\, \psi_2) &= \{\psi_1 \,\mathcal{R}\, \psi_2\} \cup sub(\psi_1) \cup sub(\psi_2) \\
sub(@_q\psi_1) &= \{@_q\psi_1\} \\
sub(\neg @_q\psi_1) &= \{\neg @_q\psi_1\}.
\end{aligned}$$

Es gilt dann $(s,m) \in Q^\varphi$ für alle $s \in \text{sub}(\varphi)$ und $m \in \mathbb{N}^{|P|-1}$. m ist ein Tupel aus natürlichen Zahlen, in dem für jeden anderen Prozess gezählt wird, welches der für eine entfernte Formel zu betrachtende Wartepunkt ist. Das $|P|-1$ entsteht, weil die umzuwandelnde Formel immer auf einem Prozess ausgewertet wird und dieser selber keine Position in m braucht. q_0^φ ist $(\varphi, \vec{0})$.

Sei $a \in AP$, $\sigma \in \Sigma$ und $\psi_1, \psi_2 \in X$ in NNF. Für die Transitionsfunktion δ^φ werden drei Hilfsfunktionen definiert. Zuerst eine bijektive Funktion $n : P \setminus p \to \{1, 2, \ldots, |P|-1\}$, die jedem Prozess außer p einen Index zuordnet. Für Prozess p wird kein Index benötigt, weil die umzuwandelnde Formel $@_p\varphi$ auf p selber ausgewertet wird. Als nächstes die Funktion $h : \mathbb{N}^{|P|-1} \times \Sigma \to \mathbb{N}^{|P|-1}$ mit

$$h(m,\sigma) = m' \iff \forall q \in P : m'_{n(q)} = h'(q,m,\sigma)$$

wobei für $h' : P \times \mathbb{N}^{|P|-1} \times \Sigma \to \mathbb{N}$ gilt

$$h'(q,m,\sigma) = \begin{cases} l & \text{wenn } \lambda_{\downarrow l}^q \in \sigma \\ m_{n(q)} & \text{sonst} \end{cases}.$$

h sucht in einem Element σ des Alphabetes Wartepunkte und aktualisiert dann das Tupel m der bisher gesehenen Wartepunkte, um zu wissen, welche Werte der empfangenen Nachrichten abge-

fragt werden müssen. Dafür wird von h die ID eines Wartepunktes an die Stelle in m geschrieben, die zu dem Prozess gehört, auf den durch den Wartepunkt gewartet werden soll.

Als letztes wird noch eine Funktion $f : P \times \mathbb{N} \times \text{fSDTL} \to \mathbb{B}_3$ benötigt, die von einem Prozess q, dem Index der zu betrachtenden Synchronisationsaktion k und einer fSDTL-Formel ψ auf den Wahrheitswert abbildet, der auf dem Prozess, auf dem die Funktion ausgeführt wird, in der durch die Parameter spezifizierten Stelle im Speicher steht.

Für $\delta^\varphi : Q \times \Sigma \to \mathbb{B}^+(Q)$ gilt dann folgendes:

$$\delta^\varphi((a,m),\sigma) = \begin{cases} \text{true} & \text{wenn } a \in \sigma \\ \text{false} & \text{sonst} \end{cases}$$

$$\delta^\varphi((\neg a,m),\sigma) = \begin{cases} \text{true} & \text{wenn } a \notin \sigma \\ \text{false} & \text{sonst} \end{cases}$$

$$\delta^\varphi((\psi_1 \vee \psi_2,m),\sigma) = \delta^\varphi((\psi_1,m),\sigma) \vee \delta^\varphi((\psi_2,m),\sigma)$$

$$\delta^\varphi((\psi_1 \wedge \psi_2,m),\sigma) = \delta^\varphi((\psi_1,m),\sigma) \wedge \delta^\varphi((\psi_2,m),\sigma)$$

$$\delta^\varphi((\bigcirc \psi_1,m),\sigma) = \begin{cases} \delta^\varphi(\psi_1,h(m,\sigma)) & \text{wenn } \psi_1 \in \text{fSDTL} \\ (\psi_1,h(m,\sigma)) & \text{sonst} \end{cases}$$

$$\delta^\varphi((\psi_1 \,\mathcal{U}\, \psi_2,m),\sigma) = \delta^\varphi((\psi_2 \vee (\psi_1 \wedge \bigcirc \psi_1 \,\mathcal{U}\, \psi_2),m),\sigma)$$

$$\delta^\varphi((\psi_1 \,\mathcal{R}\, \psi_2,m),\sigma) = \delta^\varphi((\psi_2 \wedge (\psi_1 \vee \bigcirc \psi_1 \,\mathcal{R}\, \psi_2),m),\sigma)$$

$$\delta^\varphi((@_q\psi_1,m),\sigma) = \begin{cases} \text{true} & \text{wenn } f(q,m_{n(q)},@_q\psi_1) = \top \\ \text{false} & \text{wenn } f(q,m_{n(q)},@_q\psi_1) = \bot \\ (@_q\psi_1,m) & \text{sonst} \end{cases}$$

$$\delta^\varphi((\neg@_q\psi_1,m),\sigma) = \begin{cases} \text{false} & \text{wenn } f(q,m_{n(q)},@_q\psi_1) = \top \\ \text{true} & \text{wenn } f(q,m_{n(q)},@_q\psi_1) = \bot \\ (\neg@_q\psi_1,m) & \text{sonst} \end{cases}$$

Die Menge F^φ der akzeptierenden Zustände ergibt sich dann als

$$F^\varphi = \{(@_q\psi, m), (\neg@_q\psi, m), (\psi_1 \mathcal{R} \psi_2, m) \in Q\}$$

Für alle Sendepunkt-IABAs $S^{\varphi\,j}_i \in S^\varphi$ gilt $\Sigma^{\varphi\,j}_i = \Sigma$, $Q^{\varphi\,j}_i = Q^\varphi \cup \{(\lambda, m) \mid m \in \mathbb{N}^{|P|-1}\}$, $q^{\varphi\,j}_{0\,i} = (\lambda, \vec{0})$ und $F^{\varphi\,j}_i = F^\varphi \cup \{(\lambda, m) \mid m \in \mathbb{N}^{|P|-1}\}$. Auch $\delta^{\varphi\,j}_i$ entspricht δ^φ bis auf folgende zusätzliche Transitionen:

$$\delta^{\varphi\,j}_i((\lambda, m), \sigma) = \begin{cases} \delta^\varphi((\varphi, m), \sigma) & \text{wenn } \exists \lambda_{\uparrow i}^{n^{-1}(j)} \in \sigma \\ (\lambda, h(m, \sigma)) & \text{sonst} \end{cases}$$

Im Gegensatz zu der Umwandlung für LTL_3 muss Q^φ in diesem Fall mehr sein als nur die Menge der Subformeln von φ. Jeder Zustand benötigt noch ein $|P|-1$-Tupel m, im Folgenden Wartepunkttupel genannt, zum Merken der gesehenen Wartepunkte. Die verschiedenen $S^{\varphi\,j}_i$ repräsentieren die Auswertungen ab gesehenen Sendepunkten. Daher besitzt jedes $Q^{\varphi\,j}_i$ zusätzlich noch Zustände (λ, m). Diese werden im Folgenden Wartezustände genannt und stellen auch solche dar, welche auf den, zu $S^{\varphi\,j}_i$ passenden, Sendepunkt warten.

δ^φ entspricht fast der Transitionsfunktion aus der Umwandlung für LTL_3. Unterschiedlich ist, dass zwischendurch m aktualisiert werden muss, damit sich die gesehenen Wartepunkte gemerkt werden und dass es noch Transitionen für Zustände mit einer entfernten Formel gibt. Das Aktualisieren für Wartepunkte ist nur bei dem Next-Operator ($\bigcirc$) nötig, da alle anderen temporalen Operatoren durch abrollen auf den Next-Operator zurückgeführt werden. Entspricht die Teilformel, die von dem Next-Operator umgeben ist, einer fSDTL-Formel, so ist es nötig, dass diese Teilformel direkt ausgewertet wird. Andernfalls könnte in den Zustand der entfernten Formel gewechselt werden obwohl vorher schon in einer Nachricht stand, welchen endgültigen Wert diese hat und damit das Ergebnis schon bekannt ist. Das Aktualisieren für Sendepunkte ist nicht nötig, da diese durch die Sendepunkt-IABAs $S^{\varphi\,j}_i$ repräsentiert werden. Befindet man sich in einem Zustand mit einer entfernten Teilformel, so ist das σ gar nicht mehr wichtig, denn eine entfernte Teilformel wartet nur auf einen Wahrheitswert aus einer Nachricht und daher ist es egal, was auf dem eigenen Prozess passiert. Des Weiteren muss auch bei einem Wartepunkt keine Aktualisierung mehr stattfinden, denn wenn sich ein Automat in einem Zustand mit einer entfernten Teilformel befindet, so ist der Wartepunkt, ab dem ausgewertet werden soll, schon sicher, weil in der Semantik nur nach Wartepunkten vor dem Zeitpunkt gesucht wird, ab dem die entfernte Teilformel ausgewertet werden soll. In den $\delta^{\varphi\,j}_i$ wird, solange der gesuchte Sendepunkt nicht auftritt, in dem aktuellen Wartezustand gewartet und m gegebenenfalls aktualisiert. Tritt der gesucht Sendepunkt auf, so beginnt die Auswertung. Da zu demselben Zeitpunkt auch der erste Schritt des Automaten durchgeführt wird, wird der Startzustand des A^φ entsprechenden Teilautomaten in dem $S^{\varphi\,j}_i$ übersprungen und stattdessen direkt in den entsprechenden, zweiten Zustand gewechselt.

f ist eine Funktion, die zum Auslesen der aktuellen Nachrichtenwerte aus dem Speicher des Prozesses für die Transitionsfunktion genutzt wird. Sie betrachtet die, mit ihren Parametern spezifizierte, Stelle im Speicher und gibt den dort stehenden Wahrheitswert zurück.

Am Anfang muss noch die gesamte Formel ausgewertet werden und es wurden noch keine Wartepunkte gesehen, weshalb $q_0^{\varphi} = (\varphi, \vec{0})$ gilt. Da zu Beginn auch noch keine Sendepunkte gesehen wurden, sind alle $S^{\varphi}{}_i^j$ in ihrem Wartezustand, weshalb für alle $S^{\varphi}{}_i^j$ gilt $q_{0\,i}^{\varphi\,j} = (\lambda, \vec{0})$.

Die Menge der akzeptierenden Zustände ergibt sich, weil Zustände der Form $(\psi_1 \mathcal{R} \psi_2, m)$ in jedem Schritt wieder in sich selbst zurückkehren oder in einem Zustand mit derselben Teilformel und einem aktualisierten m enden. Wenn diese Zustände nicht akzeptierend wären, könnte von ihnen aus deshalb nie akzeptiert werden, was aber möglich sein muss. Zustände, die auf einen Wert für eine entfernte Formel warten, also Zustände der Form $(@_q \psi, m)$ oder $(\neg @_q \psi, m)$, sind akzeptierend, damit der Automat nur verwirft, wenn fSDTL$_\omega$ den Wahrheitswert $\perp$ als Ergebnis einer Auswertung ausgibt. Für die $S^{\varphi}{}_i^j$ werden noch die Wartezustände hinzugefügt. Dies ist nötig, damit diese am Ende der Umwandlung als Ausgabewert ? besitzen und dadurch nicht zu früh ein endgültiger Wahrheitswert in einer Nachricht versandt wird.

Als nächstes wird eine wichtige Beobachtung über die Struktur der entstandenen IABAs festgehalten. Diese ist wichtig für die später folgende Analyse der Umwandlung, da sie beschreibt, dass die IABAs eine sehr einfache Struktur besitzen. Um diese zu beschreiben wird zuerst eine Schicht definiert.

Definition 4.10 (Alternierende Schicht)**.** *Eine Menge von Zuständen $Q_L \subseteq Q$ von einem IABA $(\Sigma, Q, q_0, \delta, F)$ wird auch alternierende Schicht (kurz: Schicht) genannt, wenn es ein m gibt, so dass folgende Bedingungen erfüllt sind:*

- *Für alle $(\psi, m) \in Q$ gilt auch $(\psi, m) \in Q_L$ und*
- *für alle $(\psi, m), (\psi', m') \in Q_L$ gilt $m = m'$.*

Für eine alternierende Schicht Q_L ergibt sich die Transitionsfunktion $\delta_L : Q_L \times \Sigma \to \mathbb{B}^+(Q_L)$ wie folgt:

$$\forall q \in Q_L : \delta_L(q, \sigma) = \delta(q, \sigma) \Leftrightarrow \delta(q, \sigma) \in \mathbb{B}^+(Q_L)$$

Außerdem ergibt sich die Menge der akzeptierenden Zustände $F_L \subseteq F$ in Q_L durch

$$\forall q \in Q_L : q \in F_L \Leftrightarrow q \in F$$

Eine alternierende Schicht eines IABAs kann als ein Teil-ABA ohne Startzustand angesehen werden, aus dem man nur durch einen Wartepunkt wechseln kann. Die zwei Bedingungen stellen dabei sicher, dass eine Schicht immer genau alle Zustände mit demselben Wartepunkttupel enthält. Eine alternierende Schicht ist eine Teilmenge Q_L der Zustände des IABAs, von dem sie gebildet wird. Die Transitionsfunktion der Schicht enthält alle Transitionen, die von einem Zustand der Schicht

in Zustände der Schicht wechseln und die Menge der akzeptierenden Zustände der Schicht enthält alle Zustände, die auch in dem IABA akzeptierend sind.

Die Struktur der einzelnen IABAs aus $(A^{\varphi}, S^{\varphi})$ kann nun mittels alternierender Schichten betrachtet werden. Der IABA A^{φ} besitzt durch seinen Zustandsraum und seine Transitionsfunktion unendlich viele, isomorphe Schichten. Alle Zustände mit demselben m bilden immer eine Schicht und die Transitionen zwischen diesen bilden die Transitionsfunktion der Schicht. Weiter entspricht eine alternierende Schicht genau dem ABA, der bei der Umwandlung einer LTL-Formel in einen ABA entsteht, bis auf den zusätzlichen @-Operator und den fehlenden Startzustand. Durch gesehene Wartepunkte kann in dem IABA A^{φ} zwischen den Schichten gewechselt werden. Da immer nur neuere Wartepunkte gesehen werden können, kann von einer alternierenden Schicht nur in eine andere alternierende Schicht gewechselt werden, deren einzelnen Werte in m gleich oder größer sind als die der vorherigen Schicht. Aus diesem Grund ist es auch nie möglich, in eine frühere Schicht zurück zu wechseln. Für die IABAs aus S^{φ} kann noch eine äquivalente Einteilung in isomorphe Schichten vorgenommen werden. Sobald diese in INBAs transformiert wurden ist dies nicht mehr möglich, da dann nicht mehr alle Schichten isomorph sind, wie später genauer beschrieben wird. Aus diesem Grund ist die wichtige Erkenntnis für die IABAs aus S^{φ} nur, dass A^{φ}, den sie als Teilautomaten besitzen und in den, von den Wartezuständen aus, durch den gesuchten Sendepunkt gewechselt wird, die vorher beschriebene Schichtstruktur besitzt.

