Diplomarbeit

Formale Fundierung und effizientere Algorithmen für die schrittbasierte TLDA-Interleavingsemantik

Niels Lohmann

7. September 2005

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

Gutachter:
Peter Massuthe
Prof. Dr. Wolfgang Reisig

Zusammenfassung

Die Temporal Logic of Distributed Actions (TLDA) ist eine neue temporale Logik mit einer Halbordnungssemantik und eignet sich somit zur Spezifikation verteilter Systeme. Allerdings existieren für TLDA noch keine Werkzeuge, die die Spezifikation und Verifikation unterstützen. Die Verfügbarkeit solcher Werkzeuge trägt jedoch entscheidend zur Verbreitung der Logik bei.

In [Loh05] wurde mit der Ausarbeitung und Implementierung einer schrittbasierten Interleavingsemantik für TLDA die Grundlage für die Entwicklung eines expliziten TLDA-Modelcheckers gelegt. Der genaue Zusammenhang zwischen der Halbordnungs- und Interleavingsemantik wurde jedoch nicht bewiesen. Außerdem handelt es sich bei der Implementierung der Interleavingsemantik um einen Brute-Force-Prototyp, der nur für sehr einfach Spezifikationen in der Lage ist das Transitionssystem zu konstruieren.

Die formale Fundierung sowie die Ausarbeitung effizienter Algorithmen und Datenstrukturen ist Inhalt dieser Arbeit. Anhand mehrerer Fallstudien untersuchen wir die Leistungsfähigkeit des erweiterten Prototyps und illustrieren die Beziehung zwischen der Halbordnungs- und Interleavingsemantik.

Inhaltsverzeichnis

1 Einleitung

Systeme werden immer 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 [CES86].

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.

Da es derzeit wenig Erfahrung beim Modelchecking halbordnungsbasierter Formalismen gibt, ist eine Anpassung vorhandener Arbeit an TLDA nicht möglich. Explizite Modelchecker setzen Transitionssysteme – Graphen, dessen Knoten Zuständen und dessen Kanten Zustandsübergängen entsprechen und in denen das Verifikationsproblem durch Suche im Zustandsraum gelöst wird – und damit Interleavingsemantiken voraus.

In [Loh05] wurden verschiedene Interleavingsemantiken für TLDA diskutiert und eine schrittbasierte Interleavingsemantik vorgestellt und prototypisch implementiert – im Rahmen dieser Arbeit bezeichnen wir diesen ersten Prototypen als Brute-Force-Imple-

mentierung. Das zu einer Spezifikation konstruierte Transitionssystem bildet die Grundlage für die weitere Entwicklung eines expliziten Modelcheckers für TLDA. Um jedoch anhand eines Transitionssystemes Aussagen über die Abläufe eines spezifizierten Systemes abzuleiten, muss der Zusammenhang zwischen Halbordnungs- und Interleavingsemantik formal fundiert werden. Weiterhin ergeben sich durch den gewählten Zustands- und Aktionsbegriff der schrittbasierten Interleavingsemantik Komplexitätsprobleme: Der Rechen- und Speicheraufwand, der notwendig ist, die Aktionen einer Spezifikation zu berechnen, wächst exponentiell in der Länge der Spezifikation. Diese Probleme wurden bisher nicht gelöst, sodass der Brute-Force-Prototyp nur die Analyse von einfachen Spezifikationen erlaubt.

Die vorliegende Arbeit ergänzt [Loh05] um eine formale Fundierung der schrittbasierten Interleavingsemantik, stellt verschiedene verbesserte Algorithmen und Datenstrukturen vor, erweitert die Implementierung der Semantik und analysiert ihre Leistungsfähigkeit anhand mehrerer Fallstudien.

Dazu führen wir in Kapitel 2 in die Logik TLDA ein und legen damit die theoretischen Grundlagen dieser Arbeit. Insbesondere die Definition der Halbordnungssemantik und der Umbebungsinvarianz spielen dabei entscheidende Rollen.

