Studienarbeit

# Implementierung einer schrittbasierten Interleavingsemantik für die Temporal Logic of Distributed Actions (TLDA)

Niels Lohmann

20. Juni 2005


Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät II
Insititut für Informatik

Betreuer: Peter Massuthe

**Zusammenfassung**

Die *Temporal Logic of Distributed Actions* (TLDA) ist eine temporale Logik mit einer Halbordnungssemantik und eignet sich somit zur Spezifikation verteilter Systeme. Allerdings gibt es noch wenig Erfahrung bei der computergestützten Verifikation halbordnungsbasierter Formalismen. Viele bekannte effiziente Algorithmen des expliziten Modelchecking setzen ein Transitionssystem und somit eine Interleavingsemantik voraus.

In dieser Arbeit wird eine Interleavingsemantik für TLDA vorgeschlagen und ein Prototyp entwickelt, der für eine bestimmte Klasse von TLDA-Spezifikationen das Transitionssystem aufbaut. Die entwickelte Semantik soll zusammen mit dem Prototypen die Grundlage für die weitere Arbeit an einem Modelchecker für TLDA sein.

# Inhaltsverzeichnis

**1 Einleitung** **5**

**2 Die Logik TLDA** **7**
2.1 Semantisches Modell . . . . . 7
2.2 Grafische Darstellung . . . . . 9
2.3 Syntax . . . . . 10
2.4 Semantik . . . . . 13
2.5 Weitere Definitionen . . . . . 16

**3 Interleavingsemantik** **19**
3.1 Interleavingsemantiken und Modelchecking . . . . . 19
3.2 Transitionsbasierte TLDA-Interleavingsemantik . . . . . 20
3.3 Schrittbasierte TLDA-Interleavingsemantik . . . . . 22
3.4 Transitionssysteme für Spezifikationen . . . . . 24

**4 Implementierung der Semantik** **27**
4.1 Spezifikationssprache TLDA$^+$ . . . . . 27
4.2 Schematischer Aufbau von TLDC . . . . . 29
4.3 Analyse der Eingabespezifikation . . . . . 30
4.4 Berechnung des reduzierten Transitionssystemes . . . . . 32
4.5 Einschränkungen von TLDC . . . . . 34
4.6 Probleme bei der Konstruktion . . . . . 34

**5 Beispiele** **36**
5.1 Einleitendes Beispiel . . . . . 36
5.2 Wechselseitiger Ausschluss . . . . . 40

**6 Zusammenfassung und Ausblick** **44**
6.1 Zusammenfassung . . . . . 44
6.2 Ausblick . . . . . 44

**Abbildungsverzeichnis** **46**

**Literaturverzeichnis** **47**

# 1 Einleitung

Systeme werden zunehmend komplexer, bestehen aus mehreren nebenläufigen Komponenten und interagieren als reaktive Systeme in hohem Maße mit ihrer Umgebung. Durch diese zunehmende Komplexität wirken sich Fehler meist auf große Teile des Systemes aus, sind gleichzeitig jedoch immer schwerer lokalisierbar. Allerdings können Testläufe nur die An-, nicht aber die Abwesenheit von Fehlern aufzeigen, sodass Fehlerfreiheit letztlich nur durch eine systematisches Durchmustern aller möglichen Systemzustände sichergestellt werden kann.

Dieser Ansatz der Verifikation wird Modelchecking genannt. Ein System muss dazu modelliert werden und die zu überprüfenden Eigenschaften in einer temporalen Logik formuliert werden [Pnu77]. Ein Werkzeug (der Modelchecker) überprüft den Zustandsraum des Systemes und verifiziert bzw. falsifiziert die beschriebenen Eigenschaften. Allerdings kann auch das zu überprüfende System in temporaler Logik modelliert (spezifiziert) werden [Pnu79], sodass temporale Logiken eine einheitliche Methode für die Spezifizierung und Verifizierung von Systemen darstellen.

Die Temporal Logic of Actions [Lam94], kurz TLA, ist eine temporale Logik mit einer Interleavingsemantik, die sich durch mehrere Fallstudien als Modellierungsmethode verteilter Algorithmen im Hard- und Softwarebereich bewährt hat und unter Systemmodellierern weit verbreitet ist. Die computergestützte Verifikation von TLA-Spezifikationen ermöglicht der Modelchecker TLC [YML99], mit dem bereits komplexe industriell entwickelte Hardwareprotokolle analysiert und korrigiert werden konnten [LMTY02].

Die Temporal Logic of Distributed Actions [Ale05], kurz TLDA, ist eine neue temporale Logik, die syntaktisch als eine Variante von TLA angesehen werden kann, jedoch auf halbgeordneten Abläufen basiert. Durch diese Halbordnungssemantik ist TLDA in gewisser Hinsicht ausdrucksstärker als TLA, da kausale Zusammenhänge und insbesondere Nebenläufigkeit aus den Abläufen abgelesen werden können. Zu dieser Logik existiert jedoch noch kein Modelchecker oder andere Werkzeuge, die die Spezifikation und Verifikation unterstützen.

Diese Arbeit soll die Grundlage für einen TLDA-Modelchecker sein, der für die Verbreitung der Logik hilfreich ist. Allerdings gibt es derzeit wenig Erfahrung bei der computergestützten Verifikation halbordnungsbasierter Formalismen, sodass eine Anpassung vorhandener Arbeit an TLDA nicht möglich ist. Die meisten expliziten Modelchecker setzen Transitionssysteme — Graphen, bestehend aus Zuständen und Aktionen, in denen das Verifikationsproblem durch Suche im Zustandsraum gelöst wird — und damit Interleavingsemantiken voraus.

Die Entwicklung einer Interleavingsemantik, also die Ausarbeitung eines Zustands- und eines Aktionsbegriffes ist der erste Teil dieser Arbeit. Der zweite Teil befasst sich mit der Implementierung dieser Semantik, d.h. eines Werkzeuges, das, gegeben eine TLDA-Spezifikation, ein Transitionssystem aufbaut, die Grundlage für späteres explizites Modelchecking ist.

Dazu gliedert sich die Arbeit wie folgt: In Kapitel 2 führen wir in die Logik TLDA ein. Wir beschreiben die Syntax und Semantik und legen die theoretischen Grundlagen für die weitere Arbeit.

In Kapitel 3 beschreiben wir die Eigenschaften von Interleavingsemantiken und untersuchen einen bestehenden Ansatz einer TLDA-Interleavingsemantik und zeigen, dass dieser Ansatz nicht für die computergestützte Verifikation anwendbar ist. Wir arbeiten eine neue Interleavingsemantik aus und beschreiben die Konstruktion von Transitionssystemen für spezifizierte Systeme.

In Kapitel 4 beschreiben wir, wie dieser Ansatz implementiert werden kann und diskutieren durch die Implementierung notwendige Einschränkungen, Anpassungen und entstehende Probleme der zuvor theoretisch vorgestellten Semantik.

Mit dem im Rahmen dieser Arbeit entstandenen Prototyp TLDC illustrieren wir in Kapitel 5 anhand zweier Beispiele die Semantik und die Konstruktion der Transitionssysteme.

Zuletzt fassen wir in Kapitel 6 die Resultate der Arbeit sowie die offen gebliebenen Probleme und interessante Fragestellungen zusammen.

# 2 Die Logik TLDA

In diesem Kapitel führen wir in die Temporal Logic of Distributed Actions (TLDA) ein. Dazu beschreiben wir zunächst das semantische Modell der Logik und alle dafür notwendigen Begriffe. Im nächsten Abschnitt geben wir eine grafische Darstellung an, mit deren Hilfe wir in den folgenden Kapiteln Beispiele illustrieren werden. Anschließend befassen wir uns mit dem syntaktischen Aufbau von TLDA-Formeln. In Abschnitt 2.4 gehen wir noch einmal — dann formal — auf die Semantik und die Auswertung von TLDA-Formeln ein. Zuletzt folgen weitere Definitionen, die wir für die folgenden Kapitel benötigen.

Die Definitionen in diesem Kapitel entstammen größtenteils [Ale05] und wurden teilweise für den Rahmen dieser Arbeit angepasst.

## 2.1 Semantisches Modell

Das semantische Modell von TLDA ist ein Ablauf, eine halbgeordnete Struktur, die an Kausalnetze von Petrinetzen [Rei86b] angelehnt ist. Bevor wir jedoch formal in die Semantik einführen können, müssen die Begriffe der Historie und der Transition definiert werden.

In der gesamten Arbeit sei $\mathcal{V}al$ die unendliche Menge aller Werte und $\mathcal{V}ar$ die unendliche Menge von Variablen. Eine Historie ordnet jeder Variablen aus $\mathcal{V}ar$ eine endliche oder unendliche Sequenz von Werten aus $\mathcal{V}al$ zu.

**Definition 1 (Historie)**
*Eine* Historie *ist eine Abbildung $H : \mathcal{V}ar \to \mathcal{V}al^\infty$, wobei $\mathcal{V}al^\infty$ die Menge aller endlichen und unendlichen Sequenzen über $\mathcal{V}al$ sei. Für eine Variable $x \in \mathcal{V}ar$ ist $H_x$ die Historie der Variablen $x$. Die Länge der Historie der Variablen $x$ bezeichnen wir mit $length(H_x)$.* ⌟

Eine Transition beschreibt die Aktualisierung einer einzelnen Variablen (d.h. einen Übergang eines Historienwertes vom Index $i$ zu $i+1$) oder eine synchronisierte Aktualisierung mehrerer Variablen:

**Definition 2 (Transition)**
*Sei $\emptyset \neq dom(t) \subseteq \mathcal{V}ar$ eine Menge von Variablen. Dann ist eine* Transition $t$ *definiert durch $t : dom(t) \to \mathbb{N}$. Die Variablen aus $dom(t)$ nennen wir* involvierte Variablen *der Transition $t$.* ⌟

Zwischen den Transitionen definieren wir eine Nachfolgerrelation:

**Definition 3 (Nachfolgerrelation)**
*Für zwei Transitionen $t$ und $u$ gilt $t \prec\cdot\, u$ ($u$ ist* direkter Nachfolger *von $t$) gdw. es eine Variable $x \in dom(t) \cap dom(u)$ gibt mit $t(x) = u(x) - 1$. Die transitive Hülle von $\prec\cdot$ bezeichnen wir mit $\prec$.* ⌟

Die bisher definierten Begriffe reichen aus, um den Begriff des Ablaufes formal zu definieren.

**Definition 4 (Ablauf)**
*Seien $H$ eine Historie und $T$ eine Menge von Transitionen. $\sigma = (H, T)$ ist ein* Ablauf *genau dann, wenn:*

1. *Für jede Variable $x \in \mathcal{V}ar$ gibt es genau eine Transition $t \in T$ mit $t(x) = i$ für $0 \leq i < length(H_x) - 1$.*
2. *Für alle Transitionen $t \in T$ und alle Variablen $x \in dom(t)$ gilt: $0 \leq t(x) < length(H_x) - 1$.*
3. *Die Relation $\prec$ ist irreflexiv.*

⌟

**Anmerkung:** Die Nachfolgerrelation $\prec$ zwischen Transitionen eines Ablaufes ist per Definition transitiv und nach Definition des Ablaufes irreflexiv: Es handelt sich um eine Halbordnung. Es kann also Transitionen $t$ und $u$ geben, für die weder $t \prec u$ noch $u \prec t$ gilt. Diese Transitionen sind dann *nebenläufig*.

Die folgenden Definitionen vervollständigen das semantische Modell:

**Definition 5 (Schnitt, Anfangsschnitt)**
*Sei $\sigma = (H, T)$ ein Ablauf. Eine Abbildung $C : \mathcal{V}ar \to \mathbb{N}$ ist genau dann ein* Schnitt in $\sigma$*, wenn für jede Transition $t \in T$ und alle Variablen $x, y \in dom(t)$ gilt: Wenn $t(x) < C(x)$, dann $t(y) < C(y)$. Den Schnitt $C_0$ mit $C_0(x) = 0$ für alle $x \in \mathcal{V}ar$ nennen wir* Anfangsschnitt von $\sigma$. ⌟

Ein Schnitt ordnet also jeder Variablen $x$ einen Index in $H_x$ zu und kann somit als lokaler Zustand verstanden werden, wie wir im folgenden Kapitel sehen werden.

**Notation:** Für zwei Schnitte $C_1, C_2$ schreiben wir $C_1 \leq C_2$ falls $C_1(x) \leq C_2(x)$ für alle $x \in \mathcal{V}ar$. Analog verwenden wir die Kurzschreibweisen $C_1 < C_2$, $C_1 \geq C_2$ etc.

**Definition 6 (Schalten einer Transition)**
*Seien $\sigma = (H, T)$ ein Ablauf, $C$ ein Schnitt in $\sigma$ und $t \in T$ eine Transition.* $t$ schaltet in $C$ *genau dann, wenn $C(x) = t(x)$ für alle $x \in dom(x)$.* ⌟

**Definition 7 (Nachfolgeschnitt, Schritt)**
*Seien $\sigma = (H, T)$ ein Ablauf, $C$ ein Schnitt in $\sigma$ und $T_C \subseteq T$ die Menge aller Transitionen $t$ mit $t(x) = C(x)$ für alle $x \in dom(t)$. Durch Schalten aller Transitionen in $T_C$ wird der* Nachfolgeschnitt *$C'$ erreicht. Der Schnitt $C$ und der Nachfolgeschnitt $C'$ bilden einen* Schritt *$S_C$ in $\sigma$. $C'$ ist für alle $x \in \mathcal{V}ar$ definiert durch:*

$$C'(x) \triangleq \begin{cases} C(x) + 1 & \text{wenn } x \in dom(t) \text{ für ein } t \in T_C, \\ C(x) & \text{sonst.} \end{cases}$$

*Den Schritt $S_C$ beschreiben wir mit dem Tripel $S_C = (C, C', T_C)$.* ⌟

**Definition 8 (Erreichbarkeit eines Schnittes)**
*Seien $\sigma$ ein Ablauf, $C_0$ der Anfangsschnitt von $\sigma$ und $C \geq C_0$ ein Schnitt in $\sigma$.* $C$ ist von $C_0$ durch Schritte erreichbar*, falls in $\sigma$ eine Folge von Schritten*

$$(C_0, C_1, T_{C_0})\ (C_1, C_2, T_{C_1})\ \ldots\ (C_{n-1}, C_n, T_{C_{n-1}})\ (C_n, C, T_{C_n})$$

*existiert mit $n \geq 0$.* ⌟

## 2.2 Grafische Darstellung

Um die bisher definierten Begriffe zu verdeutlichen und spätere Beispiele zu illustrieren, führen wir nun in die grafische Darstellung von Abläufen ein. Abbildung 2.1 zeigt den Ablauf $\sigma_1$. Auf der linken Seite stehen die Variablen, auf der rechten Seite ist die Historie zusammen mit den Transitionen abgebildet. Wir stellen stets nur einen Teil des Ablaufes dar, was durch drei Punkte angedeutet ist.

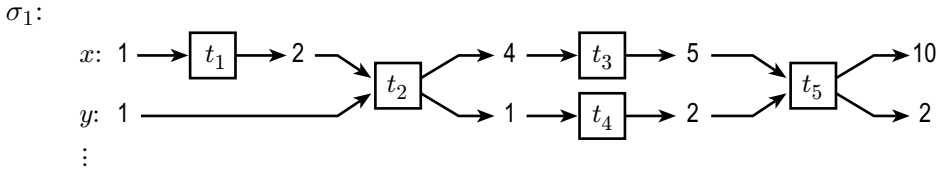


**Abbildung 2.1:** Ablauf $\sigma_1$.

Die Historien der abgebildeten Variablen sind:

$$H_x = 1 \quad 2 \quad 4 \quad 5 \quad 10,$$
$$H_y = 1 \quad 1 \quad 2 \quad 2.$$

Es gilt: $length(H_x) = 5$ und $length(H_y) = 4$ sowie $H_x(0) = 1$, $H_x(1) = 2$, $H_x(2) = 4$ usw.

Jede Transition in $\sigma_1$ ordnet den involvierten Variablen Indizes in ihrer Historie zu:

$$\begin{aligned} t_1 : \{x\} &\to \mathbb{N} \quad \text{mit } x \mapsto 0, \\ t_2 : \{x, y\} &\to \mathbb{N} \quad \text{mit } x \mapsto 1 \text{ und } y \mapsto 0, \\ t_3 : \{x\} &\to \mathbb{N} \quad \text{mit } x \mapsto 2, \\ t_4 : \{y\} &\to \mathbb{N} \quad \text{mit } y \mapsto 1, \\ t_5 : \{x, y\} &\to \mathbb{N} \quad \text{mit } x \mapsto 3 \text{ und } y \mapsto 2. \end{aligned}$$

Dabei gilt: $t_1 \prec t_2$, $t_2 \prec t_3$, $t_2 \prec t_4$, $t_3 \prec t_5$ und $t_4 \prec t_5$. Allerdings gilt weder $t_3 \prec t_4$, noch $t_4 \prec t_3$: $t_3$ und $t_4$ sind also nebenläufig.

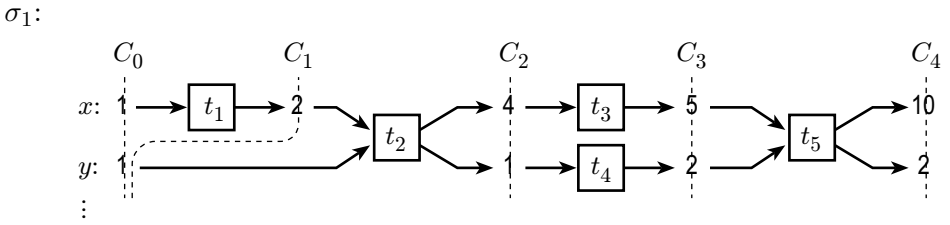


**Abbildung 2.2:** Schnitte und Schritte in $\sigma_1$.

Abbildung 2.2 zeigt alle Schnitte des Ablaufes $\sigma_1$, die durch Schritte vom Anfangsschnitt $C_0$ aus erreicht werden können. Für diese Schritte gilt:

$$\begin{aligned} S_{C_0} &= (C_0, C_1, \{t_1\}), \\ S_{C_1} &= (C_1, C_2, \{t_2\}), \\ S_{C_2} &= (C_2, C_3, \{t_3, t_4\}), \\ S_{C_3} &= (C_3, C_4, \{t_5\}). \end{aligned}$$

## 2.3 Syntax

Wir beschreiben nun die Syntax von TLDA. Grundlage ist das Alphabet von TLDA, auf dem dann induktiv Terme, Schrittformeln und schließlich Ablaufformeln definiert werden.