Dadurch ergibt sich, dass der IABA A^{φ} sowie die A^{φ} entsprechenden Teilautomaten der IABAs aus S^{φ} aus unendlich vielen, isomorphen, endlichen Strukturen bestehen, zwischen denen mit Wartepunkten gewechselt werden kann.

Diese Betrachtung mittels Schichten wird auch eine wichtige Rolle für die Betrachtung der Struktur eines entstehenden INBAs spielen. Selbiges gilt auch für die später entstehenden NIAs sowie die DIAs.

Als nächstes soll gezeigt werden, dass sich die Automaten aus dem Tupel $(A^{\varphi}, S^{\varphi})$ korrekt verhalten. Dafür wird zuerst gezeigt, dass sich die Werte der Funktion $\text{last}_{\lambda_{\downarrow}}$ aus der fSDTL-Semantik und die Werte aus dem, in der Umwandlung benutzten, m gleichen.

Lemma 4.11 (Äquivalenz von $\text{last}_{\lambda_{\downarrow}}$ und m)**.** *Zu jedem Zeitpunkt gilt* $last_{\lambda_{\downarrow}}(w,i,p,q) = m_{n(q)}$ *für einen Zustand* $(@_q\psi, m)$ *der nach* $i-1$ *Schritten von einem anderen Zustand aus in einem Formel-IABA erreicht wurde, der für eine Formel* $@_p\psi'$ *entstanden ist.*

Beweis. Wie bei der Semantik von fSDTL beschrieben, gibt $\text{last}_{\lambda_{\downarrow}}(w,i,p,q)$ den Index des letzten Wartepunktes vor der Position i im Lauf von Prozess p zurück, bei dem auf q gewartet werden muss. Da $n(q)$ den Index des Prozesses q zurückgibt, wird mit $m_{n(q)}$ die Stelle in m beschrieben, welche die Wartepunktindizes der Wartepunkte, bei denen auf Prozess q gewartet werden muss, enthält. Werden nun $i-1$ Schritte auf dem IABA ausgeführt, so wird bei jedem gesehenen Wartepunkt das m aktualisiert, solange noch ein Zustand mit einer entfernten Teilformel erreichbar ist. Dies gilt, denn immer wenn noch ein Zustand mit einer entfernten Teilformel erreichbar ist, so muss es noch einen temporalen Operator um die Teilformel in dem aktuell betrachteten Zustand geben. Dieser wird auf einen Next-Operator zurückgeführt, bei dem das Wartepunkttupel m bei einem auftretenden Wartepunkt aktualisiert wird. Das heißt, wenn ein Wartepunkt gesehen wird,

an dem auf Prozess q gewartet werden muss, so wird die Stelle $m_{n(q)}$ aktualisiert. Daher enthält $m_{n(q)}$ den Index des zuletzt gesehenen Wartepunktes, an dem auf Prozess q gewartet werden muss, nach $i-1$ Schritten und damit gilt $\text{last}_{\lambda_\downarrow}(w,i,p,q) = m_{n(q)}$, da $\text{last}_{\lambda_\downarrow}(w,i,p,q)$ den Index des letzten Wartepunktes vor Position i, also an Position $i-1$ oder früher, zurückgibt. □

Mit Hilfe dieses Lemmas kann nun die Korrektheit von A^φ gezeigt werden. Korrekt heißt in diesem Fall, dass A^φ genau dann ein Wort nicht akzeptiert, wenn die fSDTL$_\omega$-Semantik für dieses Wort $\bot$ ausgibt.

Theorem 4.12 (Korrektheit des Formel-IABAs A^φ)**.** *Sei $w \in \Sigma^\omega$. Es gilt $w \notin \mathcal{L}(A^\varphi) \Leftrightarrow$ $[\![w,0 \models \varphi]\!]_{fSDTL_\omega} = \bot$ unter der Annahme, dass die Werte, die f zurückgibt, korrekt sind.*

Beweis. Nach [Var95] gibt es für jede LTL-Formel ψ einen ABA $\overline{A}$, so dass $w \notin \mathcal{L}(\overline{A}) \Leftrightarrow [\![w \models \psi]\!]_{\text{LTL}} = \bot$ gilt. Wenn die Wartepunkte nicht betrachtet werden, also immer $m = \vec{0}$ gilt und damit nur die alternierende Schicht $\{(x,\vec{0}) \in Q^\varphi\}$ betrachtet wird, entspricht die Umwandlung der fSDTL-Formel zu A^φ der Umwandlung einer LTL-Formel zu einem entsprechenden ABA bis auf den zusätzlichen @-Operator. Von einem Zustand mit einer entfernten Teilformel aus wird nur dann verworfen, wenn in einer Nachricht für den Wert der entfernten Teilformel $\bot$ stand. Auch in der fSDTL$_\omega$-Semantik wird für eine entfernte Teilformel nur $\bot$ ausgegeben, wenn dies in einer Nachricht stand. Eine einzelne Schicht erfüllt daher die Annahme.

Alle alternierenden Schichten für die verschiedenen Wartepunkttupel sind isomorph, wie vorher schon beschrieben. Von einer Schicht kann nur in eine andere gewechselt werden, wenn ein Wartepunkt gesehen wird und dies auch nur bei der Auswertung eines Next-Operators, um sich diesen zu merken. Die Auswertung wird dabei fortgeführt, denn zusätzlich zu dem Wechseln in die andere Schicht wird auch der Schritt für den Rest des Zeichens in der Auswertung gemacht. Wenn von einer alternierenden Schicht in eine andere gewechselt wird, dabei aber der Fortschritt in der Auswertung durch Wechseln in den entsprechenden Zustand beibehalten wird, so wird das Ergebnis der Auswertung durch das Wechseln der Schicht nicht verfälscht.

Zu zeigen bleibt noch, dass sich die Wartepunkte immer korrekt gemerkt werden. Dies wurde in Lemma 4.11 auf der vorherigen Seite gezeigt. Da nun die Korrektheit der Werte, die f zurückgibt und damit auch die Korrektheit der Werte in den Nachrichten, angenommen wird und der Wert, den m für den Index des letzten Wartepunkte enthält, dem von $\text{last}_{\lambda_\downarrow}$ entspricht, gibt $f(q, m_{n(q)}, @_q\psi)$ denselben Wert wie die Auswertung der entfernten Formel $@_q\psi$ mittels fSDTL-Semantik zurück. Dadurch wartet der Automat wenn sich ein ? ergibt, geht bei $\top$ nach true oder bei $\bot$ nach false. Vor allem aber geht er genau dann nach false, wenn die fSDTL$_\omega$-Semantik $\bot$ für die entfernte Formel ergibt und verwirft daher genau dann.

Damit folgt die Korrektheit aller Schichten sowie die Korrektheit des Wechselns zwischen diesen und damit die Annahme. □

Als nächstes wird die Korrektheit von S^φ bewiesen. Der Beweis ähnelt dem für das vorherige Theorem sehr und wird daher auf diesen zurückgeführt.

Theorem 4.13 (Korrektheit der Sendepunkt-ABAs $S^{\varphi}{}_i^j$ aus S^{φ})**.** *Sei $w \in \Sigma^{\omega}$, q ein Prozess mit $n(q) = j$ und $\lambda_{\uparrow}{}_i^q \in w_h$. Es gilt $w \notin \mathcal{L}(S^{\varphi}{}_i^j) \Leftrightarrow [\![w,h \models \varphi]\!]_{fSDTL_{\omega}} = \top$ unter der Annahme, dass die Werte, die f zurückgibt, korrekt sind.*

Beweis. Wie vorher beschrieben, besteht ein $S^{\varphi}{}_i^j$ nur aus Wartezuständen, für jedes m einen, und dem Formel-IABA A^{φ} als Teilautomaten.

Das heißt, aus den Wartezuständen wechselt ein $S^{\varphi}{}_i^j$ nur heraus, wenn der entsprechende Sendepunkt an Position h des Wortes gesehen wird. Solange wird unabhängig der Eingabe in den Wartezuständen gewartet und bei einem Wartepunkt in einen anderen Wartezustand mit aktualisiertem m gewechselt. Dadurch werden sich auch innerhalb der Wartezustände die Wartepunkte in m korrekt gemerkt, da m bei jedem gesehenen aktualisiert wird. Dies ist nötig, da $\text{last}_{\lambda_{\downarrow}}$ auch in den Positionen vor h nach Wartepunkten sucht.

Wird nun die Position h des Wortes erreicht, so wechselt $S^{\varphi}{}_i^j$ von dem aktuellen Wartezustand (λ, m') zu dem Folgezustand des Zustandes (φ, m'). Es wird also die Auswertung der Formel wie gewohnt in dem Formel-IABA A^{φ} ab dem h-ten Zeichen von w mit einem anderen Startzustand, nämlich (φ, m'), begonnen. Da sich vorher die Wartepunkte in den Wartezuständen korrekt gemerkt wurden und alle alternierenden Schichten von A^{φ} identisch sind, folgt mit Theorem 4.12 auf der vorherigen Seite die Annahme. □

Wie zu sehen ist werden bei den IABAs in den Beweisen nur die nicht-akzeptierenden Zustände sowie aus der Semantik die Ausgaben von $\bot$ betrachtet. Dies reicht aus, wie später gezeigt wird, weil zusätzlich noch der Weg über die negierte Formel gegangen wird.

Beispiel 4.14 (Umwandlung einer fSDTL-Formel in IABAs)**.** *In diesem Beispiel wird die Umwandlung einer konkreten fSDTL-Formel in eine DISM betrachtet.*

Seien p und q Prozesse aus der Menge der Prozesse $P = \{p,q\}$ mit den Mengen an Nachrichten und Synchronisationsaktionen X^p und X^q und den Mengen an atomaren Propositionen $AP^p = \{a\} \cup X^p$ und $AP^q = \{b\} \cup X^q$. Die Formel $@_p\varphi = @_p \Diamond (a \wedge \bigcirc @_q(b))$ soll nun in eine DISM umgewandelt werden.

Zuerst wird das Tupel $(A^{\varphi}, S^{\varphi})$ erstellt, wie in der Umwandlung beschrieben. Für den Index von q gilt $n(q) = 1$. Da es nur einen Prozess außer p gibt, gilt $m = (m_1)$

Im Folgenden werden Nachrichten für bessere Übersichtlichkeit nicht in den Mengen angezeigt, die beschreiben, wann eine Transition beschritten wird, da sie keine Auswirkungen auf die Auswahl der Transition haben. Weiter wird durch $\lambda_{\uparrow}*$ angegeben, dass die Transition bei beliebigem oder auch keinem Sendepunkt beschritten werden kann. Selbiges gilt für $\lambda_{\downarrow}*$. $[*]$ bedeutet, dass die Werte der Nachrichten unwichtig für diese Transition sind. Σ^p ist das Alphabet von p und steht für eine beliebige Eingabe. $[x = y]$ ist eine Bedingung, welche genutzt wird um zu prüfen, ob die Werte der angekommenen Nachrichten den nötigen entsprechen. Gilt $x = y$, so kann die Transition ausgeführt werden, sonst nicht. true und false sind nur zur besseren Übersicht mehrfach eingezeichnet. Da sie keine Zustände sind gehören sie auch zu keiner Schicht.

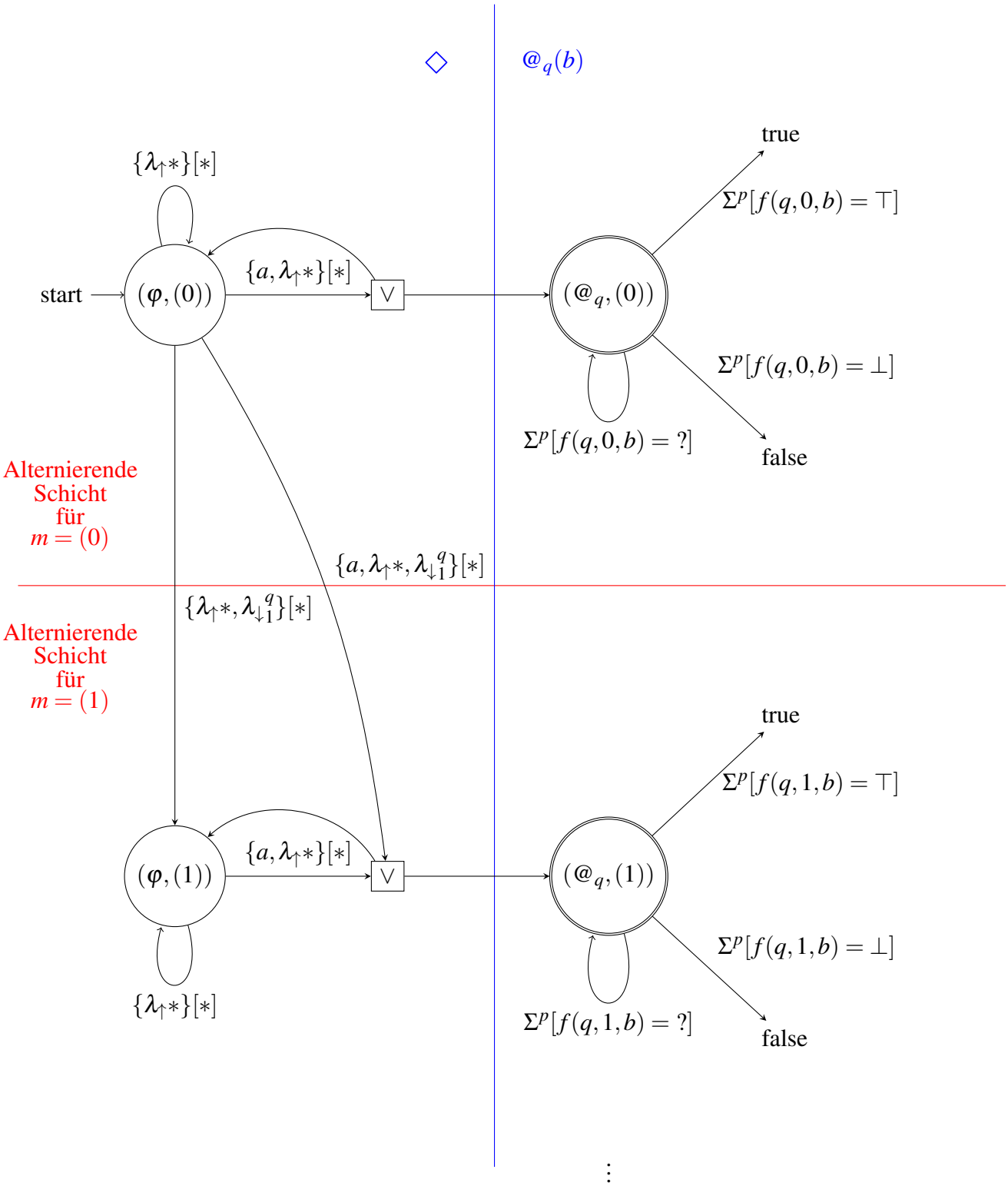


Abbildung 4.3: Der Formel-IABA A^φ, der sich für $@_p\varphi$ ergibt. Die rote Linie trennt den Automaten in Teile mit verschiedenem m und die blaue Linie beschreibt, wo welcher Teil von φ in dem Automaten repräsentiert wird.