In Kapitel 3 wiederholen wir zunächst die Intuition der schrittbasierten Interleavingsemantik und definieren die Konstruktionsvorschrift für Transitionssysteme. Des Weiteren untersuchen wir den Zusammenhang zwischen den Abläufen einer Spezifikation und den Pfaden im Transitionssystem, den Interleavings, und legen die Grundlage für die formale Definition einer TLDA-Interleavingsemantik.

Anschließend analysieren wir in Kapitel 4 die Probleme, die sich durch die naive Implementierung der Konstruktionsvorschrift des Transitionssystemes ergeben und stellen verschiedene Verbesserungsansätze, Algorithmen und Datenstrukturen vor, mit denen wir die Brute-Force-Implementierung erweitern.

Die Effizienz des erweiterten Prototyps untersuchen wir in Kapitel 5 anhand mehrerer Fallstudien. Außerdem illustrieren wir den Zusammenhang zwischen Interleaving- und Halbordnungssemantik.

Zuletzt fassen wir in Kapitel 6 die Resultate der Arbeit zusammen, beschreiben die offen gebliebenen Probleme der automatischen Konstruktion der Transitionssysteme und diskutieren mögliche Lösungswege.

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 [Rei86] 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:*

i. Für jede Variable $x \in \mathcal{V}ar$ gibt es genau eine Transition $t \in T$ mit $t(x) = i$ für $0 \leq i < length(H_x) - 1$.

ii. Für alle Transitionen $t \in T$ und alle Variablen $x \in dom(t)$ gilt: $0 \leq t(x) < length(H_x) - 1$.

iii. 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 σ, 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 σ.* ⌟

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 σ 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 σ 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* σ. 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 σ *ein Ablauf,* C_0 *der Anfangsschnitt von* σ *und* $C \geq C_0$ *ein Schnitt in* σ. C ist von C_0 aus durch Schritte erreichbar*, falls in* σ *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 $n \geq 0$. ⌟

In Abläufen lässt sich folgende Eigenschaft beobachten, die wir im folgenden Kapitel für den Beweis von Satz 3.2 benötigen:

Lemma 2.1 *Sei* σ *ein Ablauf mit Anfangsschnitt* C_0 *und einem beliebigen Schnitt* C^*. *Sei ferner* $\mathcal{A} \subset \mathcal{V}ar$ *eine endliche, nichtleere Variablenmenge. Dann existiert ein vom Anfangsschnitt* C_0 *aus durch Schritte erreichbarer Schnitt* C *mit* $C(x) \leq C^*(x)$ *für alle* $x \in \mathcal{A}$ *und* $C(x) = C^*(x)$ *für einige* $x \in \mathcal{A}$.

Beweis. Angenommen, es existiert kein solcher Schnitt C in σ. Dann existiert in σ ein durch Schritte vom Anfangsschnitt aus erreichbarer Schnitt C_1 und einen dort beginnender Schritt $S_{C_1} = (C_1, C_1', T_{C_1})$ mit $C_1(x) < C^*(x) < C_1'(x)$ für alle $x \in \mathcal{A}$. Für eine beliebige Systemvariable $x \in \mathcal{A}$ gilt somit $C_1'(x) - C_1(x) > 2$. Dies ist ein Widerspruch zur Definition des Schrittes (vgl. Def. 7). □

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 σ_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.

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.

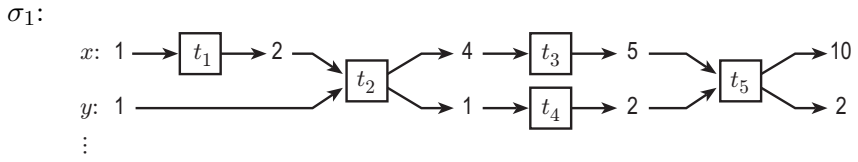


Abbildung 2.1: Ablauf σ_1.

Jede Transition in σ_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.