**Definition 9 (Alphabet)**
*Das* Alphabet von TLDA *besteht aus vier unendlichen, disjunkten Mengen:*

- *eine Menge von Funktionssymbolen $\mathcal{F}$,*
- *eine Menge von Relationssymbolen $\mathcal{R}$,*
- *eine Menge spezieller Symbole Symb,*
- *eine Variablenmenge $\mathcal{V}ar_{all}$.*

*Jedes Funktions- und Relationssymbol besitzt eine Stelligkeit. Nullstellige Funktionssymbole beschreiben Konstanten.* ⌟

**Notation:** Für zweistellige Relationen benutzen wir oft die Infix-Schreibtweise und schreiben $aRb$ anstelle von $R(a, b)$. Dabei nennen wir $a$ die linke Seite und $b$ die rechte Seite von $R$.

Zunächst betrachten wir die Mengen der speziellen Symbole *Symb* und die Variablenmenge $\mathcal{V}ar_{all}$ genauer.

**Definition 10 (Menge der speziellen Symbole *Symb*)**
*Die Menge Symb enthalte die booleschen Operatoren* $\neg$, $\wedge$, $\Rightarrow$, *die Quantoren* $\exists$ *und* $\widetilde{\exists}$, *den Temporaloperator* $\Box$ *sowie Klammern.* ⌟

**Definition 11 (Variablenmenge $\mathcal{V}ar_{all}$)**
*Die Variablenmenge* $\mathcal{V}ar_{all}$ *ist partitioniert in:*

- *rigide Variablen:* $\mathcal{V}ar_{rigid} = \{\mathsf{m}, \mathsf{n}, \ldots\}$,
- *flexible Variablen:* $\mathcal{V}ar = \{x, y, \ldots\}$,
- *gestrichene flexible Variablen:* $\mathcal{V}ar' = \{x' \mid x \in \mathcal{V}ar\}$,
- $\sim$*-Variablen:* $\widetilde{\mathcal{V}ar} = \{\widetilde{a} \mid \emptyset \neq a \subseteq .\mathcal{V}ar\}$

⌟

Die ersten drei Variablenmengen sind auch aus anderen Formalismen, wie beispielsweise TLA [Lam94], bekannt. Neu sind hingegen die $\sim$-Variablen, die aus Mengen von flexiblen Variablen gebildet werden.

**Notation:** Für $\widetilde{\{x\}}$ und $\widetilde{\{x, y\}}$ verwenden wir die Kurzschreibweise $\widetilde{x}$ und $\widetilde{xy}$.

Aufbauend auf den Variablenmengen, Funktions- und Relationssymbolen sowie den speziellen Symbolen aus *Symb* definieren wir nun:

**Definition 12 (Terme)**
*Die Menge der* Terme *ist wie folgt induktiv definiert:*

- *Variablen aus* $\mathcal{V}ar_{rigid} \cup \mathcal{V}ar \cup \mathcal{V}ar'$ *sind Terme.*
- *Wenn* $t_1, \ldots, t_n$ *Terme sind und* $f \in \mathcal{F}$ *ein* $n$*-stelliges Funktionssymbol mit* $n \geq 0$, *so ist auch* $f(t_1, \ldots, t_n)$ *ein Term.*

⌟

**Definition 13 (Schrittformeln)**
*Die Menge der* Schrittformeln *ist wie folgt induktiv definiert:*

- *Variablen aus* $\widetilde{\mathcal{V}ar}$ *sind Schrittformeln.*
- *Wenn* $t_1, \ldots, t_n$ *Terme sind und* $R \in \mathcal{R}$ *ein n-stelliges Relationssymbol mit* $n \geq 1$*, so ist* $R(t_1, \ldots, t_n)$ *eine Schrittformel.*
- *Wenn* $\varphi$ *und* $\psi$ *Schrittformeln sind, so sind auch* $\neg\varphi$*,* $(\varphi \wedge \psi)$ *und* $(\varphi \Rightarrow \psi)$ *Schrittformeln.*
- *Wenn* $\mathsf{m} \in \mathcal{V}ar_{rigid}$ *und* $\varphi$ *eine Schrittformel ist, so ist auch* $\exists \mathsf{m}\ \varphi$ *eine Schrittformel.*
- *Wenn* $\varphi$ *eine Schrittformel ist, so auch* $\widetilde{\exists}\varphi$*.*

⌟

**Notation:** Wir schreiben griechische Kleinbuchstaben für Schrittformeln.

**Beispiel 1.** Folgende Formeln sind Schrittformeln:

$$x = 4 \wedge y' = 2, \qquad \widetilde{\exists}(x' = \mathsf{m}), \qquad \widetilde{xy} \Rightarrow (x' = 2 \cdot x \wedge y' = y).$$

⌟

**Definition 14 (Ablaufformeln)**
*Die Menge der* Ablaufformeln *ist wie folgt induktiv definiert:*

- *Schrittformeln sind Ablaufformeln.*
- *Wenn* $\Phi$ *und* $\Psi$ *Ablaufformeln sind, so sind auch* $\neg\Phi$*,* $(\Phi \wedge \Psi)$ *und* $(\Phi \Rightarrow \Psi)$ *Ablaufformeln.*
- *Wenn* $\Phi$ *eine Ablaufformel ist, so auch* $\Box\Phi$*.*
- *Wenn* $\mathsf{m} \in \mathcal{V}ar_{rigid}$ *und* $\Phi$ *eine Ablaufformel ist, so ist auch* $\exists \mathsf{m}\ \Phi$ *eine Ablaufformel.*

⌟

**Notation:** Wir schreiben griechische Großbuchstaben für Ablaufformeln.

**Beispiel 2.** Folgende Formeln sind Ablaufformeln:

$$\Box(x \geq y), \qquad x = 1 \wedge \Box(\widetilde{x} \Rightarrow x' > x).$$

⌟

**Definition 15 (Zustandsformel)**
*Eine* Zustandsformel *ist eine Ablaufformel, in der keine Variablen aus* $\mathcal{V}ar' \cup \widetilde{\mathcal{V}ar}$ *und keine* $\widetilde{\exists}$*-Quantoren vorkommen.* ⌟

## 2.4 Semantik

Wir wollen im Folgenden die formale Semantik für die Logik TLDA angeben. Dazu werden die Formeln analog zu ihrem induktiven syntaktischen Aufbau ausgewertet.

**Definition 16 (Interpretation)**
*Eine* Interpretation $I$ *von* $(\mathcal{F}, \mathcal{R})$ *über* $\mathcal{V}al$ *enthält:*

- *eine nichtleere Menge* $\mathcal{V}al$*, das* Universum*,*
- *für jedes n-stellige Funktionssymbol* $f \in \mathcal{F}$ *eine Funktion* $f^I : \mathcal{V}al^n \to \mathcal{V}al$*,*
- *für jedes n-stellige Relationssymbol* $R \in \mathcal{R}$ *eine Relation* $R^I \subseteq \mathcal{V}al^n$*.*

⌟

Wir definieren zunächst die Auswertung der rigiden Variablen.

**Definition 17 (Belegung, Belegungsvariante)**
*Eine Abbildung* $\beta : \mathcal{V}ar_{rigid} \to \mathcal{V}al$ *heißt* Belegung für $\mathcal{V}ar_{rigid}$*.*

*Seien* $\mathsf{m} \in \mathcal{V}ar_{rigid}$ *und* $q \in \mathcal{V}al$*. Eine Belegung* $\beta[\mathsf{m} \leftarrow q]$ *heißt* Belegungsvariante von $\beta$ bzgl. $\mathsf{m}$*, wenn gilt:* $\beta[\mathsf{m} \leftarrow q](\mathsf{m}) = q$ *und* $\beta[\mathsf{m} \leftarrow q](\mathsf{k}) = \beta(\mathsf{k})$ *für alle* $\mathsf{k} \in \mathcal{V}ar_{rigid} \backslash \{\mathsf{m}\}$*.*

⌟

Aufbauend auf einer Interpretation, einer Belegung und einer Historie werden Terme ausgewertet:

**Definition 18 (Auswertung von Termen)**
*Seien* $\sigma = (H, T)$ *ein Ablauf,* $C$ *ein Schnitt in* $\sigma$ *und* $C'$ *sein Folgeschnitt. Seien* $I$ *eine Interpretation von* $(\mathcal{F}, \mathcal{R})$ *über* $\mathcal{V}al$ *und* $\beta$ *eine Belegung für* $\mathcal{V}ar_{rigid}$*. Der* Wert eines Termes *im Schnitt* $C$*, notiert durch eine Abbildung* $I_C$*, die Terme auf Werte aus* $\mathcal{V}al$ *abbildet, ist wie folgt definiert:*

- $I_C(\mathsf{m}) = \beta(\mathsf{m})$*, falls* $\mathsf{m} \in \mathcal{V}ar_{rigid}$*,*
- $I_C(x) = H_x(C(x))$*, falls* $x \in \mathcal{V}ar$*,*
- $I_C(x') = H_x(C'(x))$*, falls* $x' \in \mathcal{V}ar'$*, und*
- $I_C(f(t_1, \ldots, t_n)) = f^I(I_C(t_1), \ldots, I_C(t_n))$*, falls* $f \in \mathcal{F}$ *ein n-stelliges Funktionssymbol ist und* $t_1, \ldots, t_n$ *Terme sind.*

⌟

Schrittformeln werden in Schritten ausgewertet:

**Definition 19 (Auswertung von Schrittformeln)**
*Seien $\sigma = (H, T)$ ein Ablauf und $S_C = (C, C', T_C)$ ein Schritt in $\sigma$. Seien $I$ eine Interpretation von $(\mathcal{F}, \mathcal{R})$ über $\mathcal{V}al$ und $\beta$ eine Belegung für $\mathcal{V}ar_{rigid}$. Die Gültigkeit einer Schrittformel $\varphi$ im Schritt $S_C$ unter der Belegung $\beta$ und der Interpretation $I$, notiert durch $(S_C, I) \Vdash_\beta \varphi$, ist wie folgt definiert:*

- $(S_C, I) \Vdash_\beta \widetilde{a}$, *falls* $a \subseteq dom(t)$ *für ein* $t \in T_C$,
- $(S_C, I) \Vdash_\beta R(t_1, \ldots, t_n)$, *falls* $R \in \mathcal{R}$ *ein n-stelliges Relationssymbol ist,* $t_1, \ldots, t_n$ *Terme sind und es gilt:* $(I_C(t_1), \ldots, I_C(t_n)) \in R^I$,
- $(S_C, I) \Vdash_\beta \neg\varphi$, *falls nicht gilt:* $(S_C, I) \Vdash_\beta \varphi$,
- $(S_C, I) \Vdash_\beta (\varphi \wedge \psi)$, *falls gilt:* $(S_C, I) \Vdash_\beta \varphi$ *und* $(S_C, I) \Vdash_\beta \psi$,
- $(S_C, I) \Vdash_\beta (\varphi \Rightarrow \psi)$, *falls gilt:* $(S_C, I) \Vdash_\beta \neg\varphi$ *oder* $(S_C, I) \Vdash_\beta \psi$,
- $(S_C, I) \Vdash_\beta \exists \mathsf{m}\ \varphi$, *falls* $\mathsf{m} \in \mathcal{V}ar_{rigid}$ *und es einen Wert* $q \in \mathcal{V}al$ *gibt, sodass* $(S_C, I) \Vdash_{\beta[\mathsf{m} \leftarrow q]} \varphi$, *und*
- $(S_C, I) \Vdash_\beta \widetilde{\exists}\varphi$, *falls es einen Ablauf* $\sigma^* = (H^*, T^*)$ *gibt mit* $H_x(C(x)) = H^*_x(C(x))$ *für alle* $x \in \mathcal{V}ar$ *und* $C$ *ein Schnitt in* $\sigma^*$ *ist mit* $(S^*_C, I) \Vdash_\beta \varphi$.

⌟

Da sowohl die $\sim$-Variablen als auch der $\widetilde{\exists}$-Quantor in TLDA neu sind, beschreiben wir noch einmal ihre Bedeutung:

- Die Variable $a \in \widetilde{\mathcal{V}ar}$ ist eine boolesche Variable, die in einem Schritt genau dann wahr ist, falls im Schritt genau eine Transition schaltet, die gleichzeitig all Variablen $x \in a$ involviert.
- Der $\widetilde{\exists}$-Quantor quantifiziert eine gesamte Schrittformel $\varphi$, und der Ausdruck $\widetilde{\exists}\varphi$ ist genau dann wahr, wenn ein Ablauf existiert, der mindestens im Schnitt $C$ mit $\sigma$ übereinstimmt und in dem ein in $C$ anfangender Schritt existiert, der $\varphi$ erfüllt.

**Notation:** Wenn die Interpretation von $(\mathcal{F}, \mathcal{R})$ über $\mathcal{V}al$ und die Belegung für $\mathcal{V}ar_{rigid}$ aus dem Kontext klar sind, schreiben wir kurz $S_C \Vdash \varphi$ anstelle von $(S_C, I) \Vdash_\beta \varphi$.

**Beispiel 3.** Wir wollen nun einige Schrittformeln anhand des in Abbildung 2.2 angegebenen Ablaufes $\sigma_1$ auswerten und konzentrieren uns auf die durch $\sim$-Variablen gebildeten Schrittformeln. Es gilt:

| | |
|---|---|
| $S_{C_1} \Vdash \widetilde{xy}$ | da $\{x, y\} \subseteq dom(t_2)$ und $t_2 \in T_{C_1}$ |
| $S_{C_2} \Vdash \widetilde{x} \wedge \widetilde{y}$ | da $x \in dom(t_3)$, $y \in dom(t_4)$ und $t_3, t_4 \in T_{C_2}$ |
| $S_{C_2} \Vdash \neg\widetilde{xy}$ | da es keine Transition $t \in T_{S_{C_2}}$ gibt mit $\{x, y\} \subseteq dom(t)$ |

⌟

Die aus $\sim$-Variablen gebildeten Schrittformeln haben in Schritten gewisse Eigenschaften, die wir an dieser Stelle festhalten wollen und zu einem späteren Zeitpunkt (Abschnitt 4.6) zur Überprüfung von Schrittformeln auf Widerspruchsfreiheit benutzen werden.

**Korollar 2.1 (Eigenschaften von $\sim$-Variablen in Schritten)**
*Seien $S_C$ ein Schritt, $\widetilde{a}, \widetilde{b} \in \widetilde{\mathcal{V}ar}$ und $x \in \mathcal{V}ar$. Es gilt:*

1. $S_C \Vdash \left(\widetilde{a} \wedge \widetilde{b}\right)$, *falls* $S_C \Vdash \widetilde{c}$, $c = a \cup b$ *(Zerlegung)*
2. $S_C \Vdash \widetilde{c}$, *falls* $S_C \Vdash \left(\widetilde{a} \wedge \widetilde{b}\right)$, $c = a \cup b$ *und* $a \cap b \neq \emptyset$ *(Transitivität)*
3. $S_C \Vdash \neg\widetilde{b}$, *falls* $S_C \Vdash \neg\widetilde{a}$ *und* $a \subseteq b$ *(Erweiterung)*
4. $S_C \Vdash x' = x$, *falls* $S_C \Vdash \neg\widetilde{x}$ *(Werterhalt)*

⌟

**Beispiel 4.** Seien $S_C$ ein Schritt und $w, x, y, z \in \mathcal{V}ar$. Es gelte $S_C \Vdash (\neg\widetilde{w} \wedge \widetilde{xy} \wedge \widetilde{yz})$. Nach Korollar 2.1 gilt:

- $S_C \Vdash (\widetilde{x} \wedge \widetilde{y} \wedge \widetilde{z})$ (Zerlegung)
- $S_C \Vdash \widetilde{xyz}$ (Transitivität)
- $S_C \Vdash (\neg\widetilde{wx} \wedge \neg\widetilde{wy} \wedge \neg\widetilde{wz} \wedge \neg\widetilde{wxy} \wedge \neg\widetilde{wxz} \wedge \neg\widetilde{wyz} \wedge \neg\widetilde{wxyz})$ (Erweiterung)
- $S_C \Vdash w' = w$ (Werterhalt)

⌟

Da in Ablaufformeln auch der $\Box$-Temporaloperator vorkommen kann, können Ablaufformeln im Gegensatz zu Schrittformeln nicht in nur einem einzigen Schritt ausgewertet werden. Vielmehr ist es hier notwendig, einen in einem Schnitt beginnenden Ablauf zu betrachten:

**Definition 20 (Auswertung von Ablaufformeln)**
*Seien $\sigma$ ein Ablauf und $C$ ein Schnitt in $\sigma$. Seien $I$ eine Interpretation von $(\mathcal{F}, \mathcal{R})$ über $\mathcal{V}al$ und $\beta$ eine Belegung für $\mathcal{V}ar_{rigid}$. Die Gültigkeit einer Ablaufformel $\Phi$ im Schnitt $C$ unter der Belegung $\beta$ und der Interpretation $I$, notiert durch $(\sigma, C, I) \models_\beta \Phi$, ist wie folgt definiert:*

- $(\sigma, C, I) \models_\beta \varphi$, *falls $\varphi$ eine Schrittformel ist und es gilt:* $(S_C, I) \Vdash_\beta \varphi$,
- $(\sigma, C, I) \models_\beta \neg\Phi$, *falls nicht gilt:* $(\sigma, C, I) \models_\beta \Phi$,
- $(\sigma, C, I) \models_\beta (\Phi \wedge \Psi)$, *falls gilt:* $(\sigma, C, I) \models_\beta \Phi$ *und* $(\sigma, C, I) \models_\beta \Psi$,
- $(\sigma, C, I) \models_\beta (\Phi \Rightarrow \Psi)$, *falls gilt:* $(\sigma, C, I) \models_\beta \neg\Phi$ *oder* $(\sigma, C, I) \models_\beta \Psi$,

- $(\sigma, C, I) \models_\beta \Box\Phi$, *falls für alle Schnitte* $C^*$ *mit* $C^*(x) \geq C(x)$ *für alle* $x \in \mathcal{V}ar$ *gilt:* $(\sigma, C^*, I) \models_\beta \Phi$,
- $(\sigma, C, I) \models_\beta \exists \mathsf{m}\ \Phi$, *falls* $\mathsf{m} \in \mathcal{V}ar_{rigid}$ *und es einen Wert* $q \in \mathcal{V}al$ *gibt, sodass gilt:* $(\sigma, C, I) \models_{\beta[\mathsf{m} \leftarrow q]} \Phi$.