Der Automat $A^{\varphi} = (\Sigma^p, Q_A, q_{0A}, \delta_A, F_A)$ ist in Abbildung 4.3 auf der vorherigen Seite dargestellt. Solange kein Wartepunkt gesehen wird, verbleibt der Automat in dem oberen Teilautomaten, in dem $m_1 = 0$ gilt. Befindet sich der Automat in $(\varphi, (0))$ und sieht den Wartepunkt auf q mit der ID 1, so wechselt er in den nächsten Teilautomaten, in dem $m_1 = 1$ gilt und in den Bedingungen ein anderer Wert abgefragt wird. Die Punkte unter dem unteren Teilautomaten beschreiben die weiteren Teilautomaten für alle anderen Wartepunkte. Wird ein Wartepunkt mit einer anderen ID gesehen, so wechselt A in den entsprechenden Teilautomaten.

Die ABAs ${S^{\varphi}}_i^1 = (\Sigma^p, Q_i^1, {q_0}_i^1, \delta_i^1, F_i^1)$ werden in diesem Beispiel nicht für $@_p\varphi$ gezeigt. Die Konstruktion für die Sendepunkt-IABAs wird stattdessen an der Konstruktion der Automaten für die Teilformel $@_q(b)$ erläutert.

Wie beschrieben wurde, müssen auch alle entfernten Teilformeln von $@_p\varphi$ einzeln umgewandelt werden, damit später aus den entstehenden Automaten die Werte für die Nachrichten generiert werden. Aus diesem Grund wird als nächstes $@_q(b)$ in IABAs umgewandelt und an dieser Formel die Entstehung der Sendepunkt-IABAs erläutert.

$@_q(b)$ wird in ein Tupel aus IABAs, (A^b, S^b), umgewandelt. Dafür gilt diesmal $n(p) = 1$.

In Abbildung 4.4 auf der nächsten Seite ist der IABA für die ${S^b}_i^1 = (\Sigma^q, {Q^b}_i^1, {q_0^b}_i^1, {\delta^b}_i^1, {F^b}_i^1)$ dargestellt. A^b entspricht den Zuständen $(b, (x))$ mit $x \in \mathbb{N}$ sowie true und false und den Transitionen zwischen diesen Zuständen. A^b ist zur besseren Übersicht durch die blaue Linie von den Wartezuständen abgetrennt und daher ist dieser nicht noch einmal extra dargestellt. Die rote Linie trennt wieder die Teilautomaten mit verschiedenem m.

Der Automat wartet so lange in einem Zustand $(\lambda, (x))$ mit $x \in \mathbb{N}$, bis der entsprechende Sendepunkt gesehen wird. Bei einem Automaten ${S^b}_i^1$ ist dies $\lambda_{\uparrow_i^{n^{-1}(1)}} = \lambda_{\uparrow_i^p}$ mit $i \in \mathbb{N}$, weil $n^{-1}(1) = p$ gilt. Sobald der Sendepunkt gefunden wird, wird der Rest der aktuellen Eingabe betrachtet und in den entsprechenden Teil des Teilautomaten A^b gegangen. Wird kein Sendepunkt, dafür aber ein Wartepunkt gesehen, so bewegt sich der Automat in den nächsten Wartezustand, um sich den Wartepunkt zu merken. Die Teilautomaten für die weiteren Wartepunkt wurden nicht eingezeichnet, verhalten sich aber äquivalent. Wird der Sende- und ein Wartepunkt zur selben Zeit gesehen, so wechselt der Automat in den passenden Zustand des A^b entsprechenden Teilautomaten, der sich zusätzlich den korrekten Index des gesehenen Wartepunktes merkt. Das Merken der Wartepunkte in den ${S^b}_i^1$ ist nötig, falls in $@_q(b)$ noch ein weiterer @-Operator vorkommen würde.

Durch das Warten auf den Sendepunkt wird die Auswertung der Formel b in den IABAs ${S^b}_i^1$ verzögert und danach normal ausgeführt. In diesem Beispiel sind die Zustände $(b, (x))$ für ein $x \in \mathbb{N}$ überflüssig, aber trotzdem mit eingezeichnet, um zu zeigen, wie A^b aussehen würde.

4.3.4 IABAs zu INBAs

Der nächste Schritt der Umwandlung ist es, alle IABAs aus $(A^{\varphi}, S^{\varphi})$ in INBAs zu transformieren, wodurch das Tupel $(A_N^{\varphi}, S_N^{\varphi})$ aus einem Formel-INBA A_N^{φ} und mehreren Sendepunkt-INBAs aus S_N^{φ} mit

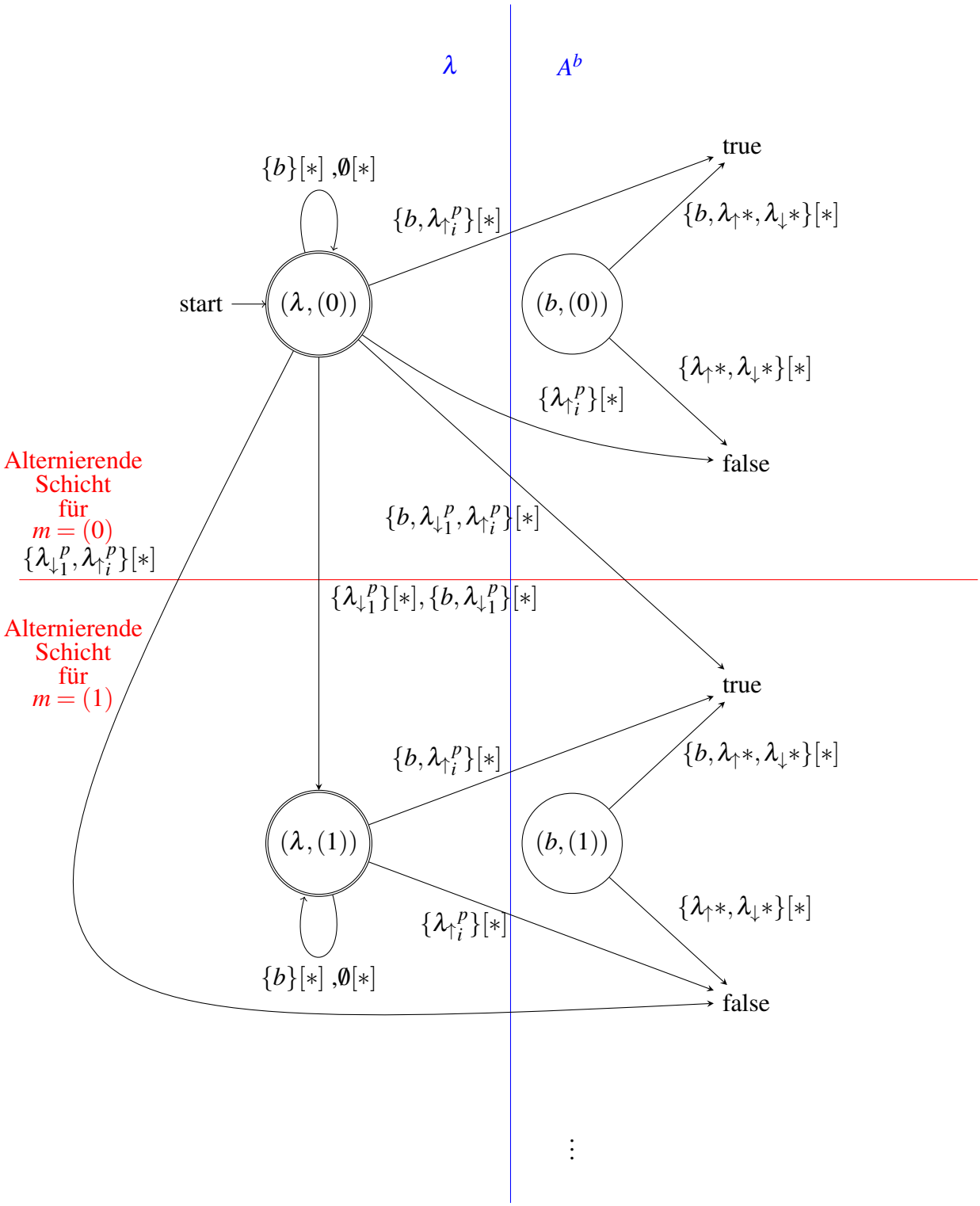


Abbildung 4.4: Die IABAs $S^{b}{}_{i}^{1}$, die sich für $@_q(b)$ ergeben. Die rote Linie trennt den Automaten wieder in Teile mit verschiedenem m und die blaue Linie trennt diesmal die Wartezustände von dem, dem Formel-IABA A^b gleichenden, Teilautomaten ab, der bei Auftreten des gesuchten Sendepunktes betreten wird.

$$S_N^{\varphi} = S_{N1}^{\varphi 1}, S_{N1}^{\varphi 2}, \ldots, S_{N1}^{\varphi |P|-1}, S_{N2}^{\varphi 1}, S_{N2}^{\varphi 2}, \ldots, S_{N2}^{\varphi |P|-1}, S_{N3}^{\varphi 1}, S_{N3}^{\varphi 2}, \ldots, S_{N3}^{\varphi |P|-1}, \ldots$$

entsteht. Dies wird auf die übliche Weise mittels einer Potenzmengenkonstruktion für die Umwandlung von einem ABA in einen NBA gemacht, wie in [BLS11] beschrieben. Die Zustände eines entstehenden INBAs sind immer ein 2-Tupel aus Mengen von Zuständen, in denen sich der IABA, aus dem der INBA entstanden ist, zu diesem Zeitpunkt gleichzeitig befindet. Dabei enthält die erste Menge des Tupels die Zustände des IABAs, für die noch ein akzeptierender Zustand gefunden werden muss und die zweite Menge Zustände des IABA, für die schon ein akzeptierender Zustand gesehen wurde. Die beiden Mengen sind immer disjunkt. Diese Transformation besitzt für IABAs aufgrund der unendlich großen Zustandsmenge eine unendlich große Laufzeit, weshalb sie praktisch nicht anwendbar ist. Aus diesem Grund wird in Abschnitt 4.4 auf Seite 93 bei der Beschränkung der Zustandsmenge der Automaten auf eine endliche Größe auch eine Möglichkeit gefunden, die Potenzmengenkonstruktion nur auf einen endlich großen Teilautomaten anzuwenden.

Da die IABAs aus der vorher beschriebenen, speziellen Struktur aus alternierenden Schichten und Übergängen zwischen alternierenden Schichten mittels Wartepunkten bestehen, ergibt sich auch nach der Potenzmengenkonstruktion zur Umwandlung der IABAs in INBAs wieder eine ähnliche Struktur. Um diese zu betrachten muss zuerst eine Definition für nicht-deterministische Schichten angegeben werden.

Definition 4.15 (Nicht-deterministische Schicht)**.** *Eine Menge von Zuständen $Q_L \subseteq Q$ von einem INBA $(\Sigma, Q, q_0, \delta, F)$, der aus einem IABA entstanden ist, wird nicht-deterministische Schicht (kurz: Schicht) genannt, wenn es ein m gibt, so dass die folgenden Bedingungen erfüllt sind:*

- *Für alle (ψ, m') aus einem $q \in Q_L$ gilt $m \geq m'$,*
- *für alle $q \in Q_L$ gibt es mindestens ein $(\psi, m) \in q$,*
- *sei $\hat{Q}_L$ die Menge, welche Q_L entspricht, wenn aus allen $q \in Q_L$ alle IABA-Zustände (ψ, m) entfernt wurden. Für alle $q, q' \in \hat{Q}_L$ gilt $q = q'$ und für alle (ψ', m') aus einem $\hat{q} \in \hat{Q}_L$ gilt $\psi' \in fSDTL$ und*
- *Q_L ist eindeutig und maximal, es gibt also keine Schicht Q'_L mit $Q_L \subset Q'_L$ oder einem $q' \in Q'_L$ mit $q' \in Q_L$.*

wobei für $\geq$: $\mathbb{N}^{|P|-1} \times \mathbb{N}^{|P|-1}$ gilt

$$a \geq b \text{ gdw. } \forall i \in \{1, 2, 3, \ldots, |P| - 1\} : a_i \geq b_i.$$

Für eine nicht-deterministische Schicht Q_L ergibt sich die Transitionsfunktion $\delta_L : Q_L \times \Sigma \to 2^{Q_L}$ wie folgt:

$$\forall q \in Q_L : \delta_L(q, \sigma) = \delta(q, \sigma) \Leftrightarrow \delta(q, \sigma) \in 2^{Q_L}$$

Außerdem ergibt sich die Menge der akzeptierenden Zustände $F_L \subseteq F$ in Q_L durch

$$\forall q \in Q_L : q \in F_L \Leftrightarrow q \in F$$

Die Änderungen in der Definition der nicht-deterministischen Schichten zur Definition der alternierenden Schichten ergeben sich, weil sich die INBAs die aktuellen Zustände der IABAs in den Zuständen merken. Dadurch ergeben sich die Schichten nicht indem alle Zustände in einer Schicht dasselbe Wartepunkttupel m besitzen. Stattdessen muss in den INBAs unterschieden werden, welche Zustände des IABAs in einem Zustand eines INBAs sind. Aus diesem Grund gibt es auch deutlich mehr Bedingungen als in der Definition der alternierenden Schichten. Die ersten beiden Bedingungen verlangen, dass es unter den IABA-Zuständen jedes Zustandes einer Schicht nur zwei Arten von Zuständen gibt. Die erste Art sind Zustände, die dasselbe, größte Wartepunkttupel in der Schicht besitzen. Von diesen gibt es in jedem Element der Schicht mindestens einen. Sie repräsentieren die neuste Schicht, in der sich der IABA befindet, wenn sich der INBA in dieser Schicht befindet. Die zweite Art von IABA-Zuständen sind jene, die ein älteres Wartepunkttupel besitzen und entfernte Formeln enthalten. Sie repräsentieren wiederum die Zustände älterer Schichten der IABAs, in denen noch auf einen endgültigen Wahrheitswert aus einer Nachricht gewartet wird. Die dritte Bedingung bedeutet, dass alle Elemente einer Schicht dieselben IABA-Zustände mit veralteten Wartepunkttupeln enthalten und die vierte Bedingung sagt, dass eine Schicht immer maximal groß ist und alle Schichten disjunkt sind.