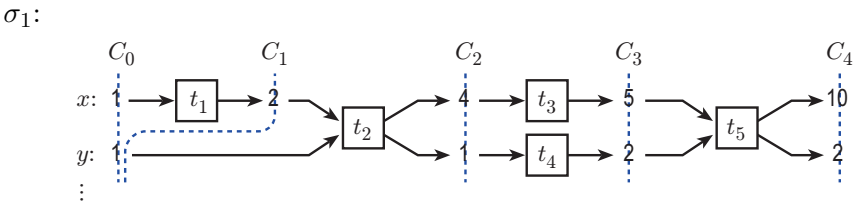


Abbildung 2.2: Schnitte und Schritte in σ_1.

Abbildung 2.2 zeigt alle Schnitte des Ablaufes σ_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-Schreibweise und schreiben aRb anstelle von $R(a, b)$.

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* φ *und* ψ *Schrittformeln sind, so sind auch* $\neg\varphi$*,* $(\varphi \wedge \psi)$ *und* $(\varphi \Rightarrow \psi)$ *Schrittformeln.*
- *Wenn* $\mathsf{m} \in \mathcal{V}ar_{rigid}$ *und* φ *eine Schrittformel ist, so ist auch* $\exists \mathsf{m}\ \varphi$ *eine Schrittformel.*
- *Wenn* φ *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* Φ *und* Ψ *Ablaufformeln sind, so sind auch* $\neg\Phi$*,* $(\Phi \wedge \Psi)$ *und* $(\Phi \Rightarrow \Psi)$ *Ablaufformeln.*
- *Wenn* Φ *eine Ablaufformel ist, so auch* $\Box\Phi$*.*
- *Wenn* $\mathsf{m} \in \mathcal{V}ar_{rigid}$ *und* Φ *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 Schrittformel, 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 β bzgl. 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* σ *und* C' *sein Nachfolgeschnitt. Seien* I *eine Interpretation von* $(\mathcal{F}, \mathcal{R})$ *über* $\mathcal{V}al$ *und* β *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* σ*. Seien* I *eine Interpretation von* $(\mathcal{F}, \mathcal{R})$ *über* $\mathcal{V}al$ *und* β *eine Belegung für* $\mathcal{V}ar_{rigid}$*. Die Gültigkeit einer Schrittformel* φ *im Schritt* S_C *unter der Belegung* β *und der Interpretation* I*, notiert durch* $(S_C, I) \Vdash_\beta \varphi$*, ist wie folgt definiert:*

- $(S_C, I) \mid\!\models_\beta \widetilde{a}$, *falls* $a \subseteq dom(t)$ *für ein* $t \in T_C$,
- $(S_C, I) \mid\!\models_\beta R(t_1, \ldots, t_n)$, *falls* $R \in \mathcal{R}$ *ein n-stelliges Relationssymbol ist,* $t_1, \ldots, t_n$ *Terme sind und es gilt:* $(I_C(t_1), \ldots, I_C(t_n)) \in R^I$,
- $(S_C, I) \mid\!\models_\beta \neg\varphi$, *falls nicht gilt:* $(S_C, I) \mid\!\models_\beta \varphi$,
- $(S_C, I) \mid\!\models_\beta (\varphi \wedge \psi)$, *falls gilt:* $(S_C, I) \mid\!\models_\beta \varphi$ *und* $(S_C, I) \mid\!\models_\beta \psi$,
- $(S_C, I) \mid\!\models_\beta (\varphi \Rightarrow \psi)$, *falls gilt:* $(S_C, I) \mid\!\models_\beta \neg\varphi$ *oder* $(S_C, I) \mid\!\models_\beta \psi$,
- $(S_C, I) \mid\!\models_\beta \exists \mathsf{m}\ \varphi$, *falls* $\mathsf{m} \in \mathcal{V}ar_{rigid}$ *und es einen Wert* $q \in \mathcal{V}al$ *gibt, sodass* $(S_C, I) \mid\!\models_{\beta[\mathsf{m} \leftarrow q]} \varphi$, *und*
- $(S_C, I) \mid\!\models_\beta \widetilde{\exists}\varphi$, *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* σ^* *ist mit* $(S^*_C, I) \mid\!\models_\beta \varphi$.