⌟

**Notation:** Wenn die Interpretation von $(\mathcal{F}, \mathcal{R})$ über $\mathcal{V}al$ und die Belegung für $\mathcal{V}ar_{rigid}$ aus dem Kontext klar ist, schreiben kurz $(\sigma, C) \models \Phi$ anstelle von $(\sigma, C, I) \models_\beta \varphi$. Außerdem schreiben wir $\sigma \models \Phi$ statt $(\sigma, C_0) \models \Phi$.

**Beispiel 5.** Wir wollen nun einige Beispiele für Schrittformeln geben, die wir anhand des Ablaufes $\sigma_1$ aus Abbildung 2.2 auswerten.

$$\sigma_1 \models \Box(x \geq y)$$
$$\sigma_1 \models x = 1 \wedge \Box(\widetilde{x} \Rightarrow x' > x)$$

⌟

Nun definieren wir einen für diese Arbeit zentralen Begriff:

**Definition 21 (Halbordnungssemantik)**
*Die* endliche Halbordnungssemantik *einer Ablaufformel* $\Phi$ *ist die Menge der endlichen Abläufe* $\sigma$*, für die gilt:* $\sigma \models \Phi$. *Die* unendliche Halbordnungssemantik *einer Ablaufformel* $\Phi$ *ist die Menge der unendlichen Abläufe* $\sigma$*, für die gilt:* $\sigma \models \Phi$. ⌟

Der Aspekt der halbgeordneten Transitionen wurde schon erläutert. Wir werden im folgenden Kapitel eine weitere Semantik, eine Interleavingsemantik, für TLDA ausarbeiten und diesen Aspekt erneut aufgreifen.

## 2.5 Weitere Definitionen

Ziel dieser Arbeit ist die Implementierung einer Interleavingsemantik, also der automatische Aufbau eines Transitionssystemes, mit dessen Hilfe später Eigenschaften von Systemen überprüft werden können. Dabei betrachten wir solche Systeme, deren Verhalten durch eine TLDA-Formel beschrieben, d.h. spezifiziert wurden. Im Folgenden verstehen wir unter einer Spezifikation also eine Formel, deren Abläufe Modelle des Verhaltens des spezifizierten Systemes sind.

Dazu definieren wir zunächst einen speziellen Aufbau von Spezifikationen, die sog. Normalform:

**Definition 22 (Normalform)**
*Eine Spezifikation* $\Phi$ *befindet sich in* Normalform*, wenn sie die Form*

$$\Phi \triangleq \mathit{Init} \wedge \Box \mathit{Next} \wedge \mathit{Liveness}$$

*hat. Dabei ist Init eine Zustandsformel, Next eine Schrittformel und Liveness eine Ablaufformel, die Lebendigkeitseigenschaften des spezifizierten Systemes beschreibt.* ⌟

**Anmerkung:** Da der Fokus dieser Arbeit auf dem Aufbau eines Transitionssystemes, nicht jedoch auf der Überprüfung von Systemeigenschaften liegt, werden die spezifizierten Lebendigkeitseigenschaften im Folgenden ignoriert.

**Beispiel 6.** Die Formel $\Phi \triangleq \mathit{Init} \wedge \Box \mathit{Next}$ mit

$$\begin{aligned} \mathit{Init} &\triangleq x = 1 \wedge y = 1 \\ \mathit{Next} &\triangleq \big(\widetilde{x} \Rightarrow \big((x' = x + 1 \wedge \neg\widetilde{xy}) \vee (x' = x \cdot 2 \wedge \widetilde{xy})\big)\big) \wedge \\ &\quad \big(\widetilde{y} \Rightarrow \big((y' = y + 1 \wedge \neg\widetilde{xy}) \vee (y' = y \wedge \widetilde{xy})\big)\big) \end{aligned}$$

ist eine Spezifikation in Normalform. Für den Ablauf $\sigma_1$ (s. Abb. 2.1) gilt: $\sigma_1 \models \Phi$. ⌟

Die Normalform ist kein rein syntaktisches Kriterium, d.h. nicht jede Spezifikation lässt sich in Normalform umformen. Jedoch zeigt sich, dass die Einschränkungen bei der Spezifizierung „realer“ Systeme keine Einschränkung darstellt. Dennoch vereinfacht der spezielle Aufbau von Spezifikationen in Normalform die Konstruktion eines Transitionssystemes, wie sich im folgenden Kapitel zeigt.

Ein Ablauf ist stets für alle Variablen aus $\mathcal{V}ar$ definiert, jedoch kann diese unendliche Variablenmenge anhand der Spezifikation, der der Ablauf zugrunde liegt, partitioniert werden:

**Definition 23 (Systemvariablen, Umgebung)**
*Die Menge der* Systemvariablen $V(\Phi) \subset \mathcal{V}ar$ *einer Spezifikation* $\Phi$ *ist die (endliche) Menge aller Variablen, die in* $\Phi$ *vorkommen:*

$$V(\Phi) \triangleq \{x \in \mathcal{V}ar \mid x, x' \text{ oder } \widetilde{a} \text{ kommt in } \Phi \text{ vor und } x \in a\}.$$

*Die Menge* $\mathcal{V}ar \backslash V(\Phi)$ *nennen wir* Umgebung von $\Phi$. ⌟

Wir wollen diese Partitionierung auf Abläufe anwenden und nur das Verhalten der Systemvariablen betrachten, während wir die Umgebung der zugrunde liegenden Spezifikation „ausblenden“. Dazu definieren wir den Begriff der Restriktion.

**Definition 24 (Restriktion einer Abbildung)**
*Für eine Abbildung* $f : A \to B$ *und eine Teilmenge* $A' \subseteq A$ *bezeichnet* $f_{|A'} : A' \to B$ *die* Restriktion der Abbildung $f$ auf $A'$. ⌟

**Definition 25 (Restriktion eines Ablaufes)**
*Seien $\sigma = (H, T)$ ein Ablauf und $\mathcal{A} \subset \mathcal{V}ar$ eine endliche Variablenmenge. $\sigma_{|\mathcal{A}} = (H_{|\mathcal{A}}, T_{|\mathcal{A}})$ heißt* Restriktion von $\sigma$ auf $\mathcal{A}$ *genau dann, wenn*

1. *$H_{|\mathcal{A}}$ ist die Restriktion von $H$ auf $\mathcal{A}$,*
2. *$T_{|\mathcal{A}} \triangleq \{t_{|\mathcal{B}} \mid \emptyset \neq \mathcal{B} = \mathcal{A} \cap dom(t)$ für ein $t \in T\}$ ist eine Transitionenmenge,*
3. *$\sigma_{|\mathcal{A}}$ ist ein Ablauf, d.h. erfüllt die Bedingungen 1-3 aus Def. 3.*

⌟

**Beispiel 7.** Abbildung 2.3 zeigt die Restriktion von $\sigma_1$ auf $\{x\}$.

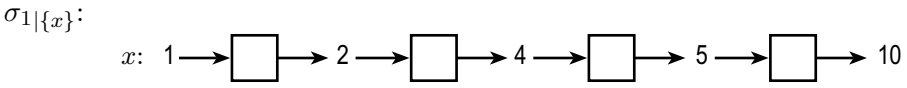


**Abbildung 2.3:** Restriktion von $\sigma_1$ auf $\{x\}$.

⌟

Die Analyse von Restriktionen von Abläufen auf die Systemvariablen ist nur dann gerechtfertigt, wenn die Eigenschaften der zugrunde liegenden Spezifikation erhalten bleiben. Für eine bestimmte Klasse von Spezifikationen ist dies der Fall:

**Definition 26 (Umgebungsinvarianz)**
*Eine Spezifikation $\Phi$ heißt* umgebungsinvariant, *wenn für alle Abläufe $\sigma, \sigma^*$ mit $\sigma_{|V(\Phi)} = \sigma^*_{|V(\Phi)}$ gilt: $\sigma \models \Phi$ gdw. $\sigma^* \models \Phi$.* ⌟

Die Eigenschaft umgebungsinvarianter Spezifikationen, losgelöst von ihrer Umgebung und nur auf den Systemvariablen ausgewertet werden zu können, wird bei der Definition einer Interleavingsemantik für TLDA im folgenden Kapitel ausgenutzt.

# 3 Interleavingsemantik

Nachdem wir in Kapitel 2 eine Einführung in TLDA gegeben haben, werden wir in diesem Kapitel eine Interleavingsemantik ausarbeiten. Dazu beschreiben wir in Abschnitt 3.1 zunächst den Unterschied zwischen Halbordnungs- und Interleavingsemantiken und deren Bedeutung bei der computergestützten Verifikation von Systemen. In Abschnitt 3.2 analysieren wir einen bestehenden Ansatz einer TLDA-Interleavingsemantik und diskutieren dessen Eignung zum Modelchecking von TLDA-Spezifikationen. Wir zeigen anschließend, dass die Wahl eines anderen Aktionsbegriffes die auftretenden Probleme löst und definieren in Abschnitt 3.3 eine schrittbasierte Interleavingsemantik. Zuletzt zeigen wir in Abschnitt 3.4, wie wir mit unserem Zustands- und Aktionsbegriff Transitionssysteme für TLDA-Spezifikationen aufbauen können und beschließen das Kapitel mit der formalen Definition der TLDA-Interleavingsemantik.

## 3.1 Interleavingsemantiken und Modelchecking

Ein Zustand charakterisiert im Allgemeinen die relevanten Größen eines Systemes zu einem bestimmten Zeitpunkt und wird meist mithilfe von Variablen beschrieben. Ein Zustand ist also eine Belegung der Variablen des Systemes, kann also als Wertetupel verstanden werden. Eine Aktion beschreibt den Übergang von einem Zustand zu seinem Nachfolgezustand, d.h. ändert bzw. aktualisiert Variablenwerte.

$$\begin{bmatrix} x = 1 \\ y = 1 \end{bmatrix} \xrightarrow{a_1} \begin{bmatrix} x = 2 \\ y = 2 \end{bmatrix} \xrightarrow{a_2} \begin{bmatrix} x = 3 \\ y = 4 \end{bmatrix}$$

**Abbildung 3.1:** Ein Interleaving mit den Variablen $x$, $y$ und den Aktionen $a_1$, $a_2$.

Ein Interleaving (vgl. Abb. 3.1) ist eine Sequenz von Zustandsübergängen und beschreibt die Veränderungen der Variablenwerte des Systemes im Laufe der Zeit. Das Verhalten eines Systemes kann mit der Menge seiner möglichen Interleavings beschrieben werden. Wir halten dies zunächst informal fest:

**Definition 27 (Interleavingsemantik)**
*Die* Interleavingsemantik *eines Systemes ist die Menge seiner Interleavings.* ⌟

Demgegenüber haben wir in Abschnitt 2.4 (Def. 21) eine Halbordnungssemantik kennengelernt. Dabei bildet die Nachfolgerrelation $\prec$ der Transitionen eine Halbordnung, d.h. es kann sowohl Transitionen geben, die durch $\prec$ geordnet sind (vgl. Ablauf $\sigma_2$ in Abb. 3.2),

als auch nebenläufige Transitionen, die nicht in der $\prec$-Relation stehen (vgl. Ablauf $\sigma_3$ in Abb. 3.3).

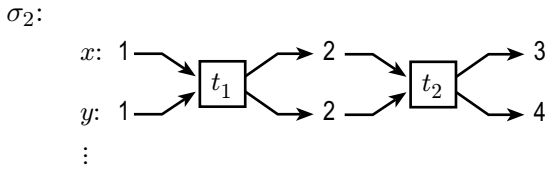


**Abbildung 3.2:** Ablauf $\sigma_2$ mit geordneten Transitionen: $t_1 \prec t_2$.

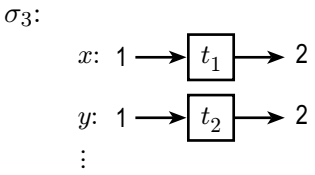


**Abbildung 3.3:** Ablauf $\sigma_3$ mit nebenläufigen Transitionen: $t_1 \not\prec t_2$ und $t_2 \not\prec t_1$.

Die Aktionen und somit auch die Zustände eines Interleavings sind stets total geordnet. Ungeordnete, also nebenläufige, Aktionen können so nicht ausgedrückt werden. Stattdessen muss ursprünglich ungeordnete Nebenläufigkeit in eine willkürliche Ordnung gebracht werden, sodass durch die exponentielle Anzahl möglicher Reihenfolgen entsprechend viele Interleavings das Verhalten des verteilten Systemes beschreiben. Der wichtigste Kritikpunkt von Interleavingsemantiken ist außerdem die exponentielle Anzahl möglicher Zwischenzustände (Zustandsraumexplosion), die durch die Verschränkung (engl. *interleaving*) nebenläufiger Aktionen entsteht.

Dennoch haben Interleavingsemantiken eine große Bedeutung bei der computergestützten Verifikation, da sie — im Gegensatz zu Halbordnungssemantiken, bei denen kaum Ansätze bekannt sind — einfach durch Graphen, sog. Transitionssystemen, repräsentiert werden können, in denen dann explizite Modelchecker (wie z.B. SPIN [Hol97], TLC [YML99] oder LoLA [Sch00]) das Verifikationsproblem durch Suche bestimmter Zustände oder Pfade in diesem Graphen lösen. Für die beschriebenen Probleme, insbesondere die Zustandsraumexplosion, sind bereits viele effiziente Reduktionstechniken bekannt.

## 3.2 Transitionsbasierte TLDA-Interleavingsemantik

Um eine Interleavingsemantik für TLDA anzugeben, müssen wir einen Zustands- und einen Aktionsbegriff ausarbeiten, d.h. wir müssen definieren, wie wir aus TLDA-Spezifikationen oder deren Abläufen Interleavings ableiten können.

Ein einfacher Ansatz wurde in [AR03] angegeben und wird an dieser Stelle kurz erläutert.

Zustände, also Wertetupel, können durch Schnitte erzeugt werden: Ein Schnitt ordnet jeder Variable einen Index in der Historie der jeweiligen Variable zu. Wird der Zustand nur von der (endlichen) Menge der Systemvariablen $V(\Phi)$ einer Spezifikation $\Phi$ beschrieben, legt $H_x(C(x))$ für alle $x \in V(\Phi)$ das durch den Schnitt $C$ erzeugte Wertetupel fest.

Als Aktionen werden Transitionen vorgeschlagen, sodass die resultierende Interleavingsemantik im folgenden transitionsbasierte TLDA-Interleavingsemantik genannt wird. Die Abbildungen 3.4 und 3.5 zeigen die Interleavings der Abläufe $\sigma_2$ (s. Abb. 3.2) und $\sigma_3$ (s. Abb. 3.3).

$$\begin{bmatrix} x=1 \\ y=1 \end{bmatrix} \xrightarrow{t_1} \begin{bmatrix} x=2 \\ y=2 \end{bmatrix} \xrightarrow{t_2} \begin{bmatrix} x=3 \\ y=4 \end{bmatrix}$$

**Abbildung 3.4:** Einziges transitionsbasiertes Interleaving von Ablauf $\sigma_2$.

$$\begin{bmatrix} x=1 \\ y=1 \end{bmatrix} \xrightarrow{t_1} \begin{bmatrix} x=2 \\ y=1 \end{bmatrix} \xrightarrow{t_2} \begin{bmatrix} x=2 \\ y=2 \end{bmatrix}$$

$$\begin{bmatrix} x=1 \\ y=1 \end{bmatrix} \xrightarrow{t_2} \begin{bmatrix} x=1 \\ y=2 \end{bmatrix} \xrightarrow{t_1} \begin{bmatrix} x=2 \\ y=2 \end{bmatrix}$$

**Abbildung 3.5:** Alle möglichen transitionsbasierten Interleavings von Ablauf $\sigma_3$.

Die transitionsbasierte TLDA-Interleavingsemantik hat jedoch entscheidende Nachteile:

- Der Effekt einer Transition wird nur durch den zugrunde liegenden Ablauf deutlich, der explizit gegeben werden müsste. Für den automatischen Aufbau eines Transitionssystemes müssen Aktionen jedoch rein syntaktisch anhand einer gegebenen Spezifikation abgeleitet werden können.
- Informationen über Nebenläufigkeit gehen verloren, da nebenläufige Transitionen in eine Reihenfolge gebracht werden. Die mögliche gleichzeitige Aktualisierung der Variablen $x$ und $y$, die in Ablauf $\sigma_3$ (s. Abb. 3.3) deutlich ist, kann in den Interleavings (s. Abb. 3.5) nicht abgelesen werden.
- Das Verifizieren von Systemeigenschaften, die in TLDA formuliert sind, ist anhand der Interleavings nur sehr schwer möglich. Für die Beispielabläufe $\sigma_2$ und $\sigma_3$ gilt:

  $$\sigma_2 \models (x' = 2 \wedge y' = 2) \qquad \sigma_3 \models (x' = 2 \wedge y' = 2) \tag{3.1}$$

  Diese Formel muss im Schritt $S_{C_0}$ der jeweiligen Abläufe ausgewertet werden. Allerdings können aus den Interleavings keine Informationen über Schritte gewonnen werden, da der Schritt $S_{C_0}$ im Interleaving von Ablauf $\sigma_2$ (s. Abb. 3.4) einem Zustandsübergang, in den Interleavings von Ablauf $\sigma_3$ (s. Abb. 3.5) jedoch zwei Zustandsübergängen entspricht.

Außerdem gilt in den Beispielabläufen:

$$\sigma_2 \models \widetilde{xy} \qquad \sigma_3 \models \neg\widetilde{xy} \tag{3.2}$$

Auch diese Formeln lassen sich nur schwierig (wenn nicht unmöglich) anhand der Interleavings auswerten.

Im folgenden Abschnitt geben wir eine andere TLDA-Interleavingsemantik an, die die hier beschriebenen Probleme löst. Weitere Probleme werden in [AR03] beschrieben, auf die wir in dieser Arbeit jedoch nicht eingehen werden.

## 3.3 Schrittbasierte TLDA-Interleavingsemantik

Die beschriebenen Probleme der transitionsbasierten Interleavingsemantik entspringen dem gewählten Aktionsbegriff. Um die Probleme zu lösen bzw. zu umgehen, arbeiten wir einen anderen Aktionsbegriff aus, der zusammen mit dem in Abschnitt 3.2 beschriebenen Zustandsbegriff aus [AR03] eine Interleavingsemantik für TLDA definiert.