Wie vorher erwähnt sollen nun der Formel-INBA A_N^φ und die Sendepunkt-INBAs $S_{Ni}^{\varphi\,j}$ mittels nicht-deterministischer Schichten betrachtet werden. In dem INBA A_N^φ sind auch die Schichten und Übergänge zwischen diesen Schichten vorhanden, die es auch in dem IABA A^φ gibt, nur, dass auf allen alternierenden Schichten lokal die vorher beschriebene Potenzmengenkonstruktion ausgeführt wurde, um sie in nicht-deterministische Schichten umzuwandeln. In diesen Schichten enthält jeder Zustand nur Zustände des IABAs mit demselben Wartepunkttupel. Es gibt allerdings noch weitere Schichten, da man in dem IABA A^φ auch in mehreren Zuständen gleichzeitig sein kann, die sich in verschiedenen Schichten befinden. Diese Schichten werden im Folgenden als Zwischenschichten bezeichnet, sind aber nicht-deterministische Schichten. In den Zwischenschichten enthält jeder Zustand nicht mehr nur Zustände des IABAs A^φ mit demselben Wartepunkttupel, sondern auch Zustände mit älteren Wartepunkttupeln, wie im vorherigen Absatz beschrieben. Da aber alle Zustände des IABAs mit einem älteren Wartepunkttupel in den Zuständen einer Schicht dieselben sein müssen, sind die Zwischenschichten isomorph zu den anderen nicht-deterministischen Schichten in A_N^φ, weil innerhalb einer Schicht nur Fortschritt für die Zustände des IABAs A^φ mit dem neusten Wartepunkttupel in den Zuständen der Schicht gemacht werden kann. Aus diesem Grund haben alle Schichten und Zwischenschichten in dem INBA A_N^φ, wie auch in dem IABA A^φ, dieselbe Struktur. In den INBAs aus S_N^φ sind nun nicht mehr alle Schichten isomorph, da die Wartezustände exakt so aufgebaut sind wie in den IABAs aus S^φ. Aus diesem Grund würde jeder Wartezustand nur zu einer Schicht gehören und in den Zwischenschichten fehlen, wodurch es zwei Typen von Schichten gibt. Betrachtet man allerdings wieder nur den Teilautomaten in den INBAs aus S_N^φ, in den durch den entsprechenden Sendepunkt aus den Wartezuständen gewechselt wird, so entspricht dieser wiederum A_N^φ.

Es ergibt sich allerdings noch die Änderung, dass zwischen den Zwischenschichten des INBAs A_N^φ auch ohne unbedingt einen Wartepunkt zu sehen gewechselt werden kann. Dies entsteht aus zwei Gründen. Zum Einen, weil auch für die Zustände des IABAs A^φ mit entfernten Formeln und mit älteren Wartepunkttupeln Fortschritt durch das Eintreffen einer Nachricht mit den benötigten Wahrheitswerten gemacht werden kann. Passiert dies, so wird von einer Zwischenschicht in eine andere Schicht des INBAs S_N^φ gewechselt. Zum Anderen kann es passieren, dass alle IABA-Zustände mit aktuellen Wartepunkttupel in einem Zustand einer Schicht von S_N^φ durch ein gelesenes Zeichen in true enden. Dadurch verschwinden sie aus dem INBA-Zustand und ein IABA-Zustand mit einem älteren Wartepunkttupel ist dann der Zustand mit dem neusten Wartepunkttupel in dem sich ergebenen INBA-Zustand. Dadurch befindet sich der INBA danach in einer anderen Schicht. Dabei kann es passieren, dass in den INBA-Zustand gewechselt wird, der keinen IABA-Zustand mehr enthält. Dieser ist nach der Potenzmengenkonstruktion ein akzeptierender Zustand, aus dem es nicht mehr möglich ist, heraus zu wechseln. Dieser Zustand gehört außerdem zu keiner Schicht.

4.3.5 INBAs zu NIAs

Als nächstes werden in allen INBAs aus $(A_N^\varphi, S_N^\varphi)$ alle Zustände gesucht, von denen aus noch ein Wort akzeptiert werden kann. Alle diese Zustände werden dann als akzeptierend markiert, wodurch nur die Zustände nicht akzeptierend sind, von denen aus auf keinen Fall mehr akzeptiert werden kann. Dies wird mittels eines Leerheitstest für jeden Zustand erreicht. Die, durch den Leerheitstest entstehenden, Automaten ergeben das Tupel $(A_F^\varphi, S_F^\varphi)$ mit

$$S_F^\varphi = S_{F1}^{\varphi 1}, S_{F1}^{\varphi 2}, \ldots, S_{F1}^{\varphi |P|-1}, S_{F2}^{\varphi 1}, S_{F2}^{\varphi 2}, \ldots, S_{F2}^{\varphi |P|-1}, S_{F3}^{\varphi 1}, S_{F3}^{\varphi 2}, \ldots, S_{F3}^{\varphi |P|-1}, \ldots$$

und werden dann als NIAs interpretiert. Dabei wird A_F^φ wieder als Formel-NIA und die $S_{Fi}^{\varphi j}$ als Sendepunkt-NIAs bezeichnet. Der Leerheitstest für einen INBA $(\Sigma, Q, q_0, \delta, F)$ wird durchgeführt, indem F durch F' mit

$$F' = \{q \mid q \in Q \wedge \exists w \in \Sigma^\omega : w \in \mathcal{L}((\Sigma, Q, q, \delta, F))\}$$

ersetzt wird. Bei dem Leerheitstest wird von jedem Zustand aus untersucht, ob noch ein Wort akzeptiert werden kann. Wird eines gefunden, so wird der Zustand als akzeptierend markiert. Dadurch sind in dem INBA nach dem Leerheitstest alle Zustände akzeptierend, von denen aus noch akzeptiert werden kann.

Wie auch die Potenzmengenkonstruktion ist auch der Leerheitstest auf unendlich großen Automaten praktisch nicht anwendbar. Aus diesem Grund wird bei der Beschränkung der Automaten auf endliche Größen in Abschnitt 4.4 auf Seite 93 auch eine Möglichkeit gefunden, den Leerheitstest nur auf einem endlichen Teilautomaten auszuführen.

4.3.6 NIAs zu DIAs

Die NIAs aus $(A_F^\varphi, S_F^\varphi)$ werden als nächstes wiederum auf die übliche Weise durch eine Potenzmengenkonstruktion in DIAs umgewandelt, wie in Unterabschnitt 4.3.1 auf Seite 68 beschrieben, wodurch sich das Tupel $(A_D^\varphi, S_D^\varphi)$ mit

$$S_D^\varphi = S_{D1}^{\varphi 1}, S_{D1}^{\varphi 2}, \ldots, S_{D1}^{\varphi |P|-1}, S_{D2}^{\varphi 1}, S_{D2}^{\varphi 2}, \ldots, S_{D2}^{\varphi |P|-1}, S_{D3}^{\varphi 1}, S_{D3}^{\varphi 2}, \ldots, S_{D3}^{\varphi |P|-1}, \ldots$$

aus dem Formel-DIA A_D^φ und den Sendepunkt-DIAs $S_{Di}^{\varphi j}$ ergibt. Die Zustände eines entstehenden DIAs sind immer eine Menge von Zuständen, in denen sich der NIA, aus dem der DIA entstanden ist, nicht-deterministisch zu einem Zeitpunkt befinden kann.

Da auch die NIAs unendlich groß sind ergeben sich für die hier genutzte Potenzmengenkonstruktion dieselben Probleme wie für die aus Unterabschnitt 4.3.4 auf Seite 79.

Da die NIAs auch eine Struktur aus Schichten und Übergängen aufweisen, denn sie sind von den Zuständen und Transitionen her identisch zu den INBAs, ergibt sich auch nach der Potenzmengenkonstruktion zur Umwandlung in die DIAs wieder eine solche Struktur. Um diese zu betrachten wird wiederum zuerst eine Definition für deterministische Schichten gegeben.

Definition 4.16 (Deterministische Schicht)**.** *Eine Menge von Zuständen $Q_L \subseteq Q$ von einem DIA $(\Sigma, Q, q_0, \delta, F)$, der aus einem, aus einem IABA entstandenen, NIA entstanden ist, wird nicht-deterministische Schicht (kurz: Schicht) genannt, wenn es ein m gibt, so dass die folgenden Bedingungen erfüllt sind:*

- *Für alle IABA-Zustände (ψ, m') aus den NIA-Zuständen aus einem $q \in Q_L$ gilt $m \geq m'$,*
- *es gibt mindestens einen NIA-Zustand aus einem $q \in Q_L$ der einen IABA-Zustand (ψ, m) enthält,*
- *sei $\hat{Q}_L$ die Menge, welche Q_L entspricht, wenn aus den NIA-Zuständen aus allen $q \in Q_L$ alle IABA-Zustände (ψ, m) entfernt wurden. Für alle $q, q' \in \hat{Q}_L$ gilt $q = q'$ und für alle (ψ', m') aus NIA-Zuständen eines $\hat{q} \in \hat{Q}_L$ gilt $\psi' \in$ fSDTL und*
- *Q_L ist eindeutig und maximal, es gibt also keine Schicht Q'_L mit $Q_L \subset Q'_L$ oder einem $q' \in Q'_L$ mit $q' \in Q_L$.*

wobei die Relation $\geq$ wie in der Definition der nicht-deterministischen Schichten (Definition 4.15 auf Seite 81) definiert ist.

Für eine deterministische Schicht Q_L ergibt sich die Transitionsfunktion $\delta_L : Q_L \times \Sigma \to Q_L$ wie folgt:

$$\forall q \in Q_L : \delta_L(q, \sigma) = \delta(q, \sigma) \Leftrightarrow \delta(q, \sigma) \in Q_L$$

Außerdem ergibt sich die Menge der akzeptierenden Zustände $F_L \subseteq F$ in Q_L durch

$$\forall q \in Q_L : q \in F_L \Leftrightarrow q \in F$$

Die Änderungen in der Definition der deterministischen Schichten zu der Definition der nicht-deterministischen Schichten entstehen wieder durch die Änderungen an den Zuständen des Automaten. Die Zustände eines DIAs sind Mengen von NIA-Zuständen. Die erste Bedingung verlangt, dass es ein Wartepunkttupel gibt, welches die Wartepunkttupel der IABA-Zustände in den DIA-Zuständen einer Schicht nach oben begrenzt. Die zweite Bedingung verlangt, dass es in mindestens einem NIA-Zustand jedes Zustandes einer Schicht einen IABA-Zustand mit diesem neusten Wartepunkttupel gibt. Die dritte Bedingung drückt aus, dass alle DIA-Zustände einer Schicht exakt gleich sind, wenn alle IABA-Zustände mit dem neusten Wartepunkttupel aus den DIA-Zuständen entfernt werden. Dadurch können sich innerhalb einer Schicht nur die IABA-Zustände mit dem neusten Wartepunkttupel ändern, wie es auch schon in den nicht-deterministischen und alternierenden Schichten der Fall war. Weiter bedeutet die dritte Bedingung, dass alle IABA-Zustände, die nicht das neuste Wartepunkttupel enthalten, entfernte Formeln enthalten. Die letzte Bedingung sagt wieder aus, dass eine Schicht immer maximal groß ist und dass alle Schichten disjunkt sind.

Betrachtet man den DIA A_D^φ mittels Schichten, so kann man ihn auf dieselbe Art beschreiben, wie den INBA A_N^φ. Aus allen nicht-deterministischen Schichten des NIAs entsteht durch eine Potenzmengenkonstruktion auf einer der Schichten jeweils eine deterministische Schichten und genau wie bei den INBA-Schichten beschrieben entstehen außerdem wieder Zwischenschichten. In dem DIA A_D^φ existieren durch die Potenzmengenkonstruktion zusätzlich noch zwei Zustände, die in keiner Schicht sind. Zum Einen der Zustand $\emptyset$, der keinen NIA-Zustand mehr enthält, welches eine nicht-akzeptierende Senke ist. In diesen wechselt A_D^φ, wenn alle NIA-Zustände aus einem seiner Zustände entfernt wurden, indem es von den NIA-Zuständen aus für ein Zeichen keine Transitionen gab. Zum Anderen noch den Zustand $\{(\emptyset,\emptyset)\}$. In diesen Zustand wechselt A_D^φ, wenn alle IABA-Zustände aus den NIA-Zuständen in true endeten. Für die DIAs aus S_D^φ gilt das Gleiche wie für die INBAs aus S_N^φ, außer, dass die Automaten aus S_D^φ nun A_D^φ als Teilautomaten besitzen.

In dem Lemma soll nun gezeigt werden, dass ein DIA, der auf dem vorher beschriebenen Weg während dieser Umwandlung entsteht, immer diese Schichtenstruktur besitzt.