⌟

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 φ, und der Ausdruck $\widetilde{\exists}\varphi$ ist genau dann wahr, wenn ein Ablauf existiert, der mindestens im Schnitt C mit σ übereinstimmt und in dem ein in C anfangender Schritt existiert, der φ 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 $S_C \mid\!\models \varphi$ anstelle von $(S_C, I) \mid\!\models_\beta \varphi$.

Beispiel 3. Wir wollen nun einige Schrittformeln anhand des in Abbildung 2.2 angegebenen Ablaufes σ_1 auswerten und konzentrieren uns auf die durch $\sim$-Variablen gebildeten Schrittformeln. Es gilt:

$$
\begin{array}{ll}
S_{C_1} \mid\!\models \widetilde{xy} & \text{da } \{x, y\} \subseteq dom(t_2) \text{ und } t_2 \in T_{C_1} \\
S_{C_2} \mid\!\models \widetilde{x} \wedge \widetilde{y} & \text{da } x \in dom(t_3),\ y \in dom(t_4) \text{ und } t_3, t_4 \in T_{C_2} \\
S_{C_2} \mid\!\models \neg\widetilde{xy} & \text{da es keine Transition } t \in T_{S_{C_2}} \text{ gibt mit } \{x, y\} \subseteq dom(t)
\end{array}
$$

⌟

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 σ *ein Ablauf und* C *ein Schnitt in* σ. *Seien* I *eine Interpretation von* $(\mathcal{F}, \mathcal{R})$ *über* $\mathcal{V}al$ *und* β *eine Belegung für* $\mathcal{V}ar_{rigid}$. *Die Gültigkeit einer Ablaufformel* Φ *im Schnitt* C

unter der Belegung β und der Interpretation I, notiert durch $(\sigma, C, I) \models_\beta \Phi$, ist wie folgt definiert:

- $(\sigma, C, I) \models_\beta \varphi$, *falls* φ *eine Schrittformel ist und es gilt:* $(S_C, I) \|\models_\beta \varphi$,
- $(\sigma, C, I) \models_\beta \neg\Phi$, *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 4. Wir wollen nun einige Beispiele für Schrittformeln geben, die wir anhand des Ablaufes σ_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 Φ ist die Menge der endlichen Abläufe σ, für die gilt: $\sigma \models \Phi$. Die* unendliche Halbordnungssemantik *einer Ablaufformel Φ ist die Menge der unendlichen Abläufe σ, für die gilt: $\sigma \models \Phi$.* ⌟

2.5 Weitere Definitionen

Wie in Kapitel 1 beschrieben, können Systeme durch temporallogische Formeln, z. B. Ablaufformeln, spezifiziert werden. Im Rest dieser Arbeit verstehen wir unter einer Spezifikation demnach eine Formel, deren Abläufe Modelle des Verhaltens des spezifizierten Systemes sind. Neben dieser Halbordnungssemantik geben wir im nächsten Kapitel eine formale Fundierung für eine Interleavingsemantik an, sodass das Verhalten eines spezifizierten Systemes auch an seinem Transitionssystem untersucht werden kann.

Dazu definieren wir im Rest dieses Kapitels einige Konzepte, die wir in den nächsten Kapiteln benötigen. Zunächst betrachten wir einen speziellen Aufbau von Spezifikationen, die sog. Normalform:

Definition 22 (Normalform)
Eine Spezifikation Φ *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 in dieser Arbeit der Aufbau eines Transitionssystemes, nicht jedoch die Überprüfung von Systemeigenschaften betrachtet wird, werden die spezifizierten Lebendigkeitseigenschaften im Folgenden ignorieren.