Wie beschrieben, betrachten wir Spezifikationen $\Phi \triangleq \mathit{Init} \wedge \Box\mathit{Next}$ in Normalform. In jedem Ablauf von $\Phi$ muss also, beim Anfangsschnitt beginnend, jeder Schritt die Schrittformel *Next* erfüllen. Wir verfolgen nun den Ansatz, die durch *Next* beschriebenen Schritte als Grundlage für einen Aktionsbegriff zu betrachten und so eine schrittbasierte Interleavingsemantik zu definieren.

Um diese Aktionen syntaktisch aus der gegebenen Spezifikation abzuleiten, wird *Next* in disjunktive Normalform umgeformt. Jeder Schritt, der *Next* erfüllt, erfüllt eine Menge von Klauseln der disjunktiven Normalform von *Next*. Jede dieser Klauseln beschreibt den Übergang von einem Schnitt zu seinem Nachfolgeschnitt. Im Folgenden sei also die Menge aller Klauseln der disjunktiven Normalform von *Next* die Menge der Aktionen.

Gegenüber [AR03] betrachten wir nun nicht einzelne Transitionen, sondern jeweils die maximale Menge aller Transitionen, die in einem Schnitt schalten können, als Aktionen. Somit „überspringen“ wir Schnitte (vgl. Schnitt $C^*$ in Abbildung 3.6, der zwischen $C$ und dessen Nachfolgeschnitt $C'$ liegt), die durch Schalten nur einiger Transitionen erreicht werden könnten. Diese Schnitte würden nicht durch Zustände in Interleavings repräsentiert werden, sodass eine Analyse des spezifizierten Systemes anhand der Interleavings nicht möglich wäre.

Um den Schnitt $C^*$ zu erreichen, darf von $C$ aus lediglich die Transition $t_1$, nicht aber $t_2$ schalten. Somit ist $C^*$ nur dann durch Schritte erreichbar, wenn das Schalten von $t_2$ im Schnitt $C$ nicht möglich ist.

Wie beschrieben ist in Abbildung 3.6 nur ein Teil der Variablen dargestellt — die Umgebung von $\sigma_4$ ist ausgeblendet. In dieser Umgebung kann es jedoch eine Variable *env* geben, die durch geeignete Synchronisation das Schalten von $t_2$ hinauszögert. Abbildung 3.7 zeigt einen Ablauf $\sigma_4^*$ mit dem gewünschten Verhalten: Die Umgebungsvariable *env* ist

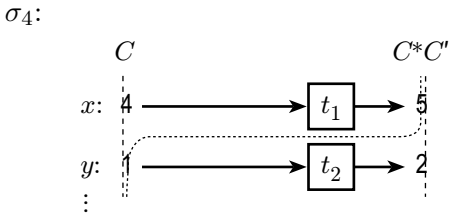


**Abbildung 3.6:** Ablauf $\sigma_4$ mit Schnitt $C^*$.

in $t_2$ involviert, allerdings kann $t_2$ nicht in $C$ schalten, da zuvor $u$ schalten muss ($u \prec t_2$). Somit ist $C^*$ in $\sigma_4^*$ durch den Schritt $S_C = (C, C^*, \{t_1, u\})$ erreichbar. Der schwarze Punkt (•) steht für einen beliebigen Wert von *env*.

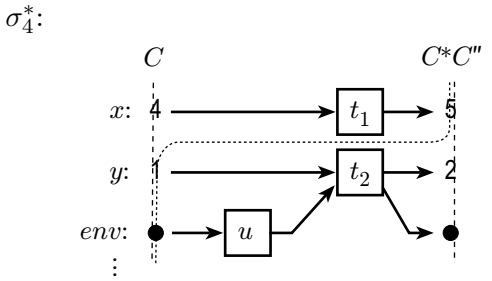


**Abbildung 3.7:** Ablauf $\sigma_4^*$ mit Umgebungsvariable *env*.

Allerdings ist $\sigma_4^*$ nicht notwendigerweise ein Ablauf der Spezifikation $\Phi$, der $\sigma_4$ zugrunde lag. Nimmt man jedoch an, dass $\Phi$ umgebungsinvariant ist, ist auch $\sigma_4^*$ ein Ablauf von $\Phi$ gdw. $\sigma_4$ ein Ablauf von $\Phi$ ist (vgl. Def. 26). Der beschriebene Aktionsbegriff erlaubt es also, eine Interleavingsemantik für Systeme, die mit umgebungsinvarianten Formeln in Normalform spezifiziert wurden, zu definieren.

Die Abläufe $\sigma_2$ (Abb. 3.2) und $\sigma_3$ (Abb. 3.3) lassen sich durch umgebungsinvariante Spezifikationen in Normalform beschreiben. Am Beispiel des Ablaufes $\sigma_3$ können wir nun grob zeigen, wie die in Abschnitt 3.2 angesprochenen Probleme der transitionsbasierten Interleavingsemantik gelöst wurden. Abbildung 3.8 zeigt die resultierenden Interleavings von Ablauf $\sigma_3$. Wir gehen an dieser Stelle nicht auf die Ableitung der Aktionen ein, jedoch ist offensichtlich, dass durch gleichzeitiges Schalten der nebenläufigen Transitionen $t_1$ und $t_2$ im Schritt $S_{C_0}$ in $\sigma_3$ ein drittes Interleaving entsteht.

Dieses Interleaving ermöglicht die Überprüfung von (3.1). Des Weiteren können weitere Eigenschaften (z.B. die Involviertheit der Systemvariablen, die zur Auswertung von (3.2) notwendig ist) aus den Aktionen abgleitet werden.

Da die computergestützte Verifikation nicht Thema dieser Arbeit ist, gehen wir nicht weiter darauf ein, sondern beschreiben im folgenden Abschnitt, wie wir, gegeben eine

$$\begin{bmatrix} x=1 \\ y=1 \end{bmatrix} \xrightarrow{\alpha_1} \begin{bmatrix} x=2 \\ y=1 \end{bmatrix} \xrightarrow{\alpha_2} \begin{bmatrix} x=2 \\ y=2 \end{bmatrix}$$

$$\begin{bmatrix} x=1 \\ y=1 \end{bmatrix} \xrightarrow{\alpha_2} \begin{bmatrix} x=1 \\ y=2 \end{bmatrix} \xrightarrow{\alpha_1} \begin{bmatrix} x=2 \\ y=2 \end{bmatrix}$$

$$\begin{bmatrix} x=1 \\ y=1 \end{bmatrix} \xrightarrow{\alpha_3} \begin{bmatrix} x=2 \\ y=2 \end{bmatrix}$$

**Abbildung 3.8:** Alle möglichen schrittbasierten Interleavings von Ablauf $\sigma_3$.

umgebungsinvariante Spezifikation in Normalform, einen Graphen konstruieren können, der alle Interleavings des spezifizierten Systemes enthält.

## 3.4 Transitionssysteme für Spezifikationen

Aufbauend auf dem beschriebenen Zustands- und Aktionsbegriff kann nun ein Transitionssystem konstruiert werden, ein gerichteter Graph, dessen Knoten Zuständen und dessen Kanten Aktionen entsprechen. Dieses Transitionssystem bildet dann die Grundlage für die computergestützte Verifikation.

**Definition 28 (Transitionssystem)**
*Seien $\mathcal{V}$ eine endliche Variablenmenge und $\mathcal{D}$ eine Menge von Werten. Ein* Transitionssystem *$\mathcal{TS}$ über $\mathcal{V}$ und $\mathcal{D}$ ist ein 5-Tupel $\mathcal{TS} = (S, S_0, Act, R, L)$:*

- *$S$ ist eine Zustandsmenge.*
- *$S_0 \subseteq S$ ist eine Menge der Anfangszustände.*
- *$Act$ ist eine endliche Menge von Aktionen.*
- *$R \subseteq S \times Act \times S$ ist eine Zustandsübergangsrelation, wobei jeder Zustandsübergang mit einer Aktion beschriftet ist. Für $(s, \alpha, s') \in R$ schreiben wir auch $s \xrightarrow{\alpha} s'$.*
- *$L : S \to (\mathcal{V} \to \mathcal{D})$ ist eine Funktion, die jeden Zustand mit einer Belegung der Variablen beschriftet.*

⌟

Wir zeigen nun, wie wir zu einer TLDA-Spezifikation $\Phi$ das Transitionssystem $\mathcal{TS}$ über $V(\Phi)$ und $\mathcal{V}al$ konstruieren können:

**Definition 29 (Konstruktion eines Transitionssystemes)**
*Sei $\Phi \triangleq Init \wedge \Box Next \wedge Liveness$ eine Spezifikation in Normalform. Zu $\Phi$ kann das Transitionssystem $\mathcal{TS}_\Phi = (S, S_0, Act, R, L)$ über $V(\Phi)$ und $\mathcal{V}al$ wie folgt konstruiert werden:*

*Vorbereitung: Die Schrittformel Next wird in disjunktive Normalform umgeformt. Im Folgenden habe Next also die Form* $\alpha_0 \vee \alpha_1 \vee \ldots \vee \alpha_n$. *Die Menge der Aktionen Act bestehe aus den Klauseln von Next, d.h.* $Act \triangleq \{\alpha_0, \alpha_1, \ldots, \alpha_n\}$. *Initial seien die Mengen* $S$, $S_0$ *und* $R$ *leer.* $L$ *sei initial die leere Funktion.*

*Das Transitionsystem* $\mathcal{TS}_\Phi$ *wird nun induktiv über den Schritten der Abläufe von* $\Phi$ *konstruiert. Sei* $\sigma = (H, T)$ *ein beliebiger Ablauf mit* $\sigma \models \Phi$.

*IA: Die Zustandsformel Init von* $\Phi$ *gilt im Anfangsschnitt* $C_0$ *des Ablaufes* $\sigma$. *Für* $C_0$ *wird ein neuer Zustand* $s_{C_0}$ *zur Zustandsmenge* $S$ *und zur Menge der Anfangszustände* $S_0$ *hinzugefügt und mit der Belegung der Systemvariablen im Schnitt* $C_0$ *beschriftet, d.h.* $L(s_{C_0}) \triangleq \{x \mapsto H_x(C_0(x)) \mid x \in V(\Phi)\}$.

*IS: Sei* $S_C = (C, C', T_C)$ *ein Schritt in* $\sigma$, *der durch Schritte vom Anfangsschnitt* $C_0$ *in* $\sigma$ *erreichbar ist. Nach Induktionsannahme existiert ein Zustand* $s_C \in S$ *mit der Beschriftung* $L(s_C) = \{x \mapsto H_x(C(x)) \mid x \in V(\Phi)\}$. *Für den Folgeschnitt* $C'$ *von* $C$ *wird ein neuer Zustand* $s_{C'}$ *zur Zustandsmenge* $S$ *hinzugefügt und mit* $L(s_{C'}) \triangleq \{x \mapsto H_x(C'(x)) \mid x \in V(\Phi)\}$ *beschriftet.*

*Wegen* $\sigma \models \Phi$ *gilt insbesondere* $\sigma \models \Box Next$, *d.h. die Schrittformel Next gilt in jedem Schritt von* $\sigma$. *Es gilt also insbesondere* $S_C \Vdash Next$, *also* $S_C \Vdash \alpha_0 \vee \alpha_1 \vee \ldots \vee \alpha_n$. *D.h.* $S_C$ *erfüllt eine Menge* $A \subseteq Act$ *von Klauseln von Next. Es wird für jede Klausel* $\alpha \in A$ *ein Zustandsübergang* $s_C \xrightarrow{\alpha} s_{C'}$ *hinzugefügt.*

*Dieser Vorgang wird für alle Abläufe von* $\Phi$ *durchgeführt.* ⌟

**Anmerkung:** Das es i.A. unendlich viele Abläufe $\sigma$ mit $\sigma \models \Phi$ gibt, hat $\mathcal{TS}_\Phi$ i.d.R. unendlich viele Zustände. Für widersprüchliche Spezifikationen ist $S = S_0 = R = \emptyset$, d.h. das Transitionssystem hat weder Zustände noch Zustandsübergänge.

Für umgebungsinvariante Spezifikationen $\Phi$ enthält $\mathcal{TS}_\Phi$ genau die möglichen Interleavings der Abläufe von $\Phi$. Somit ist jeder Ablauf durch ein Interleaving und jedes Interleaving durch ein Ablauf repräsentiert. Dieser Zusammenhang erlaubt uns nun, die bereits intuitiv beschriebene schrittbasierte TLDA-Interleavingsemantik zu formalisieren.

**Definition 30 (Schrittbasierte TLDA-Interleavingsemantik)**
*Seien* $\Phi$ *eine umgebungsinvariante Spezifikation und* $\mathcal{TS}_\Phi$ *ein Transitionssystem zu* $\Phi$.

- *Die* endliche Interleavingsemantik *einer umgebungsinvarianten Spezifikation* $\Phi$ *ist die Menge aller endlichen Zustandsübergangsfolgen* $s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \cdots s_{n-1} \xrightarrow{\alpha_n} s_n$ *in* $\mathcal{TS}_\Phi$.
- *Die* unendliche Interleavingsemantik *einer umgebungsinvarianten Spezifikation* $\Phi$ *ist die Menge aller unendlichen Zustandsübergangsfolgen* $s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \ldots$ *in* $\mathcal{TS}_\Phi$.

⌟

Wir können nun Systeme durch eine umgebungsinvariante Spezifikation $\Phi$ in Normalform modellieren und ihr Verhalten sowohl durch die Menge der Abläufe $\sigma \models \Phi$, als auch durch die Menge der Pfade im Transitionssystem $\mathcal{TS}_\Phi$ untersuchen. Dabei ist die Menge der Zustände in $\mathcal{TS}_\Phi$ jedoch unendlich, da für jeden Schnitt in jedem Ablauf ein Zustand hinzugefügt wurde. Wir haben Zustände jedoch als Wertebelegungen der Systemvariablen definiert, sodass Zustände in $\mathcal{TS}_\Phi$ mit gleicher Beschriftung $L$ zusammengefasst werden können.

**Definition 31 (Äquivalente Zustände)**
*Sei $\mathcal{TS}_\Phi$ ein Transitionssystem einer Spezifikation $\Phi$. Zwei Zustände $s, s' \in S$ sind genau dann* äquivalent*, wenn ihre Beschriftungen gleich sind: $s \sim s'$ gdw. $L(s) = L(s')$.* ⌟

Da die $\sim$-Relation reflexiv, symmetrisch und transitiv ist, handelt es sich um eine Äquivalenzrelation, mit der sich die Menge $S$ der Zustände und die Menge $S_0$ der Anfangszustände in Äquivalenzklassen partitionieren lassen.

**Definition 32 (Reduziertes Transitionssystem)**
*Sei $\mathcal{TS}_\Phi = (S, S_0, Act, R, L)$ ein Transitionssystem einer Spezifikation $\Phi$. Das 5-Tupel $\mathcal{TS}_\Phi/\sim = (S/\sim, S_0/\sim, Act, R/\sim, L)$ heißt* reduziertes Transitionssystem von $\Phi$ *genau dann, wenn*

1. *$S/\sim \subseteq S$ enthält aus jeder Äquivalenzklasse von $S$ genau einen Repräsentanten.*
2. *$S_0/\sim \subseteq S_0$ enthält aus jeder Äquivalenzklasse von $S_0$ genau einen Repräsentanten.*
3. *$R/\sim \triangleq \{(s_1, \alpha, s_2) \mid s_1, s_2 \in S/\sim, (s_1', \alpha, s_2') \in R, s_1 \sim s_1', s_2 \sim s_2'\}$ ist eine Zustandsübergangsrelation.*

⌟

Das reduzierte Transitionssystem $\mathcal{TS}_\Phi/\sim$ hat für eine Spezifikation mit endlichen Wertebereichen der Systemvariablen $V(\Phi)$ stets endlich viele Zustände. Diese Beobachtung spielt beim expliziten Modelchecking, bei dem Eigenschaften durch Tiefensuche im reduzierten Transitionssystem überprüft werden, eine wichtige Rolle und wird im folgenden Kapitel aufgegriffen.

# 4 Implementierung der Semantik

Nachdem wir in den letzten beiden Kapiteln die theoretischen Grundlagen der vorgeschlagenen schrittbasieren Interleavingsemantik für TLDA beschrieben haben, wollen wir nun zeigen, wie letztere prototypisch im Werkzeug TLDC implementiert wurde. Der Name TLDC spielt dabei an den TLA-Modelchecker TLC [YML99] an.

In Abschnitt 3.4 zeigten wir, wie zu einer gegebenen umgebungsinvarianten Spezifikation $\Phi$ das reduzierte Transitionssystem $\mathcal{TS}_\Phi/\sim$ konstruiert werden kann. Dazu betrachteten wir alle Schritte jedes Ablaufes von $\Phi$ und fügten Zustände und Zustandsübergänge hinzu. Dieser Konstruktionsansatz erfordert jedoch die explizite Kenntnis der (unendlich vielen) Abläufe von $\Phi$ und kann daher nicht implementiert werden. Daher arbeiteten wir einen neuen Konstruktionsansatz aus, der die notwendigen Informationen über Zustände und Zustandsübergänge syntaktisch aus der Spezifikation ableitet.

Wir beschreiben im Folgenden zunächst TLDA$^+$, eine Spezifikationsprache für TLDA. Anschließend wird in Abschnitt 4.2 der schematische Aufbau des Prototyps TLDC skizziert und in den Abschnitten 4.3 und 4.4 detailliert erläutert. In Abschnitt 4.5 charakterisieren wir die Klasse von Spezifikationen, für die TLDC Transitionssysteme konstruieren kann. Zuletzt beschreiben wir in Abschnitt 4.6 die Probleme, die bei der automatischen Konstruktion der reduzierten Transitionssysteme auftreten.