Lemma 4.17 (Schichtung für DIAs aus $(A_D^\varphi, S_D^\varphi)$)**.** *Für die DIAs aus $(A_D^\varphi, S_D^\varphi)$ gilt folgendes:*

1. *Es gibt eine gültige Schichtung für A_D^φ, es ist also möglich die Zustandsmenge Q des DIAs A_D^φ ohne die Zustände $\emptyset$ und $\{(\emptyset,\emptyset)\}$ so in Teilmengen $Q_1, Q_2, Q_3, \ldots, \subseteq Q \setminus \{\emptyset, \{(\emptyset,\emptyset)\}\}$ aufzuteilen, dass $Q_1, Q_2, Q_3, \ldots$ deterministische Schichten sind und $\bigcup_{i\in\mathbb{N}} Q_i = Q \setminus \{\emptyset, \{(\emptyset,\emptyset)\}\}$ gilt.*
2. *Alle Schichten des DIAs A_D^φ sind isomorph und besitzen nur endlich viele Zustände.*
3. *Wird der, dem Formel-IABA A^φ entsprechende, Teilautomat in den Sendepunkt-IABAs aus S^φ durch den Formel-DIA A_D^φ ersetzt, so ergeben sich die Sendepunkt-DIAs aus S_D^φ.*

Beweis. Für 1. muss gezeigt werden, dass es Schichten gibt, so dass die Bedingungen der deterministischen Schichten erfüllt sind und jeder Zustand außer den Zuständen $\emptyset$ und $\{(\emptyset,\emptyset)\}$ in

genau einer Schicht ist. Um dies zu zeigen kann der gerichtete, azyklische Graph, der als Lauf eines IABAs entsteht, betrachtet werden. In dem Lauf eines IABAs wird in jedem Schritt in jedem Pfad Fortschritt gemacht. Das heißt, für jeden Zustand auf einer Ebene in dem Lauf wird für ein Eingabesymbol durch die Transitionsfunktion in Zustände in der nächsten Ebene gewechselt. Aus diesem Grund kann es auf einer Ebene eines Laufes von A^φ nur Zustände geben, die entweder das aktuellste Wartepunkttupel oder eine entfernte Teilformel als Formel besitzen, denn nur für Zustände mit entfernten Formeln wird durch einen Wartepunkt das Wartepunkttupel nicht aktualisiert.

Deshalb kann es in den Zuständen des DIAs A_D^φ nur Zustände (ψ, m) des IABAs A^φ mit alten Wartepunkttupeln geben, bei denen ψ eine fSDTL-Formel ist, da für alle anderen bei Auftreten eines Wartepunktes in einen Zustand mit aktualisiertem Wartepunkttupel gewechselt wird. Jeder Zustand des NIAs A_N^φ in einem Zustand des DIAs A_D^φ kann also nur aus Zuständen des IABAs A^φ mit aktuellem Wartepunkttupel sowie aus einer Kombination von Zuständen (ψ', m') bestehen, wobei m' ein veraltetes Wartepunkttupel ist und $\psi' \in$ fSDTL. Deterministische Schichten für A_D^φ können deswegen erstellt werden, indem alle Zustände zu einer Schicht gehören, die ohne die IABA-Zustände mit aktuellem Wartepunkttupel gleich sind und in denen nur die IABA-Zustände mit aktuellem Wartepunkttupel variieren. Dabei ergibt die Vereinigung der entstehenden Schichten wieder $Q \setminus \{\emptyset, \{(\emptyset, \emptyset)\}\}$, weil es für jede Kombination von verschiedenen Mengen aus IABA-Zuständen (ψ', m') mit $\psi' \in$ fSDTL eine Schicht in dem DIA A_D^φ gibt, in der jeder Zustand NIA-Zustände aus A_N^φ enthält, so dass es für jede Menge der Kombination mindestens einen NIA-Zustand gibt, der genau die IABA-Zustände aus der Menge als IABA-Zustände mit veraltetem Wartepunkttupel enthält. Da eine Schicht für eine feste Kombination von verschiedenen Mengen aus IABA-Zuständen immer genau alle Zustände von A_D^φ enthält, die NIA-Zustände aus A_N^φ enthalten, so dass die vorherige Bedingung erfüllt ist, ergibt sich durch die Vereinigung der Schichten $Q \setminus \{\emptyset, \{(\emptyset, \emptyset)\}\}$. Damit gibt es eine gültige Schichtung des DIAs A_D^φ.

Um 2. zu zeigen, wird zuerst die Schicht Q_0 des IABAs A^φ betrachtet, die alle Zustände $(\psi, \vec{0})$ aus A^φ enthält. Wird nun nur diese Schicht betrachtet, so haben Wartepunkte keine Auswirkungen auf die Transitionen. Aus diesem Grund entspricht diese Schicht dem entstehenden ABA aus der Umwandlung für LTL mit zusätzlichem @-Operator. Da eine entfernte Formel für einen festen Wartepunkt nur einen Zustand erzeugt, ist Q_0 endlich groß.

Auf Basis der Struktur der Schichten, die in 1. angegeben wurde, werden im Folgenden die Schichten des Formel-DIAs A_D^φ betrachtet. Als erstes wird die Schicht Q_s von A_D^φ untersucht, die den Startzustand enthält. Diese besitzt in ihren Zuständen keine Zustände des IABAs A^φ mit veraltetem Wartepunkttupel, denn bevor ein Zeichen gelesen wird kann noch kein Wartepunkt aufgetreten sein, weshalb alle IABA-Zustände in dem Startzustand das Wartepunkttupel $\vec{0}$ enthalten und dies damit auch für alle anderen Zustände aus Q_s gilt. Die Zustände der Schicht Q_s bestehen deshalb nur aus IABA-Zuständen aus der alternierenden Schicht Q_0 des IABAs A^φ. Aus diesem Grund muss es sich bei der Schicht Q_s des Formel-DIAs A_D^φ um den DFA handeln, der aus der Umwandlung von Q_0 entsteht. Damit ist diese Schicht des DIAs A_D^φ nur endlich groß.

Da die IABA-Zustände mit veralteten Wartepunkttupeln in einer Schicht in allen Zuständen gleich vorhanden sind, sind für die Struktur einer Schicht nur die IABA-Zustände aus A^φ mit aktuellem

Wartepunkttupel interessant. Lässt man die IABA-Zustände mit veraltetem Wartepunkttupel weg, so muss es für jeden Zustand q der betrachteten Schicht einen Zustand aus Q_s geben, der identisch zu q ist, wenn alle Wartepunkttupel in q durch $\vec{0}$ ersetzt werden. Andernfalls hätte q nicht entstehen können, weil in dem IABA die Wartepunkte in einem Zeichen keinen Einfluss auf die Veränderung der Formel in einem Zustand haben. Das heißt, eine Schicht kann nicht mehr Zustände besitzen als Q_s. Besitzt eine Schicht von A_D^φ weniger Zustände, so können die übrigen Zustände hinzugefügt werden, auch wenn sie unerreichbar sind. Außerdem gibt es in jeder Schicht identische Transitionen, weil für eine Formel eines IABA-Zustandes in einem Zustand einer Schicht des DIAs A_D^φ unabhängig des Wartepunkttupels für ein Zeichen des Eingabewortes die Formel des Folgezustandes bestimmt wird. Damit sind alle Schichten von A_D^φ isomorph und endlich.

Für 3. werden zunächst die Wartezustände, also die Zustände (λ, m) mit $m \in \mathbb{N}^{|P|-1}$, in den IABAs aus S^φ betrachtet. Die Transitionen von einem Wartezustand zu einem Wartezustand sind bereits deterministisch. Außerdem ist es nicht möglich in einen Wartezustand zurückzukehren, wenn die Wartezustände einmal verlassen wurden, weil der Teilautomat, der dem Formel-IABA A^φ entspricht und beim Verlassen der Wartezustände betreten wird, die Wartezustände nicht enthält und es deswegen in seiner Transitionsfunktion keine Transition in die Wartezustände geben kann. Aus diesem Grund werden die Wartezustände und die Transitionen zwischen diesen bei der Umwandlung der IABAs aus S^φ in die DIAs aus S_D^φ nicht verändert. Alle übrigen Zustände in den IABAs aus S^φ sind die des Formel-IABA A^φ entsprechenden Teilautomaten. Da es nicht möglich ist aus A^φ hinaus zu wechseln, wird er wie gewohnt in einen DIA umgewandelt und deshalb ergibt sich als Teilautomat in den DIAs aus S_D^φ der Formel-DIA A_D^φ. □

Führt man dieselbe Prozedur für $\neg\varphi$ durch, so erhält man am Ende auch ein Tupel aus DIAs, $(A_D^{\neg\varphi}, S_D^{\neg\varphi})$, mit

$$S_D^{\neg\varphi} = S_D^{\neg\varphi 1}{}_1, S_D^{\neg\varphi 2}{}_1, \ldots, S_D^{\neg\varphi |P|-1}{}_1, S_D^{\neg\varphi 1}{}_2, S_D^{\neg\varphi 2}{}_2, \ldots, S_D^{\neg\varphi |P|-1}{}_2, S_D^{\neg\varphi 1}{}_3, S_D^{\neg\varphi 2}{}_3, \ldots, S_D^{\neg\varphi |P|-1}{}_3, \ldots.$$

Durch die beiden entstandenen Tupel aus DIAs weiß man nun, wie bei den beiden entstehenden DFAs in der Umwandlung für LTL_3, durch den Formel-DIA A_D^φ bzw. durch die Sendepunkt-DIAs $S_{Di}^{\varphi\, j}$, welche Zustände auf keinen Fall mehr akzeptieren können und durch den Formel-DIA $A_D^{\neg\varphi}$ bzw. durch die Sendepunkt-DIAs $S_D^{\neg\varphi\, j}{}_i$, welche auf keinen Fall nicht akzeptieren können, also welche Zustände immer akzeptieren. Aus den jeweils zusammengehörigen DIAs können Paare gebildet werden, aus denen dann eine ISM erstellt werden kann. Es entsteht daher das Paar $(A_D^\varphi, A_D^{\neg\varphi})$ sowie für jedes $i \in \mathbb{N}$ und $j \in \{1, 2, \ldots, |P|-1\}$ die Paare $(S_{Di}^{\varphi\, j}, S_D^{\neg\varphi\, j}{}_i)$.

Als nächstes werden zwei Lemmata für die Eigenschaften der Paare gezeigt. Diese folgen dem Beweis aus [BLS11].

Lemma 4.18 (Verhältnis der fSDTL-Auswertung und dem DIA-Paar $(A_D^\varphi, A_D^{\neg\varphi})$)**.** *Sei $w \in \Sigma^*$. Es gilt*

$$[\![w \models @_p\varphi]\!]_{fSDTL} = \begin{cases} \top & gdw.\ w \notin \mathcal{L}(A_D^{\neg\varphi}) \\ \bot & gdw.\ w \notin \mathcal{L}(A_D^{\varphi}) \\ ? & gdw.\ w \in (\mathcal{L}(A_D^{\neg\varphi}) \cap \mathcal{L}(A_D^{\varphi})) \end{cases}$$

unter der Annahme, dass die Werte, die f zurückgibt, korrekt sind.

Beweis. Seien im Folgenden $A_F^{\varphi} = (\Sigma, Q_F^{\varphi}, q0_F^{\varphi}, \delta_F^{\varphi}, F_F^{\varphi})$ und $A_F^{\neg\varphi} = (\Sigma, Q_F^{\neg\varphi}, q0_F^{\neg\varphi}, \delta_F^{\neg\varphi}, F_F^{\neg\varphi})$ die NIAs, aus denen A_D^{φ} und $A_D^{\neg\varphi}$ entstanden sind. Dann gilt $\mathcal{L}(A_D^{\varphi}) = \mathcal{L}(A_F^{\varphi})$ und $\mathcal{L}(A_D^{\neg\varphi}) = \mathcal{L}(A_F^{\neg\varphi})$. Weiter seien $A_N^{\varphi} = (\Sigma, Q_N^{\varphi}, q0_N^{\varphi}, \delta_N^{\varphi}, F_N^{\varphi})$ und $A_N^{\neg\varphi} = (\Sigma, Q_N^{\neg\varphi}, q0_N^{\neg\varphi}, \delta_N^{\neg\varphi}, F_N^{\neg\varphi})$ die INBAs, aus denen A_F^{φ} und $A_F^{\neg\varphi}$ durch den Leerheitstest entstanden sind. Sei weiter $n \in \mathbb{N}$.

Zuerst wird $[\![w \models @_p\varphi]\!]_{\text{fSDTL}} = \top$ gdw. $w \notin \mathcal{L}(A_D^{\neg\varphi})$ gezeigt:

Sei $w \in \Sigma^n$ ein endlicher Präfix. Lässt man den INBA $A_N^{\neg\varphi}$ über w laufen, so wird eine Menge von Zuständen

$$Z = \delta(\delta(\ldots(\delta(\delta(q0_N^{\neg\varphi}, w_0), w_1)), w_{n-2}), w_{n-1})$$

generiert. Wenn $\exists q \in Z : q \in F_F^{\neg\varphi}$ gilt, dann kann ein $u \in \Sigma^{\omega}$ gewählt werden, so dass $wu \in \mathcal{L}(A_N^{\neg\varphi})$. So ein Zustand q existiert nach der Definition gdw. $w \in \mathcal{L}(A_F^{\neg\varphi})$. Wenn nun $w \notin \mathcal{L}(A_F^{\neg\varphi})$ ist, dann wird jede mögliche Verlängerung wu von w von $A_N^{\neg\varphi}$ verworfen. Das heißt, das mit Theorem 4.12 auf Seite 76 $\forall u \in \Sigma^{\omega} : [\![wu_0 \models \varphi]\!]_{\text{fSDTL}_{\omega}} = \top$ und daher auch $[\![w \models \varphi]\!]_{\text{fSDTL}} = \top$ folgt.

Für $[\![w \models @_p\varphi]\!]_{\text{fSDTL}} = \bot$ gdw. $w \notin \mathcal{L}(A_D^{\varphi})$ folgt der Beweis analog, indem φ durch $\neg\varphi$ ausgetauscht wird.

Als letztes wird noch $[\![w \models @_p\varphi]\!]_{\text{fSDTL}} = ?$ gdw. $w \in (\mathcal{L}(A_D^{\neg\varphi}) \cap \mathcal{L}(A_D^{\varphi}))$ gezeigt:

Sei $w \in \Sigma^n$ ein endlicher Präfix. Lässt man den NBA $A_N^{\neg\varphi}$ über w laufen, so wird eine Menge von Zuständen

$$Z^{\neg\varphi} = \delta(\delta(\ldots(\delta(\delta(q0_N^{\neg\varphi}, w_0), w_1)), w_{n-2}), w_{n-1})$$

generiert. Lässt man den NBA A_N^{φ} über w laufen, so wird eine Menge von Zuständen

$$Z^{\varphi} = \delta(\delta(\ldots(\delta(\delta(q0_N^{\neg\varphi}, w_0), w_1)), w_{n-2}), w_{n-1})$$

generiert. Wenn nun $\exists q \in Z^{\neg\varphi} : q \in F_F^{\neg\varphi}$ und $\exists q' \in Z^{\varphi} : q' \in F_F^{\varphi}$ gelten, so können $u, u' \in \Sigma^{\omega}$ gewählt werden, so dass $wu \in \mathcal{L}(A_N^{\neg\varphi})$ und $wu' \in \mathcal{L}(A_N^{\varphi})$. Solche Zustände q und q' existieren gdw. $w \in \mathcal{L}(A_F^{\neg\varphi})$ und $w \in \mathcal{L}(A_F^{\varphi})$, also $w \in (\mathcal{L}(A_F^{\neg\varphi}) \cap \mathcal{L}(A_F^{\varphi}))$. Damit folgt nach Theorem 4.12 auf Seite 76 $[\![wu_0 \models \varphi]\!]_{\text{fSDTL}_{\omega}} \neq \top$ und $[\![wu'_0 \models \varphi]\!]_{\text{fSDTL}_{\omega}} \neq \bot$ und daher insgesamt $[\![w \models \varphi]\!]_{\text{fSDTL}} = ?$ □

Lemma 4.19 (Verhältnis der fSDTL-Auswertung und dem DIA-Paar $(S_{Di}^{\varphi\,j}, S_{D\ i}^{\neg\varphi\,j})$)**.** *Sei $w \in \Sigma^x$, $0 \leq h < x$, q ein Prozess mit $n(q) = j$ und $\lambda_{\uparrow_i^q} \in w_h$. Es gilt*

$$[\![w_{\geq h} \models @_p \varphi]\!]_{fSDTL} = \begin{cases} \top & gdw.\ w \notin \mathcal{L}(S_{D\ i}^{\neg\varphi\,j}) \\ \bot & gdw.\ w \notin \mathcal{L}(S_{Di}^{\varphi\,j}) \\ ? & gdw.\ w \in (\mathcal{L}(S_{D\ i}^{\neg\varphi\,j}) \cap \mathcal{L}(S_{Di}^{\varphi\,j})) \end{cases}$$

unter der Annahme, dass die Werte, die f zurückgibt, korrekt sind.