## 4.1 Spezifikationssprache TLDA$^+$

Um TLDA-Spezifikationen maschinell verarbeiten zu können, müssen sie zunächst in eine entsprechende Form gebracht werden. Dazu wurde die Spezifikationssprache TLDA$^+$ entwickelt, die auf der in Abschnitt 2.3 eingeführten Syntax basiert, jedoch um Elemente erweitert wurde, die für den automatischen Aufbau eines reduzierten Transitionssystemes notwendig sind:

- Eine TLDA$^+$-Spezifikation beginnt stets mit einer Präambel, in der alle Systemvariablen mit ihren (endlichen) Wertebereichen aufgelistet sind. Dadurch wird sichergestellt, dass das zu konstruierende reduzierte Transitionssystem stets einen endlichen Zustandsraum hat und somit die Grundvoraussetzung für explizites Modelchecking erfüllt ist. Im Folgenden bezeichnen wir den Wertebereich einer Systemvariablen $x \in V(\Phi)$ mit $range(x)$.

| ASCII | Symbol | Beschreibung | Beispiel |
|---|---|---|---|
| ’ | $'$ | Symbol für gestrichene Variablen | x’ |
| ~ | $\sim$ | Symbol für $\sim$-Variablen | x~ |
| = | $=$ | gleich | x = 1 |
| > | $>$ | größer | x > 1 |
| < | $<$ | kleiner | x < 1 |
| >= | $\geq$ | größer oder gleich | x >= 1 |
| <= | $\leq$ | kleiner oder gleich | x <= 1 |
| != | $\neq$ | ungleich | x != 1 |
| \in | $\in$ | Element von | x \in {1,2} |
| + | $+$ | Addition | x + 1 |
| - | $-$ | Subtraktion | x - 1 |
| * | $\cdot$ | Multiplikation | x * 1 |
| / | $\div$ | Division | x / 1 |
| % | mod | Modulo | x % 1 |
| TRUE | TRUE | Tautologie | x~ = TRUE |
| FALSE | FALSE | Kontradiktion | x~ = FALSE |
| ! | $\neg$ | Negation | ! x~ |
| /\ | $\wedge$ | Konjunktion | x~ /\ y~ |
| \/ | $\vee$ | Disjunktion | x~ \/ y~ |
| => | $\Rightarrow$ | Implikation | x~ => y~ |
| <=> | $\Leftrightarrow$ | Äquivalenz | x~ <=> y~ |
| [] | $\Box$ | Always-Operator | [] x~ |
| ( | ( | runde öffnende Klammer | (1 + x) * 2 |
| ) | ) | runde schließende Klammer | (1 + x) * 2 |
| { | { | öffnende Mengenklammer | {1,2} |
| } | } | schließende Mengenklammer | {1,2} |
| , | , | Komma | {1,2,3,4} |
| .. | .. | Wertebereich | {0..10} |
| == | $\triangleq$ | definiert als | X == 1 |
| VARIABLES | VARIABLES | Schlüsselwort: Variablendefinition | |
| CONSTANTS | CONSTANTS | Schlüsselwort: Konstantendefinition | |

**Abbildung 4.1:** TLDA$^+$-Lexik.

- Innerhalb der Präambel muss außerdem den rigiden Variablen der Spezifikation neben einem Wertebereich auch ein Initialwert zugewiesen werden. Diese Initialisierung entspricht der Abbildung $\beta : \mathcal{V}ar_{rigid} \rightarrow \mathcal{V}al$, also einer Belegung für $\mathcal{V}ar_{rigid}$.
- TLDA$^+$ liegt die übliche Interpretation der Relationssymbole $=$, $\neq$, $<$, $>$, $\leq$ und $\geq$ auf den ganzen Zahlen, der Relationssymbole $=$ und $\neq$ auf Zeichenketten sowie der Funktionssymbole $+$, $-$, $\cdot$, $\div$ und mod auf den ganzen Zahlen zugrunde.

Abbildung 4.1 zeigt die Lexik von TLDA$^+$, die stark an die Spezifikationssprache TLA$^+$ [Lam02] angelehnt ist, jedoch viele der TLA$^+$-Konzepte (Mengenvariablen, Arrays, Modularisierung) nicht unterstützt. Das folgende Beispiel zeigt, wie TLDA-Spezifikationen in TLDA$^+$ ausgedrückt werden.

**Beispiel 8.** In Abschnitt 2.5 gaben wir eine Spezifikation für den Ablauf $\sigma_1$ aus Abbildung 2.1 an:

$$\begin{aligned}
\Phi &\triangleq \mathit{Init} \wedge \Box \mathit{Next} \\
\mathit{Init} &\triangleq x = 1 \wedge y = 1 \\
\mathit{Next} &\triangleq \big(\widetilde{x} \Rightarrow \big((x' = x + 1 \wedge \neg\widetilde{x\,y}) \vee (x' = x \cdot 2 \wedge \widetilde{x y})\big)\big) \wedge \\
&\qquad \big(\widetilde{y} \Rightarrow \big((y' = y + 1 \wedge \neg\widetilde{x\,y}) \vee (y' = y \wedge \widetilde{x y})\big)\big)
\end{aligned}$$

Für die Wertebereiche $\mathit{range}(x) \triangleq \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ und $\mathit{range}(y) \triangleq \{1, 2\}$ kann diese Spezifikation bspw. wie folgt in TLDA$^+$ angegeben werden:

```
VARIABLES
  x \in {1..10},
  y \in {1,2}

x=1 /\ y=1
/\ []( x~ => ((x'=x+1 /\ !{x,y}~) \/ (x'=x*2 /\ {x,y}~)) )
/\ []( y~ => ((y'=y+1 /\ !{x,y}~) \/ (y'=y   /\ {x,y}~)) )
```

⌟

## 4.2 Schematischer Aufbau von TLDC

Der Prototyp TLDC soll folgendes Problem lösen:

> Gegeben: umgebungsinvariante TLDA$^+$-Spezifikation $\Phi$ in Normalform,
> Gesucht: reduziertes Transitionssystem $\mathcal{TS}_\Phi/\!\sim$.

Allerdings kann die Konstruktionsvorschrift aus Abschnitt 3.4 (Def. 29) nicht angewendet werden, da dazu die Kenntnis der unendlich vielen Abläufe von $\Phi$ erforderlich wäre und somit nicht implementiert werden kann. Stattdessen müssen wir die Interleavings — und damit das Transitionssystem — berechnen, anstatt sie durch Beobachtung der Schnitte und Schritte in Abläufen zu konstruieren.

Um ein Interleaving zu berechnen, müssen wir, ausgehend von einem Anfangszustand, mithilfe der Aktionen Nachfolgezustände berechnen. Sind alle Anfangszustände und alle Aktionen bekannt, können so alle Interleavings — und somit das Transitionssystem — konstruiert werden.

In jedem Ablauf, der die gegebene Spezifikation $\Phi \triangleq \mathit{Init} \wedge \Box \mathit{Next}$ erfüllt, erfüllt der Anfangsschnitt die Zustandsformel *Init*. Demnach beschreibt *Init* alle möglichen Anfangsschnitte der Abläufe von $\Phi$, sodass die Anfangszustände aus *Init* syntaktisch abgeleitet werden können. Des Weiteren haben wir die Aktionen als Klauseln der disjunktiven Normalform von *Next* definiert, sodass wir die Aktionen ebenfalls aus der gegebenen Spezifikation ableiten können.

Abbildung 4.2 zeigt die einzelnen Schritte, die zur Konstruktion des reduzierten Transitionssystemes $\mathcal{TS}_\Phi/\sim$ notwendig sind. Diese Schritte lassen sich in vier Phasen (Vorbereitung, Umformung, Informationsgewinnung, Konstruktion) aufteilen, die jeweils aufeinander aufbauen. Dabei lösen die ersten drei Phasen das beschriebene Problem, aus $\Phi$ auf syntaktischem Wege Informationen über Aktionen und Anfangszustände abzuleiten.

| | |
|---|---|
| 1. Einlesen der Spezifikation<br>2. Überprüfung auf Fehler (z.B. Lexik, Syntax) | Vorbereitung |
| 3. Umformung in disjunktive Normalform<br>4. Entfernen nicht erfüllbarer Klauseln | Umformung |
| 5. Analyse von *Init*<br>6. Analyse von *Next* | Informationsgewinnung |
| 7. Berechnung der Anfangszustände<br>8. Berechnung der Nachfolgezustände | Konstruktion von $\mathcal{TS}_\Phi/\sim$ |

**Abbildung 4.2:** Schematischer Ablauf der Konstruktion des reduzierten Transitionssystemes.

Dieses Problem entspricht dem klassischen Szenario des Übersetzerbaus: ein Programm einer Eingabesprache (hier: die TLDA$^+$-Spezifikation) muss eingelesen, überprüft und umgeformt (hier: in disjunktive Normalform) werden. Das Ergebnis (hier: Informationen über die Anfangszustände und Aktionen) wird dann in eine Zielsprache überführt (hier: ein Programm, das das reduzierte Transitionssystem konstruiert).

Wir betrachten diese Probleme losgelöst voneinander, sodass sich die folgenden zwei Teilprobleme ergeben:

> Gegeben: umgebungsinvariante TLDA$^+$-Spezifikation $\Phi$ in Normalform,
> Gesucht: Informationen über die Menge der Anfangszustände und Aktionen.
>
> Gegeben: Informationen über die Menge der Anfangszustände und Aktionen,
> Gesucht: reduziertes Transitionssystem $\mathcal{TS}_\Phi/\sim$.

Diese Zweiteilung in Übersetzung (die Phasen der Vorbereitung, Umformung und Informationsgewinnung) und Berechnung des reduzierten Transitionssystemes (die letzte Phase) betrachten wir in den nächsten beiden Abschnitten, in denen wir auch die verwendeten Algorithmen und Werkzeuge vorstellen, genauer.

## 4.3 Analyse der Eingabespezifikation

Der Prototyp TLDC, der die Übersetzung der TLDA$^+$-Spezifikation $\Phi \triangleq \mathit{Init} \wedge \Box \mathit{Next}$ in ein Programm realisiert, wurde in C++ [Str97] implementiert. Diese Programmiersprache ermöglicht nicht nur die Integration verbreiteter Werkzeuge des Übersetzerbaues, sondern stellt auch die notwendige Leistungsfähigkeit (Geschwindigkeit und Speicherverwaltung) zur Verfügung. Des weiteren ermöglicht C++ einen objektorientierten, modularen Aufbau sowohl des Übersetzers TLDC, als auch des erzeugten Programmes zum Aufbau

des reduzierten Transitionssystemes, was Wartung, Erweiterung und Dokumentation erleichtert.

Für die Übersetzung sind folgende Schritte notwendig (vgl. Abb. 4.2):

**Vorbereitung**

Um eine Eingabespezifikation einzulesen und auf Fehler zu überprüfen, muss sie zunächst lexikalisch analysiert werden, d.h. es muss überprüft werden, ob die Eingabe der Lexik von TLDA$^+$ (s. Abbildung 4.1) genügt, d.h. nur erlaubte Symbole enthält. Dazu verwenden wir das Werkzeug Flex [Pax95], das die notwendigen Funktionen zur Verfügung stellt und die Eingabespezifikation in sog. Token aufteilt. Ein Token kann dabei ein Schlüsselwort (z.B. VARIABLES), ein Bezeichner, ein Funktions- oder Relationssymbol oder ein sonstiges Symbol (Klammern, etc.) sein.

Anschließend wird überprüft, ob diese Tokensequenz einem Wort der Sprache TLDA$^+$ entspricht, d.h. ob es aus dem Startsymbol der TLDA$^+$-Grammatik ableitbar ist.[1] Dieser Schritt wird syntaktische Analyse genannt. Das Werkzeug Bison [DS04] bietet diese Funktionalität und baut einen sog. Syntaxbaum auf. Die Wurzel dieses Baumes ist das Startsymbol der TLDA$^+$-Grammatik, die inneren Knoten sind Nichtterminalsymbole, und die Blätter sind die Token (Terminalsymbole) der Eingabespezifikation.

Diese Datenstruktur wird dann verwendet, um Informationen über die Systemvariablen zu extrahieren und damit die Spezifikation auf semantische Fehler hin zu überprüfen. Semantische Fehler sind bspw. nicht in der Präambel angegebene Variablensymbole, Überläufe der Wertbereiche der Variablen oder Typkonflikte bei Funktionsaufrufen bzw. in Relationen.

**Umformung**

Im nächsten Schritt soll nun die Schrittformel *Next* in disjunktive Normalform überführt werden. Dazu verwenden wir das Werkzeug Kimwitu++ [LP02], das zunächst von allen Knoten im Syntaxbaum, die nur der syntaktischen Analyse dienen, für die Umformung der Spezifikation jedoch keine Rolle spielen (z.B. Schlüsselworte oder Klammern), abstrahiert und einen abstrakten Syntaxbaum erzeugt.

Die Überführung der Spezifikation in disjunktive Normalform wird nun durch Umformung einzelner Teilbäume des abstrakten Syntaxbaumes realisiert. Dazu werden Ersetzungsregeln formuliert, mit denen sowohl die Umformung in disjunktive Normalform, als auch die Überprüfung der Klauseln auf Widersprüchlichkeit realisiert werden können.

Viele der Klauseln der disjunktiven Normalform von *Next* sind in sich widersprüchlich, d.h. können von keinem Schritt der Abläufe des spezifizierten Systemes erfüllt werden und scheiden somit als Aktionen aus. Widersprüchliche Klauseln bedeuten jedoch keine Entwurfsfehler, sondern entstehen meist bei Umformung in disjunktive Normalform. Die

[1] Wir gehen an dieser Stelle nicht auf die Grammatik ein.

Überprüfung einer Klausel auf Widerspruchsfreiheit ist jedoch nicht trivial, da oft nur die $\sim$-Variablen den Grund für Widersprüche darstellen, solche Widersprüche jedoch meist nicht rein syntaktisch zu erkennen sind.

Mit den Ableitungsregeln aus Korollar 2.1 müssen zunächst Informationen über $\sim$-Variablen gewonnen werden. Dabei muss die 2. Regel (Transitivität) meist mehrfach hintereinander ausgeführt werden, um die transitive Hülle der $\sim$-Variablen zu berechnen. In Abschnitt 4.6 diskutieren wir die Probleme, die sich durch die erhöhte Rechenzeit dieser Überprüfung ergeben.

#### Informationsgewinnung

Wegen des speziellen Aufbaues der Eingabespezifikation (bestehend aus Präambel, Zustandsformel *Init* und Schrittformel *Next*), ist es nun leicht möglich, die notwendigen Informationen über die Anfangszustände (aus *Init*) und die Aktionen (aus *Next*) abzuleiten. Dazu wird der (nun umgeformte) abstrakte Syntaxbaum traversiert und C++-Code für ein Programm erzeugt, das im letzten Schritt das Transitionssystem aufbaut. Auf dieses Programm, d.h. auf den Algorithmus, der das Transitionssystem letztendlich erzeugt, gehen wir im folgenden Abschnitt ein.

## 4.4 Berechnung des reduzierten Transitionssystemes

Anhand der Informationen über die Anfangszustände und Aktionen können wir nun das reduzierte Transitionssystem konstruieren. Dazu müssen wir zunächst die Anfangszustände explizit aus den Informationen berechnen. Ausgehend von dieser Zustandsmenge können wir dann mit den Aktionen die Nachfolgezustände und damit das gesamte (reduzierte) Transitionssystem konstruieren.

Bei der Implementierung trennen wir nicht mehr zwischen einem Zustand $s$ und seiner Beschriftung $L(s)$, sondern beschreiben einen Zustand mit einem Wertetupel $s = (q_1, \ldots, q_n)$, das jeder Systemvariablen $x_i$ aus $V(\Phi) = \{x_1, \ldots, x_n\}$ einen Wert $q_i$ aus dem Wertebereich $\mathit{range}(x_i)$ zuordnet. Dabei schreiben wir kurz $s[i]$ für den $i$-ten Wert des Wertetupels $s$.

Im Folgenden sei $\Phi \triangleq \mathit{Init} \wedge \Box \mathit{Next}$ eine umgebungsinvariante TLDA$^+$-Spezifikation in Normalform. Des weiteren sei die Menge der Systemvariablen $V(\Phi) = \{x_1, \ldots, x_n\}$.

#### Berechnung der Anfangszustände

Im Übersetzungsschritt wurden aus der Zustandsformel *Init* Informationen über die Anfangszustände ermittelt. Bei *Init* handelt es sich um eine Zustandsformel, die die Anfangsschnitte der Abläufe der gegebenen Spezifikation beschreibt. Wir betrachten dabei nur Zustandsformeln, in denen keine Funktionen und keine logischen Verknüpfungen außer Konjunktionen vorkommen (s. Abschnitt 4.5).

Somit kann jedes Wertetupel, das die Systemvariablen mit Werten belegt, sodass alle Konjunktionsglieder von *Init* erfüllt sind, als ein Anfangszustand angesehen werden:

$$S_0 \triangleq \left\{ (q_1, \ldots, q_n) \;\middle|\; \mathit{Init} \wedge \bigwedge_{i=1}^{n} (q_i \in \mathit{range}(x_i)) \wedge \bigwedge_{i=1}^{n} (x_i = q_i) \right\}$$

Für Systemvariablen $x_i \in V(\Phi)$, die in *Next*, nicht aber in *Init* vorkommen, ist dabei $q_i$ ein beliebiger Wert aus $\mathit{range}(x_i)$. Falls *Init* widersprüchlich ist, gilt $S_0 = \emptyset$, und das reduzierte Transitionssystem hat weder Zustände noch Zustandsübergänge.

**Berechnung der Nachfolgezustände**

Sei nun $s$ ein Zustand des reduzierten Transitionssystemes von $\Phi$ und $\alpha$ eine Aktion.

In Abschnitt 3.4 (Def. 29) betrachteten wir die einzelnen Schritte in Abläufen der Eingabespezifikation und nahmen einen Zustandsübergang $s \xrightarrow{\alpha} s'$ genau dann in das Transitionssystem auf, wenn ein Schritt $S_{C_s} = (C_s, C'_s, T_{C_s})$ existiert mit $S_{C_s} \models \alpha$. Wir konstruieren nun die Nachfolgezustände von $s$ bzgl. $\alpha$ nicht durch Beobachtung dieses Schrittes $S_{C_s}$, sondern anhand des Zustandes $s$ und der Aktion $\alpha$ selbst.

Der Zustand $s$ im reduzierten Transitionssystem (vgl. Def. 32) repräsentiert alle Schnitte $C_s$ in den Abläufen von $\Phi$ mit $H_x(C_s(x)) = L(s)(x)$ für alle Systemvariablen $x \in V(\Phi)$. Ein Zustand $s'$ ist genau dann ein Nachfolgezustand von $s$ bzgl. $\alpha$, wenn in einem Ablauf von $\Phi$ ein Schritt $S_{C_s} = (C_s, C'_s, T_{C_s})$ existiert mit $S_{C_s} \models \alpha$. Wir können diesen Schritt, bzw. den Nachfolgezustand von $s$ bzgl. $\alpha$, der den Nachfolgeschnitt von $C_s$ repräsentiert, konstruieren, indem wir die Gültigkeit von $S_{C_s} \models \alpha$ betrachten.