Beweis. Der Beweis folgt mit Theorem 4.13 auf Seite 77 statt Theorem 4.12 auf Seite 76 analog zu dem Beweis des vorherigen Lemmas. □

Diese beiden Lemmata sind wichtig, um im nächsten Teilabschnitt die Korrektheit der ISMs zu zeigen, die dort entstehen.

4.3.7 DIAs zu ISMs

Aus jedem dieser Paare erzeugt man nun eine ISM und erhält aus den DIAs $(A_D^{\varphi}, S_D^{\varphi})$ und $(A_D^{\neg\varphi}, S_D^{\neg\varphi})$ das neue Tupel aus ISMs $(A'^{\varphi}, S'^{\varphi})$ wobei

$$S'^{\varphi} = S'^{\varphi}{}_1^1, S'^{\varphi}{}_1^2, \ldots, S'^{\varphi}{}_1^{|P|-1}, S'^{\varphi}{}_2^1, S'^{\varphi}{}_2^2, \ldots, S'^{\varphi}{}_2^{|P|-1}, S'^{\varphi}{}_3^1, S'^{\varphi}{}_3^2, \ldots, S'^{\varphi}{}_3^{|P|-1}, \ldots$$

gilt und A'^{φ} als Formel-ISM und die $S'^{\varphi\,j}_i$ als Sendepunkt-ISMs bezeichnet werden. Die ISMs des Tupels $(A'^{\varphi}, S'^{\varphi})$ werden wie folgt konstruiert:

Seien $D^{\varphi} = (\Sigma, Q^{\varphi}, q_0^{\varphi}, \delta^{\varphi}, F^{\varphi})$ und $D^{\neg\varphi} = (\Sigma, Q^{\neg\varphi}, q_0^{\neg\varphi}, \delta^{\neg\varphi}, F^{\neg\varphi})$ zwei DIAs und $(D^{\varphi}, D^{\neg\varphi})$ ein Paar aus den DIAs. Aus diesem Paar wird die gewünschte ISM $I = (\Sigma, \overline{Q}, \overline{q_0}, \overline{\delta}, \mathbb{B}_3, \lambda)$ konstruiert, indem für die Elemente von I folgendes gilt:

- $\overline{Q} = Q^{\varphi} \times Q^{\neg\varphi}$,
- $\overline{q_0} = (q_0^{\varphi}, q_0^{\neg\varphi})$,
- $\overline{\delta}((q,q'),\sigma) = (\delta^{\varphi}(q,\sigma), \delta^{\neg\varphi}(q',\sigma))$ und
- $\lambda((q,q')) = \begin{cases} \top & \text{wenn } q' \notin F^{\neg\varphi} \\ \bot & \text{wenn } q \notin F^{\varphi} \\ ? & \text{sonst} \end{cases}$

Diese Moore-Maschine simuliert die beiden DIAs. Da D^{φ} und $D^{\neg\varphi}$ unendlich viele Zustände haben, gilt dies auch für I. Ein Zustand von I ist ein Tupel der Zustände, in denen sich D^{φ} und $D^{\neg\varphi}$ zu dem Zeitpunkt befinden würden. Da D^{φ} angibt, wann nicht mehr akzeptiert werden kann und $D^{\neg\varphi}$ angibt, wann auf jeden Fall akzeptiert wird, ergibt sich durch die nicht akzeptierenden Zustände der beiden DIAs die Ausgabe der ISM.

Die mit dieser Konstruktion erstellten ISMs $(A'^{\varphi}, S'^{\varphi})$ haben dann für jedes Eingabewort dieselbe Ausgabe wie die, die bei einer Auswertung der Formel durch fSDTL entstehen würde. Dies wird in dem folgenden Theorem gezeigt.

Durch 1. wird in dem Theorem die Korrektheit von der Formel-ISM A'^{φ} gezeigt. Durch 2a. wird gezeigt, dass die Sendepunkt-ISMs ${S'^{\varphi}}_i^j$ nach dem Auftreten des entsprechenden Sendepunktes die korrekten Werte ausgeben und durch 2b. wird gezeigt, dass die Sendepunkt-ISMs vor dem Auftreten des entsprechenden Sendepunktes immer ? ausgeben.

Theorem 4.20 (Korrektheit der ISMs aus $(A'^{\varphi}, S'^{\varphi})$)**.** *Es gilt für die Formel-ISM* $A'^{\varphi} = (\Sigma, Q_{A'^{\varphi}}, q_{0A'^{\varphi}}, \delta_{A'^{\varphi}}, \mathbb{B}_3, \lambda_{A'^{\varphi}})$ *sowie die Sendepunkt-ISMs* ${S'^{\varphi}}_i^j = (\Sigma, Q_{{S'^{\varphi}}_i^j}, q_{0{S'^{\varphi}}_i^j}, \delta_{{S'^{\varphi}}_i^j}, \mathbb{B}_3, \lambda_{{S'^{\varphi}}_i^j})$ *folgendes:*

1. *Sei* $w \in \Sigma^x$, $x \in \mathbb{N}$ *und* $k_0, k_1, \ldots, k_x$ *mit* $k_0 = q_{0A'^{\varphi}}$ *und* $k_l = \delta_{A'^{\varphi}}(k_{l-1}, w_{l-1}), 1 \leq l \leq x$ *der Lauf der Formel-ISM* A'^{φ} *bei Eingabe von w. Es gilt* $[\![w \models @_p\varphi]\!]_{fSDTL} = \lambda_{A'^{\varphi}}(k_x)$.

Sei $w \in \Sigma^x$, $x \in \mathbb{N}$, $0 \leq h < x$, *q ein Prozess mit* $n(q) = j$, $\lambda_{\uparrow_i}^q \in w_h$ *und* $k_0, k_1, \ldots, k_x$ *mit* $k_0 = q_{0{S'^{\varphi}}_i^j}$ *und* $k_l = \delta_{{S'^{\varphi}}_i^j}(k_{l-1}, w_{l-1}), 1 \leq l \leq x$ *der Lauf einer Sendepunkt-ISM* ${S'^{\varphi}}_i^j$ *bei Eingabe von w.*

2a. *Es gilt* $[\![w_{\geq h} \models @_p\varphi]\!]_{fSDTL} = \lambda_{{S'^{\varphi}}_i^j}(k_x)$.

2b. *Es gilt* $\lambda_{{S'^{\varphi}}_i^j}(k_0) = \lambda_{{S'^{\varphi}}_i^j}(k_1) = \cdots = \lambda_{{S'^{\varphi}}_i^j}(k_h) = ?$.

Dabei wird angenommen, dass die Werte, die f zurückgibt, korrekt sind.

Beweis. Der Beweis für 1. und 2a. folgt direkt aus den Lemma 4.18 auf Seite 87 und Lemma 4.19 auf der vorherigen Seite, da die entstehenden ISMs durch ihre Zustände und die Transitionsfunktion nur die DIAs aus den Paaren simulieren und nach den beiden Lemmata die Ausgabefunktionen $\lambda_{A'^{\varphi}}$ und $\lambda_{{S'^{\varphi}}_i^j}$ zu jedem Zeitpunkt denselben Wahrheitswert wie die Auswertung mit der fSDTL-Semantik ausgeben.

2b. gilt, weil der entsprechenden Sendepunkt noch nicht aufgetreten ist, wenn sich die ISM in k_h befindet. Das heißt, jedes $k_0, k_1, \ldots, k_h$ ist einer der ISM-Zustände $(\{(\emptyset, \{(\lambda, m)\})\}, \{(\emptyset, \{(\lambda, m)\})\})$ für ein $m \in \mathbb{N}^{|P|-1}$, weil sich auch die beiden Sendepunkt-IABAs, aus denen die ISM entstanden ist, zu diesem Zeitpunkt in einem Zustand (λ, m) mit $m \in \mathbb{N}^{|P|-1}$ befinden müssen. Jedes (λ, m) ist durch die Definition der Menge der akzeptierenden Zustände der Sendepunkt-IABAs akzeptierend, weshalb auch jeder Zustand vor dem Auftreten des entsprechenden Sendepunktes in einem der Sendepunkt-DIAs die Form $\{(\emptyset, \{(\lambda, m)\})\}$ besitzen muss und deswegen akzeptierend ist. Dadurch ergibt sich nach der Definition der Ausgabefunktion der ISMs $\lambda_{{S'^{\varphi}}_i^j}(k_0) = \lambda_{{S'^{\varphi}}_i^j}(k_1) = \cdots = \lambda_{{S'^{\varphi}}_i^j}(k_h) = ?$. □

Die Elemente des Tupels $(A'^{\varphi}, S'^{\varphi})$ haben dieselbe Funktion wie vorher beschrieben. A'^{φ} stellt die Auswertung der gesamten Formel sowie dessen Ausgabe dar und die Elemente aus S'^{φ} die Auswertungen ab den verschiedenen Sendepunkten. Beim Versenden einer Nachricht eines Prozesses werden die aktuellen Wahrheitswerte von den ISMs A'^{φ} und $S'^{\varphi}{}_i^j$ an die Nachricht angehängt. Bei Ankunft einer Nachricht auf einem Prozess werden die Wahrheitswerte in der Nachricht genommen und in dem Speicher des Prozesses abgespeichert. Alte Wahrheitswerte werden dabei überschrieben.

Die in den letzten Abschnitten beschriebene Umwandlung von einer fSDTL-Formel in ein ∞-Tupel aus ISMs wird im weiteren Verlauf der Arbeit als eine Funktion $U : \text{fSDTL} \to \text{ISM}^{\infty}$ verwendet.

4.3.8 ISMs zu DISM

Dieses eine Tupel $(A'^{\varphi}, S'^{\varphi})$ aus ISMs genügt, wie im zweiten Abschnitt des Kapitels beschrieben, im Allgemeinen aber nicht als Automatenmodell für eine fSDTL-Formel. Die Auswertungen der Teilformeln der Form $@_q\psi$ müssen auch als Automaten dargestellt werden, damit auch die Entstehung der Werte in den Nachrichten, welche die einzelnen ISMs benötigen, durch Automaten simuliert werden.

Seien $\psi_1, \psi_2 \in \{\psi' \mid @_r\psi' \in \text{fSDTL}\}$, a eine atomare Proposition und $q \in P$ ein Prozess. Die Menge aller Teilformeln der Form $@_q\psi$ einer fSDTL-Formel ist durch die Funktion $sub_@ : \text{fSDTL} \to 2^{\text{fSDTL}}$ wie folgt definiert:

$$
\begin{aligned}
sub_@(\text{true}) &= \emptyset \\
sub_@(a) &= \emptyset \\
sub_@(\neg\psi_1) &= sub_@(\psi_1) \\
sub_@(\psi_1 \vee \psi_2) &= sub_@(\psi_1) \cup sub_@(\psi_2) \\
sub_@(\bigcirc\psi_1) &= sub_@(\psi_1) \\
sub_@(\psi_1 \,\mathcal{U}\, \psi_2) &= sub_@(\psi_1) \cup sub_@(\psi_2) \\
sub_@(@_q\psi_1) &= \{@_q\psi_1\} \cup sub_@(\psi_1)
\end{aligned}
$$

Sei $@_p\varphi$ wieder die gesamte fSDTL-Formel, dessen Auswertung durch Automaten dargestellt werden soll. Weiter sei $U(@_p\varphi) = (\overline{A}, \overline{S})$ mit

$$
\overline{S} = \overline{S}_1^1, \overline{S}_1^2, \ldots, \overline{S}_1^{|P|-1}, \overline{S}_2^1, \overline{S}_2^2, \ldots, \overline{S}_2^{|P|-1}, \overline{S}_3^1, \overline{S}_3^2, \ldots, \overline{S}_3^{|P|-1}, \ldots
$$

und $n = |sub_@(@_p\varphi) \setminus @_p\varphi|$. Das Tupel der Tupel aus ISMs, I_D, ergibt sich dann durch

$$
I_D = ((\overline{A}, \overline{S}), U_1, U_2, \ldots, U_n)
$$

mit $\forall i \neq j : U_i \neq U_j$ und $\forall 1 \leq i \leq n : \exists x \in sub_{@}(@_p\varphi) \setminus @_p\varphi : U_i = U(x)$. Das heißt, I_D enthält einmal das Tupel aus ISMs, das aus der Gesamtformel entsteht und zum Anderen die Tupel aus ISMs für alle Formeln der Form $@_q\psi$, die in der Gesamtformel vorkommen.

I_D enthält daher alle Tupel aus Automaten, die benötigt werden, um alle entfernten Formeln ab allen möglichen Sendepunkten auszuwerten sowie das Tupel aus Automaten für die gesamte Formel $@_p\varphi$, $(\overline{A}, \overline{S})$. $\overline{A}$ ist der Automat, welcher die Ausgabe der gesamten Formel stellt. Aus diesem Grund steht diese ISM auch an der ersten Stelle des ersten Tupels in I_D, denn dadurch ist $\overline{A}$ auch der Automat, der die Ausgabe des gesamten Automatenmodells erzeugt.

Die DISM, welche das Automatenmodell für die Formel $@_p\varphi$ darstellt, ist dann I_D.

In dem folgenden Theorem wird die Korrektheit der, für eine fSDTL-Formel entstehende, DISM gezeigt. Das wichtige dabei ist, dass, im Gegensatz zu den vorherigen Lemmata und Theoremen, die Korrektheit der Werte in f nicht mehr angenommen werden muss. Diese Annahme fällt weg, weil die DISM auch den Versand und Empfang von Nachrichten simuliert.

Theorem 4.21 (Korrektheit der DISM I_D)**.** *Sei $\overline{A} = (\Sigma, Q_{\overline{A}}, q_{0\overline{A}}, \delta_{\overline{A}}, \mathbb{B}_3, \lambda_{\overline{A}})$, $w \in \Sigma^n$ und $k_0, k_1, \ldots, k_n$ mit $k_0 = q_{0\overline{A}}$ und $k_l = \delta_{\overline{A}}(k_{l-1}, w_{l-1}), 1 \leq l \leq n$ der Lauf von $\overline{A}$ in I_D bei Eingabe von w. Es gilt $[\![w \models @_p\varphi]\!]_{fSDTL} = \lambda_{\overline{A}}(k_n)$.*

Beweis. Dass alle ISMs zu jedem Zeitpunkt die korrekten Wahrheitswerte ausgeben, wenn die Werte, die f zurückgibt, korrekt sind, wurde in Theorem 4.20 auf Seite 90 gezeigt. Zu zeigen bleibt, dass in dem Speicher der einzelnen Prozesse immer die korrekten Wahrheitswerte stehen, also der Nachrichtenversand funktioniert und damit die Werte in f korrekt sind.

Wie in der Definition der DISM angegeben, werden die aktuellen Ausgaben aller ISMs, die zu einem Prozess gehören, an eine Nachricht angehängt, wenn diese versandt werden soll. Damit enthält die Nachricht nach Theorem 4.20 auf Seite 90 dieselben Werte, wie es durch die Definition der Nachrichten für fSDTL der Fall ist, bis auf eine Ausnahme. In fSDTL gibt es nach der Definition keine Werte für die Auswertung von Formeln ab bestimmten Sendepunkten, wenn diese Sendepunkte noch nicht aufgetreten sind. In dem Automatenmodell ist das anders, denn es werden die Werte aller ISMs eines Prozesses in die Nachrichten geschrieben, also auch der Sendepunkt-ISMs, deren entsprechender Sendepunkt noch nicht aufgetreten ist. Nach Theorem 4.20 auf Seite 90 2b. geben diese Automaten dann ? aus. Dies hat aber keine Auswirkungen auf den sich ergebenen Wahrheitswert, weil sich in der fSDTL-Semantik für eine entfernte Formel ? sowohl ergibt, wenn es keinen Wahrheitswert für die Formel ab einem Sendepunkt gibt, als auch, wenn sich für die Formel ab diesem Sendepunkt ? ergibt.

Bei Ankunft einer Nachricht werden alle enthaltenen Wahrheitswerte in den Speicher des Prozesses geschrieben, wodurch dort die korrekten und aktuellen Wahrheitswerte stehen. Weil die Funktion f ihre Werte aus dem Speicher holt, gibt damit auch f immer die korrekten Werte zurück.

Danach folgt nun, dass alle ISMs zu jedem Zeitpunkt dieselben Wahrheitswerte wie die fSDTL-Semantik ausgeben, insbesondere auch $\overline{A}$ und damit folgt $[\![w \models @_p\varphi]\!]_{\text{fSDTL}} = \lambda_{\overline{A}}(k_n)$. □

Damit wurde eine Umwandlung von einer fSDTL-Formel in das Automatenmodell gefunden. Die entstehende DISM simuliert alle nötigen Monitore auf den verschiedenen Prozessen, den Inhalt sowie den Empfang der Nachrichten und die Ausgabe von I_D ist die Ausgabe des Automaten, der die Auswertung der gesamten Formel repräsentiert.

Ein Problem kann sich ergeben, weil ein DIA durch zwei Potenzmengenkonstruktionen entsteht, die auf einem IABA durchgeführt werden. Da der IABA im Allgemeinen unendlich viele Zustände und Transitionen besitzt, könnte der entstehende DIA und damit auch die ISMs und die DISM überabzählbar unendlich sein. Dass dies aufgrund der speziellen Struktur aus Schichten in den IABAs nicht der Fall ist, wird in dem folgenden Theorem gezeigt.

Theorem 4.22 (Abzählbare Unendlichkeit der DISM I_D)**.** *Die für $@_p\varphi$ durch die Umwandlung entstehende DISM I_D ist abzählbar unendlich.*

Beweis. In Lemma 4.17 auf Seite 85 wurde durch 1. und 2. gezeigt, dass jeder Formel-DIA aus unendlich vielen, isomorphen Schichten mit einer festen, endlichen Anzahl von Zuständen besteht. Dadurch folgt, dass die Formel-DIAs abzählbar unendlich sind. Mit 3. desselben Lemmas folgt das Selbe auch für die Sendepunkt-DIAs. Damit sind auch alle ISMs abzählbar unendlich, weil diese nur jeweils zwei DIAs simulieren. Da eine DISM nur aus unendlich vielen ISMs besteht, ist auch die DISM abzählbar unendlich. □

4.4 Beschränkung auf endliche Größen

Wie bereits während der Umwandlung in den vorherigen Abschnitten beschrieben wurde, sind einige der genutzten Algorithmen in der Praxis nicht anwendbar. Dazu gehören die beiden Potenzmengenkonstruktionen und der Leerheitstest, die jeweils unendlich viel Zeit benötigen. Des Weiteren sind die am Ende entstehenden ISMs unendlich groß und es entstehen unendlich viele ISMs. In dem Lauf eines Systems gibt es aber nie einen Zeitpunkt, bis zu dem unendlich viele Warte- oder Sendepunkte aufgetreten sind. Aus diesem Grund werden weder die gesamten unendlich großen ISMs noch unendlich viele dieser Automaten zu einem Zeitpunkt benötigt. Stattdessen können die Automaten und die Anzahl der Automaten über die Zeit wachsen. Dies wird durch die Schicht-Struktur der Automaten ermöglicht, was im weiteren Verlauf dieses Abschnittes genauer betrachtet wird.

Anstatt die Automaten erst komplett aufzubauen und dann im Nachhinein mittels Schichten zu betrachten können aufgrund der Eigenschaften der Schichten auch die Automaten durch Schichten aufgebaut werden. Dafür wird aus einer fSDTL-Formel nicht ein ∞-Tupel aus IABAs sondern nur ein einziger ABA erstellt, welcher der Schicht des Formel-IABAs mit $m = \vec{0}$ entspricht. Dieser ABA wird dann genau wie der ABA, der für eine LTL-Formel entsteht, in einen DFA umgewandelt. Das Selbe wird mit der negierten Formel getan und am Ende ergibt sich eine FSM wie es auch in [BLS11] der Fall ist, im Folgenden Schicht-FSM genannt. Diese Schicht-FSM entspricht nun einer Schicht der sonst entstehenden Formel-ISM.

Mittels dieser Schicht-FSM können nun die Formel- und Sendepunkt-FSMs aufgebaut werden. Dafür wird zunächst folgendes Theorem betrachtet.

Theorem 4.23 (Endliche Anzahl an Wartepunkten)**.** *Wenn nur endlich viele Wartepunkte existieren, so entsteht für eine fSDTL-Formel $@_p\varphi$ eine Formel-FSM anstelle einer Formel-ISM.*

Beweis. Existieren nur endlich viele Wartepunkte, so gibt es nur endlich viele verschiedene Wartepunkttupel. Dadurch folgt, dass für φ und $\neg\varphi$ nur Formel-ABAs aus den Schichten für die möglichen Wartepunkttupel der ansonsten entstehenden Formel-IABAs entstehen, weil nach Lemma 4.17 auf Seite 85 alle Schichten endlich groß sind. Aus den Formel-ABAs entstehen durch die Umwandlung DFAs und aus diesen eine FSM, weil dafür dieselbe Transformation wie in [BLS11] benutzt wird. □

Dieses Theorem sagt aus, dass sich für eine endliche Anzahl an Wartepunkten nur eine endlich große Moore-Maschine ergibt. Die entstehende FSM besteht außerdem nach Lemma 4.17 auf Seite 85 aus Schichten, die der Schicht-FSM gleichen. Das heißt, es kann am Beginn des Laufes des Systems mit der Schicht-FSM als Automat begonnen werden. Tritt nun in einem Schritt des Systems ein Wartepunkt auf, so können zu der Formel-FSM nach Theorem 4.23 und Lemma 4.17 auf Seite 85 endlich viele Schichten, die isomorph zur Schicht-FSM sind, hinzugefügt werden, um sich diesen Wartepunkt korrekt zu merken. Dies kann bei jedem weiteren Wartepunkt so fortgesetzt werden, wodurch die FSM nach endlicher Laufzeit des Systems auch nur endlich groß wird.

Die Sendepunkt-FSMs können auf ähnliche Weise entstehen. Tritt ein Sendepunkt in einem Schritt auf, so wird eine neue Sendepunkt-FSM hinzugefügt, die aus einem Wartezustand sowie der Schicht-FSM besteht. Der Sendepunkt, auf den die Sendepunkt-FSM wartet, ist der Sendepunkt, bei dessen Auftreten sie erstellt wurde. Für alle vorher gesehenen Wartepunkte werden so viele Wartezustände hinzugefügt, wie es durch die aufgetretenen Wartepunkte verschiedene Wartepunkttupel gibt, weil in Lemma 4.17 auf Seite 85 gezeigt wurde, dass die Wartezustände sowie die Transitionen zwischen diesen in den Sendepunkt-DIAs und in den Sendepunkt-IABAs identisch sind. Zu der Schicht-FSM werden, wie im vorherigen Absatz beschrieben, Schichten für die bereits aufgetretenen Wartepunkte hinzugefügt. Der Startzustand der Sendepunkt-FSM ist der Wartezustand mit dem Wartepunkttupel, das durch die vorher aufgetretenen Wartepunkte aktuell ist, da sonst alle vorher aufgetretenen Wartepunkte vergessen werden. Außerdem wird noch ein Schritt mit dem Zeichen durchgeführt, in dem der Sendepunkt, für den die Sendepunkt-FSM erstellt wurde, vorkommt. Für jeden später auftretenden Wartepunkt werden dem, der Formel-FSM entsprechenden, Teilautomaten der Sendepunkt-FSM Schichten, genau wie der Formel-FSM auch, hinzugefügt. Da nach endlicher Laufzeit des Systems nur endlich viele Warte- und Sendepunkte aufgetreten sein können, sind die Sendepunkt-FSMs immer endlich groß und es existieren auch nur endlich viele zu einem Zeitpunkt.

Auf diese Weise können alle entfernten Teilformeln der Gesamtformel gehandhabt werden, wodurch zu jedem Zeitpunkt nur endlich viele endlich große FSMs existieren, die trotzdem die Formel korrekt auswerten. Außerdem wird für die Potenzmengenkonstruktionen und den Leerheitstest zur Erstellung der Schicht-FSM nur endlich viel Zeit benötigt, weil eine Schicht nur endlich groß ist.

Im nächsten Abschnitt soll nun die Anzahl der Zustände dieser FSMs im Vergleich zur Anzahl der Teilformeln der Formel, aus der sie entstanden sind, für eine feste Anzahl an aufgetretenen Warte- und Sendepunkten betrachtet werden.

4.5 Komplexitätsanalyse

In diesem Abschnitt soll die Komplexität der endlich großen Automaten aus dem vorherigen Abschnitt betrachtet werden. Interessant dabei ist die Betrachtung der Umwandlung von einer fSDTL-Formel in Tupel aus FSMs. Dafür muss aber zuerst einmal die Komplexität der Schicht-FSM, die Komplexität einer FSM für eine feste Anzahl an Wartepunkten und die Komplexität eines Tupels aus FSMs für eine feste Anzahl an Warte- und Sendepunkten betrachtet werden. Als Maß für die Komplexität wird die Erhöhung der Zustände genommen.

Zuerst wird die Schicht-FSM betrachtet, wie im vorherigen Abschnitt beschrieben, und deren Größe im Verhältnis zur Anzahl der Teilformeln der Formel, aus der sie entstanden ist, betrachtet. Da die FSMs mit Hilfe dieser Schicht-FSM aufgebaut werden, lässt sich aus der Größe der Schicht-FSM auch auf die Komplexität der FSMs für eine feste Anzahl an Wartepunkten schließen.

Es soll nun die Komplexität der Schicht-FSM für eine fSDTL-Formel $@_p\varphi$ mit $|\varphi|$ Teilformeln betrachtet werden. In [Var95] wurde gezeigt, dass nach der Umwandlung von einer LTL-Formel in einen ABA die Anzahl der Zustände in $O(|\varphi|)$ liegt, also die Anzahl der Zustände der Anzahl der Teilformeln der LTL-Formel entspricht. Dies gilt auch für eine Schicht der IABAs für fSDTL. Da es dann bei dem Next-Operator keine Veränderung des Wartepunkttupels gibt, entspricht eine Schicht des IABAs dem ABA, der sich für LTL ergibt mit zusätzlichem @-Operator. Jede entfernte Teilformel erzeugt nur genau einen Zustand. Damit ergibt sich auch eine konstante Erhöhung für die Anzahl der Zustände im Vergleich zu der Umwandlung von einer LTL-Formel in einen ABA aus [Var95]. Das heißt, die Anzahl der Zustände einer Schicht eines IABAs liegt in $O(|\varphi|)$.

Da eine Schicht eines IABAs ein ABA ist, kann dieselbe Umwandlung wie in [BLS11] verwendet werden. Zuerst ergibt sich für den entstehenden NBA eine exponentielle Erhöhung der Zustände. Die Anzahl der Zustände einer nicht-deterministischen Schicht ist folglich in $O(2^{|\varphi|})$. Der Leerheitstest auf der Schicht erzeugt keine Erhöhung, da dieser nur mehr Zustände akzeptierend macht und keine neuen erzeugt.

Für die Umwandlung der, aus dem Leerheitstest entstehenden, Schicht eines NIAs in eine deterministische Schicht entsteht wieder eine exponentielle Erhöhung der Anzahl der Zustände. Dies folgt auch aus [BLS11], da wieder dieselbe Umwandlung genutzt wird. Die Anzahl der Zustände einer Schicht eines DIAs liegt also in $O(2^{2^{|\varphi|}})$.

Damit folgt, dass die Anzahl der Zustände der Schicht-FSM in $O(2^{2^{|\varphi|}})$ liegt.