Die Gültigkeit von $S_{C_s} \models \alpha$ hängt nach Def. 19 von der Belegung von $\mathcal{V}ar_{\mathit{rigid}}$, einer Interpretation von $(\mathcal{F}, \mathcal{R})$, der Transitionenmenge $T_{C_s}$ sowie den Historienwerten $H_x(C_s(x))$ und $H_x(C'_s(x))$ für alle $x \in V(\Phi)$ ab (die Klausel $\alpha$ ist umgebungsinvariant). Dabei ist die Belegung von $\mathcal{V}ar_{\mathit{rigid}}$ in der Präambel der Eingabespezifikation $\Phi$ angegeben und die Interpretation von $(\mathcal{F}, \mathcal{R})$ in TLDA$^+$ eindeutig festgelegt. Des Weiteren beschreibt der Zustand $s$ den Schnitt $C_s$ und somit die Historienwerte der Systemvariablen. Die Transitionenmenge $T_{C_s}$ kann nur aus den Abläufen hergeleitet werden, jedoch können wir Informationen über die Involviertheit der Systemvariablen meist aus der Schrittformel *Next*, und somit aus der Aktion $\alpha$ selbst herleiten (vgl. auch Abschnitt 4.6).

Zuletzt hängt die Gültigkeit von $S_{C_s} \models\!\!\models \alpha$ somit vom Nachfolgeschnitt $C'_s$, bzw. von den Historienwerten der Systemvariablen $H_x(C'_s(x))$ für alle $x \in V(\Phi)$, ab. Wir bilden nun, ähnlich zur Konstruktion der Anfangszustände, die Menge der Nachfolgezustände von $s$ bzgl. der Aktion $\alpha$:

$$\mathit{Succ}(s, \alpha) \triangleq \left\{ (q_1, \ldots, q_n) \;\middle|\; \alpha \wedge \bigwedge_{i=1}^{n} (q_i \in \mathit{range}(x_i)) \wedge \bigwedge_{i=1}^{n} (x_i = s[i]) \wedge \bigwedge_{i=1}^{n} \left(x'_i = q_i\right) \right\}$$

Falls $x_i$ nicht in $\alpha$ vorkommt, ist $q_i$ wieder ein beliebiger Wert aus $\mathit{range}(x_i)$. Nach Korollar 2.1 gilt außerdem, dass für jede Variable $x_i \in V(\Phi)$, für die $\neg\widetilde{x_i}$ in $\alpha$ vorkommt,

$q_i = s[i]$ ist, also dass der Wert der Varialblen $x_i$ im Nachfolgezustand unverändert bleibt. Falls die Schrittformel $\alpha$ widersprüchlich ist, existieren keine Nachfolgezustände von $s$ bzgl. $\alpha$, d.h. $Succ(s, \alpha) = \emptyset$.

Das reduzierte Transitionssystem wird nun konstruiert, indem, ausgehend von den Anfangszuständen, alle Nachfolgezustände bzgl. aller Aktionen berechnet werden.

## 4.5 Einschränkungen von TLDC

Während wir in Abschnitt 3.4 ein Verfahren angeben haben, das für beliebige umgebungsinvariante Spezifikationen in Normalform $\Phi$ das reduzierte Transitionssystem $\mathcal{TS}_\Phi/\sim$ aufbaut, hat die derzeitige Implementierung folgende Einschränkungen, die wir an dieser Stelle zusammenfassen:

- In der Zustandsformel *Init* dürfen nur Konjunktionen als logische Verknüpfungen und keine Funktionssymbole vorkommen.
- In der gesamten Spezifikation sind keine $\exists$- oder $\widetilde{\exists}$-Quantoren erlaubt.
- Auf der linken Seite einer Relation darf nur ein einziges Variablensymbol oder Konstantensymbol vorkommen.
- Symbole für Variablen aus $\mathcal{V}ar'$ dürfen nicht auf der rechten Seite einer Relation vorkommen.
- Spezifikationen werden nicht auf Umgebungsinvarianz überprüft.
- Die Wertebereiche der Systemvariablen müssen endlich sein.

Wenn in weiterer Arbeit Lösungen für diese Einschränkungen entwickelt werden, können diese wegen des modularen Aufbaues von TLDC (vgl. Abb. 4.2) leicht eingearbeitet werden. Offene Probleme bei der Konstruktion des reduzierten Transitionssystemes diskutieren wir im folgenden Abschnitt.

## 4.6 Probleme bei der Konstruktion

Bei der Konstruktion des reduzierten Transitionssystemes treten mehrere Probleme auf, die wir an dieser Stelle diskutieren wollen.

### Größe der disjunktiven Normalform

Die Umformung einer Formel in disjunktive Normalform ist im ungünstigsten Falle (falls die Formel in konjunktiver Normalform vorlag) sowohl in der Zeit als auch im Raum (Speicherplatz) exponentiell. Somit kann die Umformungsphase (vgl. Abb. 4.2) sehr lange dauern, bzw. wegen Speicherüberlaufes nicht beendet werden.

Beispielsweise wird in [Ale04] ein Konsumenten-/Produzentensystem spezifiziert, bei der die disjunktive Normalform der Schrittformel *Next* aus 64827 Klauseln besteht, von denen jedoch ein Großteil in sich widersprüchlich sind. Der resultierende abstrakte Syntaxbaum benötigt dabei ca. 100 Megabytes Speicherplatz.

### Widersprüchliche Klauseln

In Abschnitt 4.3 beschrieben wir, wie in der Umformungsphase anhand der $\sim$-Variablen widersprüchliche Klauseln erkannt und verworfen werden können. Der dazu notwendige Berechnungsaufwand bedeutet jedoch bei einer exponentiellen Klauselanzahl, dass die Berechnungszeit der Informationsgewinnungsphase (vgl. Abb. 4.2) inakzeptabel hoch ist.

Andererseits ist diese Berechnung notwendig, da nicht zu viele widersprüchliche Aktionen „übersehen“ werden, da für jede Aktion Programmcode erzeugt und übersetzt werden muss (Speicherplatzproblem). Außerdem wird bei der Konstruktion des reduzierten Transitionssystemes in jedem Zustand für jede Aktion überprüft werden, ob mit ihr ein Nachfolgezustand erreicht werden kann (Laufzeitproblem).

Dieses Problem wird am Beispiel des Konsumenten-/Produzentensystemes aus [Ale04] deutlich: Von den 64.827 Klauseln sind 64.699 — also über 99 Prozent — widersprüchlich. Verzichtet man auf die rechenintensive Überprüfung auf widersprüchliche Klauseln, ergibt sich ein Quelltext von 90 Megabytes, der nicht übersetzt werden kann.

### Anzahl der $\sim$-Variablen

Die Anzahl der möglichen $\sim$-Variablen wächst exponentiell mit der Anzahl der Systemvariablen. Bei der Überprüfung von Aktionen auf Widersprüchlichkeit sowie beim späteren Modelchecking ist es jedoch notwendig, dass Informationen über Systemvariablen in jedem Zustand des reduzierten Transitionssystemes gespeichert werden müssen. Allerdings ist die Anzahl der Zustände des reduzierten Transitionssystemes wegen der in Abschnitt 3.4 angesprochenen Zustandsraumexplosion selbst sehr groß.

Die Entwicklung leistungsstärkerer Algorithmen bzw. Heuristiken zur Lösung der angesprochenen Probleme ist Gegenstand weiterer Arbeit (vgl. Kapitel 6).

# 5 Beispiele

In diesem Kapitel geben wir zwei Beispiele an, an denen die Eingabesprache TLDA$^+$, die Konstruktion des reduzierten Transitionssystemes und der Zusammenhang zwischen Abläufen und Interleavings illustriert werden soll.

Die abgebildeten Transitionssysteme wurden mit dem Prototyp TLDC anhand der angegebenen TLDA$^+$-Spezifikationen erzeugt und mit dem Werkzeug DOT [GN00] gezeichnet.

## 5.1 Einleitendes Beispiel

Wir gaben in Abschnitt 4.1 eine TLDA$^+$-Spezifikation an, die unter anderem den Ablauf $\sigma_1$ (vgl. Abb. 2.1) beschreibt.

```
VARIABLES
  x \in {1..10},
  y \in {1,2}

x=1 /\ y=1
/\ []( x~ => ((x'=x+1 /\ !{x,y}~) \/ (x'=x*2 /\ {x,y}~)) )
/\ []( y~ => ((y'=y+1 /\ !{x,y}~) \/ (y'=y   /\ {x,y}~)) )
```

**Abbildung 5.1:** TLDA$^+$-Spezifikation des einleitenden Beispieles.

Wir wollen nun anhand dieser Spezifikation (s. Abb. 5.1) die einzelnen Schritte der Konstruktion des reduzierten Transitionssystemes erläutern.

Nach dem Einlesen der Spezifikation wird die Schrittformel *Next* in disjunktive Normalform überführt. Anschließend werden für jede Klausel die Regeln aus Korollar 2.1 angewendet, um mehr Informationen über die $\sim$-Variablen zu gewinnen (vgl. Abb. 5.2). Von den neun Klauseln sind einige widersprüchlich und werden daher entfernt (vgl. Abb. 5.3). Außerdem fassen wir mehrfach enthaltene Konjunktionsterme zusammen. Die übrigen fünf Klauseln entsprechen nun den Aktionen (vgl. Abb. 5.4), mit denen wir das reduzierte Transitionssystem aufbauen können.

Neben den Aktionen müssen Informationen über die Anfangszustände aus *Init* abgeleitet werden — offensichtlich ist $s_0 \triangleq \left[\begin{smallmatrix} x=1 \\ y=1 \end{smallmatrix}\right]$ der einzige Anfangszustand.

Nun kann das reduzierte Transitionssystem konstruiert werden, indem, ausgehend vom Anfangszustand $s_0$, die Nachfolgezustände bzgl. der jeweiligen Aktionen berechnet werden. Dabei gilt:

$$
\begin{aligned}
\mathit{Next} \triangleq\ & \neg\widetilde{x} \wedge \neg\widetilde{y} \wedge \neg\widetilde{xy} \wedge (x' = x) \wedge (y' = y) \\
& \vee \neg\widetilde{x} \wedge \widetilde{y} \wedge \neg\widetilde{xy} \wedge (x' = x) \wedge (y' = y) + 1) \\
& \vee \neg\widetilde{x} \wedge \widetilde{y} \wedge \widetilde{xy} \wedge (x' = x) \wedge (y' = y) \\
& \vee \widetilde{x} \wedge \neg\widetilde{y} \wedge \neg\widetilde{xy} \wedge (x' = x) + 1) \wedge (y' = y) \\
& \vee \widetilde{x} \wedge \neg\widetilde{xy} \wedge \neg\widetilde{xy} \wedge \widetilde{y} \wedge (x' = x + 1) \wedge (y' = y + 1) \\
& \vee \widetilde{x} \wedge \widetilde{y} \wedge \neg\widetilde{xy} \wedge \widetilde{xy} \wedge (x' = x + 1) \wedge (y' = y) \\
& \vee \widetilde{x} \wedge \neg\widetilde{y} \wedge \widetilde{xy} \wedge (x' = x \cdot 2) \wedge (y' = y) \\
& \vee \widetilde{x} \wedge \widetilde{y} \wedge \widetilde{xy} \wedge \neg\widetilde{xy} \wedge (x' = x \cdot 2) \wedge (y' = y + 1) \\
& \vee \widetilde{x} \wedge \widetilde{y} \wedge \widetilde{xy} \wedge \widetilde{xy} \wedge (x' = x \cdot 2) \wedge (y' = y)
\end{aligned}
$$

**Abbildung 5.2:** Von TLDC berechnete disjunktive Normalform von *Next*.

$$
\begin{aligned}
\mathit{Next} \triangleq\ & \neg\widetilde{x} \wedge \neg\widetilde{y} \wedge \neg\widetilde{xy} \wedge (x' = x) \wedge (y' = y) \\
& \vee \neg\widetilde{x} \wedge \widetilde{y} \wedge \neg\widetilde{xy} \wedge (x' = x) \wedge (y' = y + 1) \\
& \vee \widetilde{x} \wedge \neg\widetilde{y} \wedge \neg\widetilde{xy} \wedge (x' = x + 1) \wedge (y' = y) \\
& \vee \widetilde{x} \wedge \widetilde{y} \wedge \neg\widetilde{xy} \wedge (x' = x + 1) \wedge (y' = y + 1) \\
& \vee \widetilde{x} \wedge \widetilde{y} \wedge \widetilde{xy} \wedge (x' = x \cdot 2) \wedge (y' = y)
\end{aligned}
$$

**Abbildung 5.3:** Disjunktive Normalform von *Next* ohne widersprüchliche Klauseln.

$$
\begin{aligned}
\alpha_0 &\triangleq \neg\widetilde{x} \wedge \neg\widetilde{y} \wedge \neg\widetilde{xy} \wedge (x' = x) \wedge (y' = y) \\
\alpha_1 &\triangleq \neg\widetilde{x} \wedge \widetilde{y} \wedge \neg\widetilde{xy} \wedge (x' = x) \wedge (y' = y + 1) \\
\alpha_2 &\triangleq \widetilde{x} \wedge \neg\widetilde{y} \wedge \neg\widetilde{xy} \wedge (x' = x + 1) \wedge (y' = y) \\
\alpha_3 &\triangleq \widetilde{x} \wedge \widetilde{y} \wedge \neg\widetilde{xy} \wedge (x' = x + 1) \wedge (y' = y + 1) \\
\alpha_4 &\triangleq \widetilde{x} \wedge \widetilde{y} \wedge \widetilde{xy} \wedge (x' = x \cdot 2) \wedge (y' = y)
\end{aligned}
$$

**Abbildung 5.4:** Aktionen.

- $\begin{bmatrix} x = 1 \\ y = 1 \end{bmatrix} \xrightarrow{\alpha_0} \begin{bmatrix} x = 1 \\ y = 1 \end{bmatrix}$, da Aktion $\alpha_0$ keine Variable involviert und für eine nicht involvierte Variable $x$ stets gilt: $x' = x$. Da das reduzierte Transitionssystem konstruiert wird, wird kein neuer Zustand, sondern lediglich eine Kante $s_0 \xrightarrow{\alpha_0} s_0$ hinzugefügt.
- $\begin{bmatrix} x = 1 \\ y = 1 \end{bmatrix} \xrightarrow{\alpha_1} \begin{bmatrix} x = 1 \\ y = 2 \end{bmatrix}$, da Aktion $\alpha_1$ den Wert der Variablen $y$ inkrementiert, die Variable $x$ jedoch nicht involviert. Dieser Zustand $s_1 \triangleq \begin{bmatrix} x = 1 \\ y = 2 \end{bmatrix}$ wird zusammen mit der Kante $s_0 \xrightarrow{\alpha_1} s_1$ zum Transitionssystem hinzugefügt.
- $\begin{bmatrix} x = 1 \\ y = 1 \end{bmatrix} \xrightarrow{\alpha_2} \begin{bmatrix} x = 2 \\ y = 1 \end{bmatrix}$, da Aktion $\alpha_2$ den Wert der Variablen $x$ inkrementiert, die Variable $y$ jedoch nicht involviert. Dieser Zustand $s_2 \triangleq \begin{bmatrix} x = 2 \\ y = 1 \end{bmatrix}$ wird zusammen mit

der Kante $s_0 \xrightarrow{\alpha_2} s_2$ zum Transitionssystem hinzugefügt.

- $\left[\begin{smallmatrix} x=1 \\ y=1 \end{smallmatrix}\right] \xrightarrow{\alpha_3} \left[\begin{smallmatrix} x=2 \\ y=2 \end{smallmatrix}\right]$, da Aktion $\alpha_3$ die Werte der Variablen $x$ und $y$ inkrementiert. Die Information, dass dabei nicht $\widetilde{xy}$ gilt kann, aus dem Zustand alleine nicht abgelesen werden, ist jedoch in der Aktion $\alpha_3$ selbst enthalten. Der Zustand $s_3 \triangleq \left[\begin{smallmatrix} x=2 \\ y=2 \end{smallmatrix}\right]$ wird zusammen mit der Kante $s_0 \xrightarrow{\alpha_3} s_3$ zum Transitionssystem hinzugefügt.
- $\left[\begin{smallmatrix} x=1 \\ y=1 \end{smallmatrix}\right] \xrightarrow{\alpha_4} \left[\begin{smallmatrix} x=2 \\ y=1 \end{smallmatrix}\right]$, da Aktion $\alpha_4$ den Wert der Variablen $x$ verdoppelt und den Wert von $y$ unverändert lässt. Wieder ist die Information, dass $y$ involviert ist, in der Aktion $\alpha_4$ enthalten und könnte nicht nur anhand des Tupels abgelesen werden (die Umkehrung, d.h. $(y' = y) \Rightarrow \neg\widetilde{y}$ gilt nicht). Der Nachfolgezustand entspricht dem bereits gesehen Zustand $s_2$, sodass nur ein Zustandsübergang $s_0 \xrightarrow{\alpha_4} s_2$ hinzugefügt werden muss.

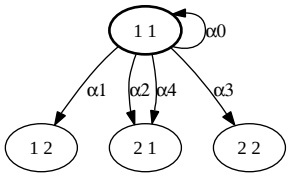


**Abbildung 5.5:** Anfangszustand mit berechneten Nachfolgezuständen.

Abbildung 5.5 zeigt den Anfangszustand mit seinen Nachfolgezuständen. Bei der Zustandsbeschriftung wurde aus Platzgründen „$x =$“ bzw. „$y =$“ weggelassen. Die Reihenfolge der Variablenwerte entspricht der Reihenfolge der Variablen in der Präambel der Spezifikation. Des Weiteren beschreiben wir den Anfangszustand mit einer fett gezeichneten Ellipse.

Ausgehend von den neu hinzugefügten Nachfolgezuständen des Anfangszustandes wird nun rekursiv das gesamte reduzierte Transitionssystem (vgl. Abb. 5.6) aufgebaut, bis schließlich keine neuen Zustände mehr hinzugefügt werden. Dabei spielt der Wertebereich der Variablen eine wichtige Rolle, da er eine stets endliche Zustandsanzahl garantiert. Beispielsweise hat der Zustand $\left[\begin{smallmatrix} x=2 \\ y=2 \end{smallmatrix}\right]$ hat keinen Nachfolgezustand bzgl. Aktion $\alpha_1$, da $\alpha_1$ den Wert der Variablen $y$ inkrementieren und damit den Wertebereich $\mathit{range}(y) = \{1, 2\}$ verlassen würde.

Das reduzierte Transitionssystem (s. Abb. 5.6) enthält nun zu jedem Ablauf der Eingabespezifikation ein entsprechendes Interleaving. Der Ablauf $\sigma_1$ (vgl. Abb. 2.2) kann dabei durch zwei Interleavings beschrieben werden (vgl. Abb 5.7), da der Schritt $S_{C_0}$ sowohl die Schrittformel $\alpha_2$, als auch $\alpha_4$ erfüllt und so der Zustand $\left[\begin{smallmatrix} x=2 \\ y=1 \end{smallmatrix}\right]$, der den Schnitt $C_1$ repräsentiert, durch zwei Aktionen ($\alpha_2$ und $\alpha_4$) vom Anfangszustand erreicht wird. Diese Interleavings entsprechen den beiden markierten Pfade im reduzierten Transitionssystem.

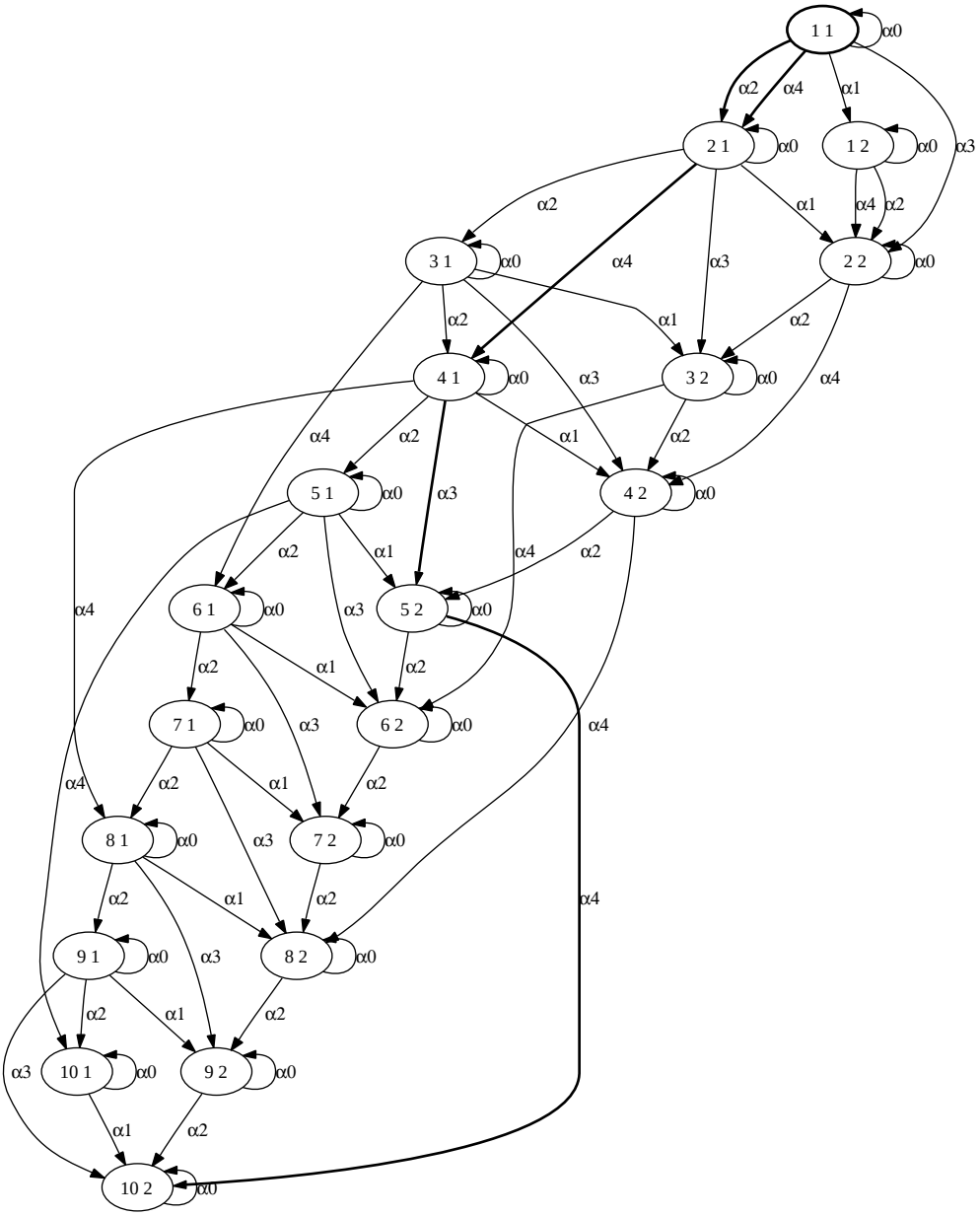


**Abbildung 5.6:** Reduziertes Transitionssystem mit zwei markierten Interleavings von Ablauf $\sigma_1$.

$$\begin{bmatrix} x=1 \\ y=1 \end{bmatrix} \xrightarrow{\alpha_2} \begin{bmatrix} x=2 \\ y=1 \end{bmatrix} \xrightarrow{\alpha_4} \begin{bmatrix} x=4 \\ y=1 \end{bmatrix} \xrightarrow{\alpha_3} \begin{bmatrix} x=5 \\ y=2 \end{bmatrix} \xrightarrow{\alpha_4} \begin{bmatrix} x=10 \\ y=2 \end{bmatrix}$$

$$\begin{bmatrix} x=1 \\ y=1 \end{bmatrix} \xrightarrow{\alpha_4} \begin{bmatrix} x=2 \\ y=1 \end{bmatrix} \xrightarrow{\alpha_4} \begin{bmatrix} x=4 \\ y=1 \end{bmatrix} \xrightarrow{\alpha_3} \begin{bmatrix} x=5 \\ y=2 \end{bmatrix} \xrightarrow{\alpha_4} \begin{bmatrix} x=10 \\ y=2 \end{bmatrix}$$

**Abbildung 5.7:** Zwei mögliche Interleavings von Ablauf $\sigma_1$.

## 5.2 Wechselseitiger Ausschluss

Verteilte Algorithmen bestehen meist aus einem Netzwerk kooperierender Komponenten. Dabei kann jeder Komponente einen „kritischen“ Zustand (vgl. [Dij65]) besitzen. Das Problem des wechselseitigen Ausschlusses (engl. *mutual exclusion*, kurz *mutex*) ist, zu gewährleisten, dass stets höchstens eine Komponente gleichzeitig kritisch ist.

In [Rei98] wurden mehrere Mutex-Algorithmen mit Petrinetzen [Rei86a] modelliert und ihre Eigenschaften verifiziert. Wir wollen in dieser Arbeit nicht detailliert auf Petrinetzmodelle eingehen, sondern wollen an einem einfachen Beispiel zeigen, wie sie in TLDA-Spezifikationen überführt werden können.

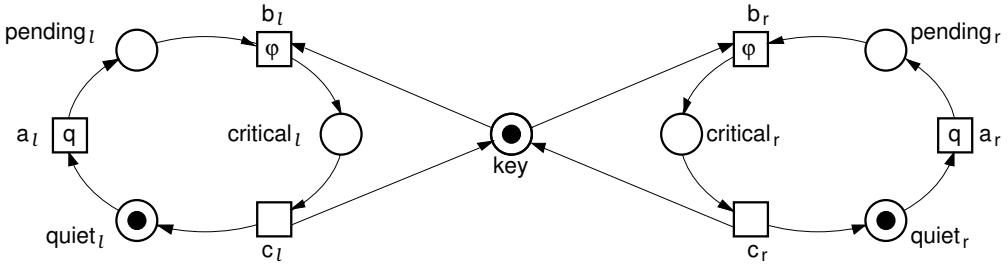


**Abbildung 5.8:** Petrinetzmodell eines Mutex-Algorithmus aus [Rei98].

Abbildung 5.8 zeigt ein Petrinetzmodell eines einfachen Mutex-Algorithmus. Jede Komponente ($l$ und $r$) hat dabei drei Zustände (*quiet*, *pending* und *critical*). Um den wechselseitigen Ausschluss von *critical* zu gewährleisten, wird ein Semaphor (*key*) eingeführt, das beim Übergang in den kritischen Zustand von der jeweiligen Komponente aufgenommen werden muss und so die andere Komponente am Übergang in den kritischen Zustand hindert.

Wir spezifizieren diesen Algorithmus in TLDA$^+$, indem wir jede Komponente und das Semaphor mit jeweils einer Variablen (*agentL*, *agentR* und *key*) beschreiben. Abbildung 5.9 zeigt die TLDA$^+$-Spezifikation des Mutex-Algorithmus.

Die Wertebereiche der Variablen ergeben sich direkt aus den Stellen des Petrinetzes. Die Zustandsformel *Init* können wir aus der Anfangsmarkierung herleiten. In den ersten drei Konjunktionsgliedern der Schrittformel *Next* beschreiben wir die lokalen Zustandsübergänge der einzelnen Komponenten, die nächsten drei Konjunktionsglieder beschreiben die Synchronisation der Komponenten. Die letzte Zeile verhindert, dass beide Komponenten gleichzeitig ihren Zustand ändern und so evtl. gleichzeitig das Semaphor aufnehmen und in den kritischen Bereich wechseln.

Abbildung 5.10 zeigt die Aktionen, die von TLDC berechnet wurden. Die disjunktive Normalform von *Next* besteht aus 4.320 Klauseln. Von diesen potentiellen Aktionen waren jedoch 4.304 Klauseln widersprüchlich und wurden verworfen. Unter den 16 verbliebenen Aktionen sind offensichtlich einige überflüssige Aktionen enthalten (z.B. $\alpha_1$, $\alpha_2$ und $\alpha_3$,

```
VARIABLES
  agentL \in {"quiet", "pending", "critical"},
  agentR \in {"quiet", "pending", "critical"},
  key \in {0,1}

agentL = "quiet" /\ agentR = "quiet" /\ key = 1

/\ []( agentL~ => ( (agentL = "quiet"    /\ agentL' = "pending")
                 \/ (agentL = "pending"  /\ agentL' = "critical")
                 \/ (agentL = "critical" /\ agentL' = "quiet") ) )

/\ []( agentR~ => ( (agentR = "quiet"    /\ agentR' = "pending")
                 \/ (agentR = "pending"  /\ agentR' = "critical")
                 \/ (agentR = "critical" /\ agentR' = "quiet") ) )

/\ []( key~ => ( (key = 1 /\ key' = 0) \/ (key = 0 /\ key' = 1) ) )

/\ []( key~ => (({key,agentL}~ /\ agentL = "pending"  /\ key = 1)
             \/ ({key,agentL}~ /\ agentL = "critical" /\ key = 0)
             \/ ({key,agentR}~ /\ agentR = "pending"  /\ key = 1)
             \/ ({key,agentR}~ /\ agentR = "critical" /\ key = 0)))

/\ []( (agentL~ /\ agentL != "quiet") => {agentL,key}~ )
/\ []( (agentR~ /\ agentR != "quiet") => {agentR,key}~ )

/\ []( !agentL~ \/ !agentR~ )
```

**Abbildung 5.9:** TLDA$^+$-Spezifikation des Mutex-Algorithmus.

bei denen es sich nur um Sonderfälle von $\alpha_0$ handelt), was jedoch in dieser Arbeit nicht weiter verfolgt wird.

Das reduzierte Transitionssystem (s. Abb. 5.11) besteht aus acht Zuständen, die mit den Werten der Systemvariablen (in der Reihenfolge der Angabe in der Präambel) beschriftet sind. Offensichtlich ist der wechselseitige Ausschluss realisiert, und die lokalen Zustandsänderungen der Komponenten sind korrekt.

In Abschnitt 5.1 zeigten wir bereits, wie Abläufe der Eingabespezifikation durch Interleavings, d.h. Pfade im reduzierten Transitionssystem, repräsentiert wurden. Wir wollen nun den umgekehrten Weg, d.h. von einem Interleaving zu einem entsprechenden Ablauf, illustrieren.

Abbildung 5.12 zeigt ein Interleaving, das einen Zyklus der Komponente *agentR* beschreibt. Zu diesem Interleaving gibt es — wegen des unspezifizierten Verhaltens der Umgebung — unendlich viele Abläufe der Eingabespezifikation. Beschränkt man sich jedoch nur auf die Systemvariablen (*agentL*, *agentR* und *key*), d.h. blendet die Umgebung aus, haben all diese Abläufe die gleiche grafische Darstellung (vgl. Ablauf $\sigma_5$ in Abb. 5.13).

$$\alpha_0 \triangleq \widetilde{\neg agentL} \wedge \widetilde{\neg agentR} \wedge \widetilde{\neg key} \wedge \widetilde{\neg agentL\, agentR} \wedge \widetilde{\neg agentL\, key} \wedge \widetilde{\neg agentR\, key} \wedge \widetilde{\neg agentL\, agentR\, key} \wedge (agentL' = agentL) \wedge (agentR' = agentR) \wedge (key' = key)$$

$$\alpha_1 \triangleq \widetilde{\neg agentL} \wedge \widetilde{\neg agentR} \wedge \widetilde{\neg key} \wedge \widetilde{\neg agentL\, agentR} \wedge \widetilde{\neg agentL\, key} \wedge \widetilde{\neg agentR\, key} \wedge \widetilde{\neg agentL\, agentR\, key} \wedge (agentL' = agentL) \wedge (agentR = \mathsf{quiet}) \wedge (agentR' = agentR) \wedge (key' = key)$$

$$\alpha_2 \triangleq \widetilde{\neg agentL} \wedge \widetilde{\neg agentR} \wedge \widetilde{\neg key} \wedge \widetilde{\neg agentL\, agentR} \wedge \widetilde{\neg agentL\, key} \wedge \widetilde{\neg agentR\, key} \wedge \widetilde{\neg agentL\, agentR\, key} \wedge (agentL = \mathsf{quiet}) \wedge (agentL' = agentL) \wedge (agentR' = agentR) \wedge (key' = key)$$

$$\alpha_3 \triangleq \widetilde{\neg agentL} \wedge \widetilde{\neg agentR} \wedge \widetilde{\neg key} \wedge \widetilde{\neg agentL\, agentR} \wedge \widetilde{\neg agentL\, key} \wedge \widetilde{\neg agentR\, key} \wedge \widetilde{\neg agentL\, agentR\, key} \wedge (agentL = \mathsf{quiet}) \wedge (agentL' = agentL) \wedge (agentR = \mathsf{quiet}) \wedge (agentR' = agentR) \wedge (key' = key)$$

$$\alpha_4 \triangleq \widetilde{\neg agentL} \wedge \widetilde{agentR} \wedge \widetilde{\neg key} \wedge \widetilde{\neg agentL\, agentR} \wedge \widetilde{\neg agentL\, key} \wedge \widetilde{\neg agentR\, key} \wedge \widetilde{\neg agentL\, agentR\, key} \wedge (agentL' = agentL) \wedge (agentR = \mathsf{quiet}) \wedge (agentR' = \mathsf{pending}) \wedge (key' = key)$$

$$\alpha_5 \triangleq \widetilde{\neg agentL} \wedge \widetilde{agentR} \wedge \widetilde{\neg key} \wedge \widetilde{\neg agentL\, agentR} \wedge \widetilde{\neg agentL\, key} \wedge \widetilde{\neg agentR\, key} \wedge \widetilde{\neg agentL\, agentR\, key} \wedge (agentL = \mathsf{quiet}) \wedge (agentL' = agentL) \wedge (agentR = \mathsf{quiet}) \wedge (agentR' = \mathsf{pending}) \wedge (key' = key)$$

$$\alpha_6 \triangleq \widetilde{\neg agentL} \wedge \widetilde{agentR} \wedge \widetilde{key} \wedge \widetilde{\neg agentL\, key} \wedge \widetilde{agentR\, key} \wedge \widetilde{\neg agentL\, agentR\, key} \wedge (agentL' = agentL) \wedge (agentR = \mathsf{pending}) \wedge (agentR' = \mathsf{critical}) \wedge (key = 1) \wedge (key' = 0)$$

$$\alpha_7 \triangleq \widetilde{\neg agentL} \wedge \widetilde{agentR} \wedge \widetilde{key} \wedge \widetilde{\neg agentL\, key} \wedge \widetilde{agentR\, key} \wedge \widetilde{\neg agentL\, agentR\, key} \wedge (agentL = \mathsf{quiet}) \wedge (agentL' = agentL) \wedge (agentR = \mathsf{pending}) \wedge (agentR' = \mathsf{critical}) \wedge (key = 1) \wedge (key' = 0)$$

$$\alpha_8 \triangleq \widetilde{\neg agentL} \wedge \widetilde{agentR} \wedge \widetilde{key} \wedge \widetilde{\neg agentL\, key} \wedge \widetilde{agentR\, key} \wedge \widetilde{\neg agentL\, agentR\, key} \wedge (agentL' = agentL) \wedge (agentR = \mathsf{critical}) \wedge (agentR' = \mathsf{pending}) \wedge (key = 0) \wedge (key' = 1)$$

$$\alpha_9 \triangleq \widetilde{\neg agentL} \wedge \widetilde{agentR} \wedge \widetilde{key} \wedge \widetilde{\neg agentL\, key} \wedge \widetilde{agentR\, key} \wedge \widetilde{\neg agentL\, agentR\, key} \wedge (agentL = \mathsf{quiet}) \wedge (agentL' = agentL) \wedge (agentR = \mathsf{critical}) \wedge (agentR' = \mathsf{pending}) \wedge (key = 0) \wedge (key' = 1)$$

$$\alpha_{10} \triangleq \widetilde{agentL} \wedge \widetilde{\neg agentR} \wedge \widetilde{\neg key} \wedge \widetilde{\neg agentL\, agentR} \wedge \widetilde{\neg agentL\, key} \wedge \widetilde{\neg agentR\, key} \wedge \widetilde{\neg agentL\, agentR\, key} \wedge (agentL = \mathsf{quiet}) \wedge (agentL' = \mathsf{pending}) \wedge (agentR' = agentR) \wedge (key' = key)$$

$$\alpha_{11} \triangleq \widetilde{agentL} \wedge \widetilde{\neg agentR} \wedge \widetilde{\neg key} \wedge \widetilde{\neg agentL\, agentR} \wedge \widetilde{\neg agentL\, key} \wedge \widetilde{\neg agentR\, key} \wedge \widetilde{\neg agentL\, agentR\, key} \wedge (agentL = \mathsf{quiet}) \wedge (agentL' = \mathsf{pending}) \wedge (agentR = \mathsf{quiet}) \wedge (agentR' = agentR) \wedge (key' = key)$$

$$\alpha_{12} \triangleq \widetilde{agentL} \wedge \widetilde{\neg agentR} \wedge \widetilde{key} \wedge \widetilde{agentL\, key} \wedge \widetilde{\neg agentR\, key} \wedge \widetilde{\neg agentL\, agentR\, key} \wedge (agentL = \mathsf{pending}) \wedge (agentL' = \mathsf{critical}) \wedge (agentR' = agentR) \wedge (key = 1) \wedge (key' = 0)$$

$$\alpha_{13} \triangleq \widetilde{agentL} \wedge \widetilde{\neg agentR} \wedge \widetilde{key} \wedge \widetilde{agentL\, key} \wedge \widetilde{\neg agentR\, key} \wedge \widetilde{\neg agentL\, agentR\, key} \wedge agentL = \mathsf{pending}) \wedge (agentL' = \mathsf{critical}) \wedge (agentR = \mathsf{quiet}) \wedge (agentR' = agentR) \wedge (key = 1) \wedge (key' = 0)$$

$$\alpha_{14} \triangleq \widetilde{agentL} \wedge \widetilde{\neg agentR} \wedge \widetilde{key} \wedge \widetilde{agentL\, key} \wedge \widetilde{\neg agentR\, key} \wedge \widetilde{\neg agentL\, agentR\, key} \wedge (agentL = \mathsf{critical}) \wedge (agentL' = \mathsf{pending}) \wedge (agentR' = agentR) \wedge (key = 0) \wedge (key' = 1)$$

$$\alpha_{15} \triangleq \widetilde{agentL} \wedge \widetilde{\neg agentR} \wedge \widetilde{key} \wedge \widetilde{agentL\, key} \wedge \widetilde{\neg agentR\, key} \wedge \widetilde{\neg agentL\, agentR\, key} \wedge (agentL = \mathsf{critical}) \wedge (agentL' = \mathsf{pending}) \wedge (agentR = \mathsf{quiet}) \wedge (agentR' = agentR) \wedge (key = 0) \wedge (key' = 1)$$