In Abschnitt 4.4 auf Seite 93 wurden die FSMs dann für jeden auftretenden Wartepunkt erweitert. Im Folgenden soll nun betrachtet werden, wie die Komplexität einer FSM zu einem Zeitpunkt ist, bis zu dem eine bestimmte Anzahl an Wartepunkten aufgetreten ist. Sei $|P| = \{p, p_1, p_2, \ldots, p_{|P|-1}\}$

die Menge der Prozesse und x_i mit $i \in \{1,2,\ldots,|P|-1\}$ die Anzahl der bei Prozess p aufgetretenen Wartepunkte, an denen Prozess p auf Prozess p_i warten musste. Das heißt, dass es für die i-te Stelle des Wartepunkttupels $x_i + 1$ Möglichkeiten gibt, nämlich die Werte 0 bis x_i. Daraus folgen insgesamt $\Pi_{i\in\{1,2,\ldots,|P|-1\}} 1 + x_i$ verschiedene Belegungen für das Wartepunkttupel m. Für $\Pi_{i\in\{1,2,\ldots,|P|-1\}} 1 + x_i$ verschiedene Möglichkeiten für das Wartepunkttupel entstehen statt nur einer Schicht, wie bei der Umwandlung für die Schicht-FSM, ebenso viele Schichten in dem ABA. Die Anzahl der Zustände für diesen ABA liegt also in $O(|\varphi| \cdot \Pi_{i\in\{1,2,\ldots,|P|-1\}} 1 + x_i)$. Die Anzahl der Zustände der FSM ist dann in

$$O\left(2^{2^{|\varphi|\cdot\Pi_{i\in\{1,2,\ldots,|P|-1\}} 1+x_i}}\right).$$

Dies gilt auch für die Sendepunkt-FSMs, denn diese enthalten nach Lemma 4.17 auf Seite 85 nur für jedes mögliche Wartepunkttupel einen Wartezustand zusätzlich, wodurch die Anzahl ihrer Zustände in dieselbe O-Klasse fällt.

Damit nicht unendlich viele Automaten entstehen wurden in Abschnitt 4.4 auf Seite 93 nur weitere Sendepunkt-FSMs hinzugefügt, wenn ein Sendepunkt aufgetreten ist. Wie im vorherigen Abschnitt kann daher auch die Anzahl der entstehenden Automaten in Abhängigkeit von den, bis zu einem Zeitpunkt aufgetretenen, Sendepunkten bestimmt werden. Sei n die Anzahl der Sendepunkte, die bis zu einem Zeitpunkt auf dem Prozess p vorgekommen sind. Das heißt, es existieren bis zu diesem Zeitpunkt n Sendepunkt-FSMs. Zusammen mit der Betrachtung der Wartepunkte ergibt sich zu diesem Zeitpunkt eine gesamte Anzahl an Zuständen des Tupels aus FSMs, die in

$$O\left(n \cdot 2^{2^{|\varphi|\cdot\Pi_{i\in\{1,2,\ldots,|P|-1\}} 1+x_i}}\right)$$

liegt.

In der Formel $@_p\varphi$ kann es aber noch entfernte Teilformeln geben. Für jede dieser Teilformeln entsteht ein Tupel aus FSMs und auf jedem Prozess kann es unterschiedlich viele Warte- und Sendepunkte bis zu diesem Zeitpunkt gegeben haben. Insgesamt ergeben sich $|sub_@(@_p\varphi)|$ Tupel. Ohne Beeinträchtigung der Allgemeinheit wird angenommen, dass es kein Tupel gibt, das für ein $@_q\psi \in sub_@(@_p\varphi) \setminus @_p\varphi$ entsteht, dessen Anzahl an Zuständen in einer größeren O-Klasse liegt als die Anzahl der Zustände des Tupels, das für $@_p\varphi$ entsteht. Damit kann dann eine obere Schranke für die entstehende Anzahl der Zustände aller Tupel durch

$$O\left(|sub_@(@_p\varphi)| \cdot n \cdot 2^{2^{|\varphi|\cdot\Pi_{i\in\{1,2,\ldots,|P|-1\}} 1+x_i}}\right)$$

angegeben werden. Dies ergibt sich, weil für jedes Element aus $|sub_@(@_p\varphi)|$ ein Tupel entsteht und $O\left(n \cdot 2^{2^{|\varphi|\cdot\Pi_{i\in\{1,2,\ldots,|P|-1\}} 1+x_i}}\right)$ die obere Schranke für das Tupel mit den meisten Zuständen ist.

5 Zusammenfassung und Ausblick

Das Ziel dieser Arbeit war es, eine Logik für verteilte, asynchrone Systeme basierend auf ptDTL aus [SVAR04] zu entwickeln. Dieser neuen Logik sollte ein eindeutiges Modell zugrunde liegen und sie sollte mächtiger sein und mehr Eigenschaften monitoren können als ptDTL sowie Impartialiy und Anticipation besitzen. Außerdem wurde ein Automatenmodell und eine Umwandlung in dieses für Formeln der neuen Logik gefordert.

Dafür wurde zuerst ptDTL um Nachrichten in den Läufen der Prozesse zu ptDTL$_N$ erweitert, wodurch eine eindeutige Semantik für den @-Operator angegeben werden konnte. Basierend auf dieser Erweiterung von ptDTL wurde dann fDTL definiert, eine Logik, die derselben Idee wie ptDTL folgt, der aber LTL$_3$ anstelle von ptLTL zugrunde liegt. Dadurch besitzt fDTL eine dreiwertige Semantik mit Impartiality und Anticipation. Im Gegensatz zu ptDTL, mit der nur maximal Safety- oder Guarantee-Eigenschaften monitorbar sind, sind durch die dreiwertige Semantik und die Impartiality mit fDTL Safety- und Guarantee-Eigenschaften sowie die gesamte Obligation-Klasse und noch einige Eigenschaften aus den Klassen Response und Persistence monitorbar. Außerdem werden durch die Anticipation mit fDTL endgültige Wahrheitswerte früher erkannt als bei ptDTL.

Als Nächstes wurde ein eindeutiges Modell des Systems entwickelt. In diesem wurden zuerst Prozesse als Transitionssysteme definiert sowie die Verteilung der Prozesse und Monitore und deren Kommunikation beschrieben. Danach wurden Scheduler definiert, welche eine Ausführungsreihenfolge der Prozesse in einem Lauf des Systems angeben. Durch das Ausführen der Transitionssysteme der Prozesse in dieser Reihenfolge wird dann eine Ausführung generiert. Des Weiteren wurden Synchronisationsaktionen entwickelt, mit denen die Synchronisationspunkte der Prozesse in ihren Läufen angezeigt werden können. Dadurch ist den Monitoren eines Prozesses bekannt, dass, wenn ihr Prozess einen Wartepunkt überschreitet, der Prozess, auf den gewartet wurde, den entsprechenden Sendepunkt mindestens erreicht haben muss. Danach wurde fSDTL als eine Erweiterung von fDTL um Synchronisationsaktionen mit dem vorher beschriebenen Modell des Systems als Grundlage definiert. Wie fDTL besitzt auch fSDTL eine dreiwertige Semantik, Impartiality und Anticipation sowie dieselben monitorbaren Eigenschaften wie fDTL. Durch die Synchronisationsaktionen wurde allerdings der große Nachteil von fDTL behoben, dass alle entfernten Teilformeln vom Beginn des Laufes des entfernten Prozesses an ausgewertet werden. Es wurde gezeigt, dass fSDTL dadurch eine größere Mächtigkeit besitzt als fDTL.

Zu untersuchen bleiben die Mächtigkeit und die Eigenschaften von fSDTL für verschiedenen Arten von Schedulern. Je nach Art des genutzten Schedulers verändert sich die Menge der möglichen Ausführungen des Systems. Außerdem sagt die Betrachtung nur eines Schedulers nichts über das Verhalten eines Systems aus, denn er beschreibt nur eine, durch Asynchronität zustande kommende, Ausführungsreihenfolge der Prozesse. Diese kann sich beim nächsten Ausführen ändern und

dann kann die Formel zu einem anderen Wahrheitswert ausgewertet werden. Es müssten daher anstelle von einer Ausführung die Ausführungen aller Scheduler betrachtet werden, damit man sagen kann, dass die Formel unabhängig von dem asynchronen Verhalten erfüllt wird.

Für die Logik fSDTL wurde dann ein Automatenmodell entwickelt. Für das Modell wurden zunächst unendlich große Automaten der Automatentypen definiert, die auch in [BLS11] für die Monitorgenerierung für LTL_3 genutzt wurden. Danach wurde eine Umwandlung einer fSDTL-Formel in ein Automatenmodell angegeben, die auf der aus [BLS11] für LTL_3 basiert, allerdings in jedem Schritt statt eines endlich großen Automaten aufgrund der Warte- und Sendepunkte für den @-Operator ein ∞-Tupel aus unendlich großen Automaten des entsprechenden Typs erzeugt. Des Weiteren ergab sich für jede entfernte Teilformel ein weiteres ∞-Tupel, wodurch sich am Ende als Automatenmodell eine DISM aus endlich vielen Tupeln mit jeweils unendlich vielen ISMs ergibt. Für die einzelnen ISMs wurde gezeigt, dass diese allerdings nur aus unendlich vielen, endlich großen, isomorphen Schichten bestehen und deswegen konnte bewiesen werden, dass die DISM trotz der beiden Potenzmengenkonstruktionen auf unendlich großen Automaten abzählbar unendlich bleibt. Als nächstes wurde gezeigt, dass die entstehenden Automaten nur endlich groß sind, wenn nur eine endliche Anzahl an Wartepunkten existiert. Dadurch konnten anstelle der ISMs FSMs betrachtet werden, die zuerst nur aus einer Schicht bestehen und pro auftretenden Wartepunkt erweitert werden. Auf dieselbe Art wurden die Sendepunkte betrachtet und bei Auftreten eines Sendepunktes ein weiterer Automat hinzugefügt, weshalb weder die unendlich großen noch unendlich viele Automaten benötigt werden. Als Letztes wurde noch die Komplexität der endlich großen Automaten für eine feste Anzahl an Warte- und Sendepunkten analysiert. Dabei ergab sich, dass jeder Automat doppelt exponentiell in dem Produkt aus der Anzahl der Teilformeln der Formel und der, sich durch die bisher aufgetretenen Wartepunkte ergebenen, Anzahl an Möglichkeiten für die Wartepunkttupel wächst. Die Anzahl der Automaten wächst linear in der Anzahl der Sendepunkte.

Es bleibt eine konstruktive Methode zur Erweiterung der wachsenden FSMs bei Auftreten eines Wartepunktes anzugeben. Es wurde nur gezeigt, dass die FSMs für eine endliche Anzahl an Wartepunkten immer endlich groß sind und weil sie aus isomorphen Schichten bestehen muss es möglich sein, diese pro Wartepunkt weiter auszubauen. Weiter bleibt zu untersuchen, wie es möglich ist, die FSMs für den praktischen Einsatz auf eine feste Größe zu begrenzen. Dafür könnte sich, zum Beispiel, immer nur eine feste Anzahl an Wartepunkten zu jedem Zeitpunkt gemerkt werden. Selbiges gilt auch für eine feste Anzahl an FSMs, das heißt, dass sich nur eine feste Anzahl an Sendepunkten gemerkt werden. Außerdem muss analysiert werden, welche Nachteile sich durch eine feste Größe und Anzahl an FSMs ergeben und wie diese Nachteile minimiert werden können. Es bleibt außerdem zu untersuchen, ob ein Automatenmodell für fSDTL-Formeln konstruiert werden kann, das auf weniger mächtigen Automaten als den hier benutzten, unendlich großen, Automaten beruht und dessen Anzahl der Zustände nicht über die Zeit wächst.

Abkürzungsverzeichnis

LTL Linear Temporal Logic

ptLTL Past-Time Linear Temporal Logic

ptDTL Past-Time Distributed Temporal Logic

fDTL Future Distributed Temporal Logic

fSDTL Future Synchronized Distributed Temporal Logic

ABA Alternierender Büchi-Automat

NBA Nicht-deterministischer Büchi-Automat

NFA Nicht-deterministischer endlicher Automat

NIA Nicht-deterministischer unendlicher Automat

DFA Deterministischer endlicher Automat

DIA Deterministischer unendlicher Automat

FSM Endliche deterministische Moore-Maschine

ISM Unendliche deterministische Moore-Maschine

DISM Verteilte unendliche deterministische Moore-Maschine

NNF Negationsnormalform

DNF Disjunktive Normalform

KNF Konjunktive Normalform

Abbildungsverzeichnis

Literaturverzeichnis

[BF12] BAUER, Andreas ; FALCONE, Yliès: Decentralised LTL Monitoring. In: *FM* Bd. 7436, Springer, 2012 (Lecture Notes in Computer Science), S. 85–100

[BLS11] BAUER, Andreas ; LEUCKER, Martin ; SCHALLHART, Christian: Runtime Verification for LTL and TLTL. In: *ACM Trans. Softw. Eng. Methodol.* 20 (2011), Nr. 4, S. 14

[FFM12] FALCONE, Yliès ; FERNANDEZ, Jean-Claude ; MOUNIER, Laurent: What can you verify and enforce at runtime? In: *STTT* 14 (2012), Nr. 3, S. 349–382

[GP10] GOODLOE, Alwyn ; PIKE, Lee: Monitoring Distributed Real-Time Systems: A Survey and Future Directions. (2010)

[Leu11] LEUCKER, Martin: Teaching Runtime Verification. In: *RV* Bd. 7186, Springer, 2011 (Lecture Notes in Computer Science), S. 34–48

[MP90] MANNA, Zohar ; PNUELI, Amir: A Hierarchy of Temporal Properties. In: *PODC*, ACM, 1990, S. 377–410

[Sch14] SCHMITZ, Malte: *Verteilte Laufzeitverifikation auf eingebetteten Systemen*, Universität zu Lübeck, Masterarbeit, 2014

[SVAR04] SEN, Koushik ; VARDHAN, Abhay ; AGHA, Gul ; ROŞU, Grigore: Efficient Decentralized Monitoring of Safety in Distributed Systems. In: *ICSE*, IEEE Computer Society, 2004, S. 418–427

[SVAR06] SEN, Koushik ; VARDHAN, Abhay ; AGHA, Gul ; ROŞU, Grigore: Decentralized runtime analysis of multithreaded applications. In: *IPDPS*, IEEE, 2006

[Var95] VARDI, Moshe Y.: An Automata-Theoretic Approach to Linear Temporal Logic. In: *Banff Higher Order Workshop* Bd. 1043, Springer, 1995 (Lecture Notes in Computer Science), S. 238–266

[Zuc86] ZUCK, Leonore: *Past Temporal Logic*, Weizmann Institute of Science, Diss., 1986