**Abbildung 5.10:** Berechnete, widerspruchsfreie Aktionen.

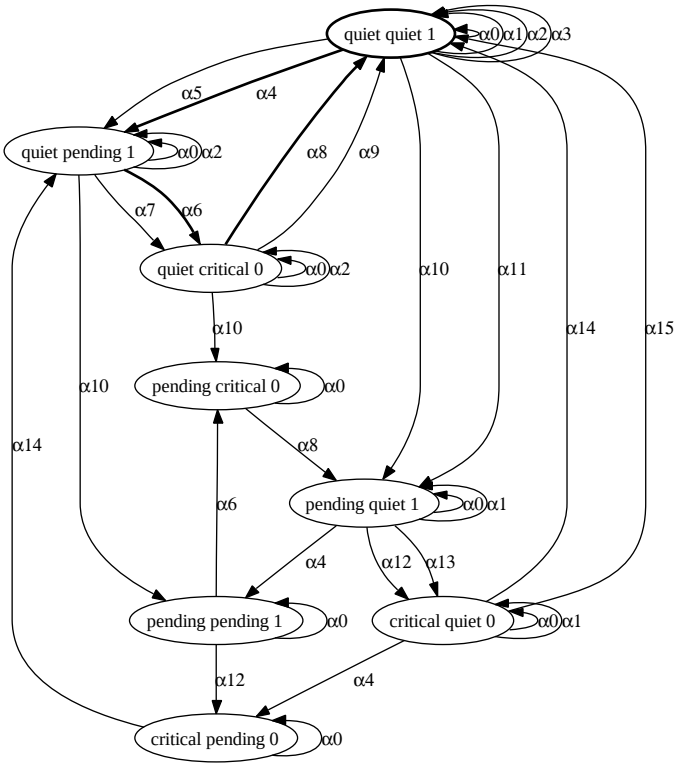


**Abbildung 5.11:** Reduziertes Transitionssystem mit markiertem Interleaving von Ablauf $\sigma_5$.

$$\begin{bmatrix} agentL = \mathsf{quiet} \\ agentR = \mathsf{quiet} \\ key = 1 \end{bmatrix} \xrightarrow{\alpha_4} \begin{bmatrix} agentL = \mathsf{quiet} \\ agentR = \mathsf{pending} \\ key = 1 \end{bmatrix} \xrightarrow{\alpha_6} \begin{bmatrix} agentL = \mathsf{quiet} \\ agentR = \mathsf{critical} \\ key = 0 \end{bmatrix} \xrightarrow{\alpha_8} \begin{bmatrix} agentL = \mathsf{quiet} \\ agentR = \mathsf{quiet} \\ key = 1 \end{bmatrix}$$

**Abbildung 5.12:** Wechselseitiger Ausschluss: ein Interleaving.

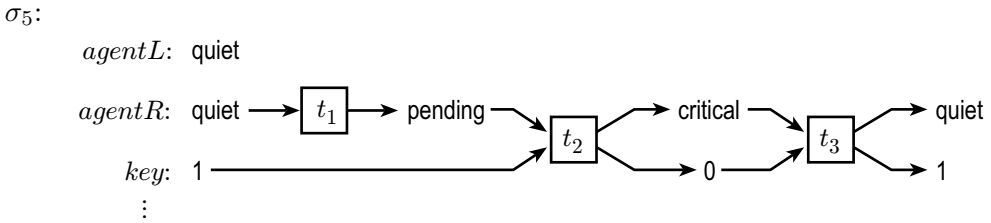


**Abbildung 5.13:** Ablauf $\sigma_5$.

# 6 Zusammenfassung und Ausblick

## 6.1 Zusammenfassung

Ziel dieser Arbeit war es, die Grundlage für die computergestützte Verifikation von TLDA-Spezifikationen zu legen. Dazu haben wir eine Interleavingsemantik für die Logik TLDA erarbeitet, Algorithmen zur Konstruktion reduzierter Transitionssysteme vorgestellt und diese prototypisch im Werkzeug TLDC implementiert.

Wir zeigten in Kapitel 3, dass der einzige bestehende Ansatz, eine transitionsbasierte Interleavingsemantik aus [AR03], für die Implementierung und späteres Modelchecking nicht geeignet ist und wählten einen neuen Aktionsbegriff und definierten die schrittbasierte Interleavingsemantik. Dabei mussten wir uns auf die Klasse der umgebungsinvarianten Spezifikationen in Normalform beschränken, für die wir einen engen Zusammenhang zwischen den Abläufen einer Spezifikation (also dem semantischen Modell der Halbordnungssemantik) und dem Transitionssystem motivieren konnten. Der gewählte Zustandsbegriff erlaubt es außerdem, das Transitionssystem zu reduzieren, sodass es für Systeme mit eingeschränkten Variablenwertebereichen stets endlich ist.

Neben dieser theoretischen Vorarbeit war die konkrete Implementierung der Semantik der zweite Hauptteil dieser Arbeit. In Kapitel 4 mussten wir dazu zunächst die Spezifikationssprache $\text{TLDA}^+$ entwickeln. Ferner rechtfertigten wir eine Aufteilung der Aufgaben in Übersetzung der Spezifikation in einen Programmtext und anschließende Konstruktion des reduzierten Transitionssystemes. Insbesondere bei der Übersetzung konnten wir so auf bewährte Werkzeuge zurückgreifen und erreichten einen hohen Grad an Modularität, Wartung und zukünftige Erweiterbarkeit des Prototyps TLDC sehr erleichtert.

Obwohl einige Probleme, auf die wir im folgenden Abschnitt eingehen werden, offen blieben, konnten wir in Kapitel 5 bereits einfache Systeme spezifizieren und mit dem Prototyp TLDC die reduzierten Transitionssysteme aufbauen und die entwickelte Semantik illustrieren.

## 6.2 Ausblick

Im Rahmen dieser Arbeit konnten wir nicht auf alle Aspekte des breiten Themengebietes eingehen, sodass viele interessante Fragestellungen und Probleme offen blieben und Gegenstand zukünftiger Forschungsarbeit sind.

In Kapitel 3 haben wir die Beziehung zwischen den Abläufen einer Spezifikation und deren Interleavings, also die Rechtfertigung der Interleavingsemantik, nur informal motiviert. Neben diesem Zusammenhang muss auch der Reduktionsschritt zum reduzierten Transitionssystem formal gerechtfertigt und bewiesen werden.

Bereits in Abschnitt 4.6 sprachen wir die Probleme an, die bei der automatischen Konstruktion des reduzierten Transitionssystemes auftreten. Um der Komplexität dieser Probleme entgegenzuwirken müssen leistungsfähigere Algorithmen und Heuristiken entwickelt werden, damit auch größere Systeme analysiert werden können. Insbesondere muss dazu eine effizientere Implementierung der Regel aus Korollar 2.1 gefunden werden, die einen Großteil der Rechenzeit des Übersetzungsschrittes ausmacht. Zusammen mit schnellen Algorithmen zur Überprüfung von Klauseln auf Widersprüchlichkeit wäre so ein on-the-fly-Verfahren zum Aufbau der disjunktiven Normalform denkbar. Anschließend könnten unter den widerspruchsfreien Aktionen jene in Abschnitt 5.2 beschriebenen „überflüssigen“ Aktionen erkannt und entfernt werden, um die Aktionszahl weiter zu senken.

Ein weiterer Aspekt ist die Erweiterung der Spezifikationssprache TLDA$^+$, um die Spezifikation verteilter Systeme zu vereinfachen. Dazu müssen die in Abschnitt 4.5 beschriebenen Einschränkungen analysiert und evtl. beseitigt werden. Durch Ausarbeitung geeigneter Umformungsregeln wäre es außerdem denkbar, die Einschränkung auf Spezifikationen in Normalform aufzuheben. Des Weiteren könnten die in [Ale05] beschriebenen syntaktischen Kriterien der Umgebungsinvarianz implementiert werden und in die Analyse der Eingabespezifikation einfließen.

Der im Rahmen dieser Arbeit entstandene Prototyp bildet die Grundlage für eine weitere Arbeit an einem TLDA-Modelchecker. Das konstruierte Transitionssystem ist dabei der Ausgangspunkt für die Entwicklung von Algorithmen, welche TLDA-Spezifikationen durch Suche im reduzierten Transitionssystem verifizieren bzw. falsifizieren können. Um der Zustandsraumexplosion entgegenzuwirken ist auch die Anpassung vorhandener Reduktionstechniken bzw. die Entwicklung TLDA-spezifischer Algorithmen notwendig, die dann in den Prototypen TLDC eingearbeitet werden können.

# Abbildungsverzeichnis


2.1 Ablauf $\sigma_1$. . . . . 9
2.2 Schnitte und Schritte in $\sigma_1$. . . . . 10
2.3 Restriktion von $\sigma_1$ auf $\{x\}$. . . . . 18

3.1 Ein Interleaving mit den Variablen $x$, $y$ und den Aktionen $a_1$, $a_2$. . . . . 19
3.2 Ablauf $\sigma_2$ mit geordneten Transitionen: $t_1 \prec t_2$. . . . . 20
3.3 Ablauf $\sigma_3$ mit nebenläufigen Transitionen: $t_1 \not\prec t_2$ und $t_2 \not\prec t_1$. . . . . 20
3.4 Einziges transitionsbasiertes Interleaving von Ablauf $\sigma_2$. . . . . 21
3.5 Alle möglichen transitionsbasierten Interleavings von Ablauf $\sigma_3$. . . . . 21
3.6 Ablauf $\sigma_4$ mit Schnitt $C^*$. . . . . 23
3.7 Ablauf $\sigma_4^*$ mit Umgebungsvariable *env*. . . . . 23
3.8 Alle möglichen schrittbasierten Interleavings von Ablauf $\sigma_3$. . . . . 24

4.1 TLDA$^+$-Lexik. . . . . 28
4.2 Schematischer Ablauf der Konstruktion des reduzierten Transitionssystemes. 30

5.1 TLDA$^+$-Spezifikation des einleitenden Beispieles. . . . . 36
5.2 Von TLDC berechnete disjunktive Normalform von *Next*. . . . . 37
5.3 Disjunktive Normalform von *Next* ohne widersprüchliche Klauseln. . . . . 37
5.4 Aktionen. . . . . 37
5.5 Anfangszustand mit berechneten Nachfolgezuständen. . . . . 38
5.6 Reduziertes Transitionssystem mit zwei markierten Interleavings von Ablauf $\sigma_1$. . . . . 39
5.7 Zwei mögliche Interleavings von Ablauf $\sigma_1$. . . . . 39
5.8 Petrinetzmodell eines Mutex-Algorithmus aus [Rei98]. . . . . 40
5.9 TLDA$^+$-Spezifikation des Mutex-Algorithmus. . . . . 41
5.10 Berechnete, widerspruchsfreie Aktionen. . . . . 42
5.11 Reduziertes Transitionssystem mit markiertem Interleaving von Ablauf $\sigma_5$. 43
5.12 Wechselseitiger Ausschluss: ein Interleaving. . . . . 43
5.13 Ablauf $\sigma_5$. . . . . 43

## Literaturverzeichnis


[Ale04] Alexander, Adrianna: Composition of Temporal Logic Specifications. In: Cortadella, Jordi (Hrsg.) ; Reisig, Wolfgang (Hrsg.): *Applications and Theory of Petri Nets 2004, 25th International Conference (ICATPN 2004)*, Springer-Verlag, September 2004 (LNCS 3099). – ISBN 3–540–22236–7, S. 98–116

[Ale05] Alexander, Adrianna: *Komposition temporallogischer Spezifikationen*, Humboldt-Universität zu Berlin, Diss., 2005. – unveröffentlicht

[AR03] Alexander, Adrianna ; Reisig, Wolfgang: Logic of Involved Variables – System Specification with Temporal Logic of Distributed Actions. In: *Application of Concurrency to System Design, 3rd International Conference (ACSD 2003)*, IEEE Computer Society, Juni 2003. – ISBN 0–7695–1887–7, S. 167–176

[Dij65] Dijkstra, Edsger W.: Solutions of a Problem in Concurrent Programming Control. In: *Communications of the ACM (CACM)* 8 (1965), September, Nr. 9, S. 569. – ISSN 0001–0782

[DS04] Donnelly, Charles ; Stallman, Richard: *Bison – The Yacc-compatible Parser Generator*. 2.0. The Free Software Foundation, Dezember 2004

[GN00] Gansner, Emden R. ; North, Stephen C.: An open graph visualization system and its applications to software engineering. In: *Software — Practice and Experience* 30 (2000), September, Nr. 11, S. 1203–1233. – ISSN 0038–0644

[Hol97] Holzmann, Gerard J.: The Model Checker SPIN. In: *IEEE Transactions on Software Engineering (TSE)* 23 (1997), Mai, Nr. 5, S. 279–295. – ISSN 0098–5589

[Lam94] Lamport, Leslie: The Temporal Logic of Actions. In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 16 (1994), Mai, Nr. 3, S. 872–923. – ISSN 0164–0925

[Lam02] Lamport, Leslie: *Specifying Systems: The TLA$^+$ Language and Tools for Hardware and Software Engineers*. Addison-Wesley Longman, Juli 2002. – ISBN 0–321–14306–X

[LMTY02] Lamport, Leslie ; Matthews, John ; Tuttle, Marc ; Yu, Yuan: Specifying and verifying systems with TLA$^+$. In: *Proceedings of the Tenth ACM SIGOPS European Workshop*, Association for Computing Machinery (ACM), September 2002, S. 45–48

[LP02] v. Löwis, Martin ; Piefel, Michael: The Term Processor Kimwitu++. In: Callaos, Nagib (Hrsg.) ; Leng, Tau (Hrsg.) ; Sanchez, Belkis (Hrsg.): *6th World Multiconference on Systemics, Cybernetics and Informatics (ISAS/SCI 2002)* Bd. V, International Institute of Informatics and Systemics (IIIS), 2002. – ISBN 980–07–8150–1

[Pax95] Paxon, Vern: *Flex, version 2.5 – A fast scanner generator.* 2.5. Free Software Foundation, März 1995

[Pnu77] Pnueli, Amir: The temporal logic of programs. In: *Foundations of Computer Science, 18th IEEE Symposium (FOCS-77)*, IEEE Computer Society, Oktober 1977, S. 46–57

[Pnu79] Pnueli, Amir: The Temporal Semantics of Concurrent Programs. In: Kahn, Gilles (Hrsg.): *Semantics of Concurrent Computation*, Springer-Verlag, Juli 1979 (LNCS 70). – ISBN 3–540–09511–X, S. 1–20

[Rei86a] Reisig, Wolfgang: *Petrinetze: Eine Einführung.* 2. Auflage. Springer-Verlag, Januar 1986. – ISBN 3–540–16622–X

[Rei86b] Reisig, Wolfgang: A strong part of concurrency. In: Rozenberg, Grzegorz (Hrsg.): *Applications and Theory of Petri Nets, 7th European Workshop*, Springer-Verlag, Juni 1986 (LNCS 266). – ISBN 3–540–18086–9, S. 238–272

[Rei98] Reisig, Wolfgang: *Elements of Distributed Algorithms: Modeling and Analysis with Petri Nets.* Springer-Verlag, November 1998. – ISBN 3–540–62752–9

[Sch00] Schmidt, Karsten: LoLA: A Low Level Analyser. In: Nielsen, Mogens (Hrsg.) ; Simpson, Dan (Hrsg.): *Application and Theory of Petri Nets, 21st International Conference (ICATPN 2000)*, Springer-Verlag, Juni 2000 (LNCS 1825). – ISBN 3–540–67693–7, S. 465–474

[Str97] Stroustrup, Bjarne: *The C++ Programming Language.* 3rd edition. Addison-Wesley Longman, Juli 1997. – ISBN 0–201–88954–4

[YML99] Yu, Yuan ; Manolios, Panagiotis ; Lamport, Leslie: Model Checking TLA$^+$ Specifications. In: Pierre, Laurence (Hrsg.) ; Kropf, Thomas (Hrsg.): *Correct Hardware Design and Verification Methods (CHARME'99)*, Springer-Verlag, Juni 1999 (LNCS 1703). – ISBN 3–540–66559–5, S. 54–66