Beispiel 5. Die Formel $\Phi_1 \triangleq \mathit{Init} \wedge \Box \mathit{Next}$ 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 σ_1 (s. Abb. 2.1) gilt: $\sigma_1 \models \Phi_1$. ⌟

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* Φ *ist die (endliche) Menge aller Variablen, die in* Φ *vorkommen:*

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

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

Beispiel 6. Für die Spezifikation Φ_1 aus Beispiel 5 gilt $V(\Phi_1) = \{x, y\}$. ⌟

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 σ auf $\mathcal{A}$ *genau dann, wenn*

i. $H_{|\mathcal{A}}$ ist die Restriktion von H auf $\mathcal{A}$,

ii. $T_{|\mathcal{A}} \triangleq \{t_{|\mathcal{B}} \mid \emptyset \neq \mathcal{B} = \mathcal{A} \cap dom(t)$ für ein $t \in T\}$ ist eine Transitionenmenge,

iii. $\sigma_{|\mathcal{A}}$ ist ein Ablauf, d. h. erfüllt die Bedingungen i–iii aus Def. 4.

⌟

Beispiel 7. Abbildung 2.3 zeigt die Restriktion von σ_1 auf $\{x\}$.

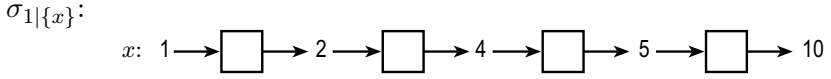


Abbildung 2.3: Restriktion von σ_1 auf $\{x\}$.

⌟

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 Φ heißt umgebungsinvariant*, wenn für alle Abläufe σ, σ^* 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 und eine Halbordnungssemantik definiert haben, werden wir uns in diesem Kapitel mit einer Interleavingsemantik beschäftigen. In [Loh05] wurden verschiedene Interleavingsemantiken für TLDA diskutiert, eine schrittbasierte Interleavingsemantik entwickelt und prototypisch implementiert, jedoch ohne den Zusammenhang zur Halbordnungssemantik formal zu beweisen. Dies werden wir in diesem Kapitel nachholen.

Dazu beschreiben wir in Abschnitt 3.1 zunächst den Unterschied zwischen Halbordnungs- und Interleavingsemantiken. Anschließend wiederholen wir kurz den Zustands- und Aktionsbegriff der schrittbasierten Interleavingsemantik, den wir in Abschnitt 3.3 mit der Definition des Transitionssystemes zu einer Spezifikation formalisieren. In Abschnitt 3.4 beweisen wir, dass für umgebungsinvariante Spezifikationen ein starker Zusammenhang zwischen Abläufen und Pfaden im Transitionssystem (Interleavings) besteht, den wir schließlich in Abschnitt 3.5 festhalten und die schrittbasierte Interleavingsemantik – nun formal fundiert – definieren.

3.1 Halbordnung, Nebenläufigkeit und Interleavings

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.

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

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 σ_2 in Abb. 3.2), als auch nebenläufige Transitionen, die nicht in der $\prec$-Relation stehen (vgl. Ablauf σ_3 in Abb. 3.3).

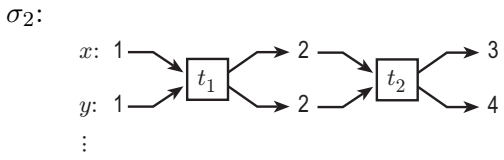


Abbildung 3.2: Ablauf σ_2 mit geordneten Transitionen: $t_1 \prec t_2$.

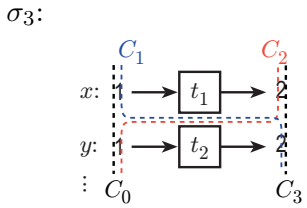


Abbildung 3.3: Ablauf σ_3 mit nebenläufigen Transitionen: $t_1 \not\prec t_2$ und $t_2 \not\prec t_1$.

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.

$$
\begin{array}{l}
\begin{matrix} x: \\ y: \end{matrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \xrightarrow{\alpha_1} \begin{bmatrix} 2 \\ 1 \end{bmatrix} \xrightarrow{\alpha_2} \begin{bmatrix} 2 \\ 2 \end{bmatrix} \\[2ex]
\begin{matrix} x: \\ y: \end{matrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \xrightarrow{\alpha_2} \begin{bmatrix} 1 \\ 2 \end{bmatrix} \xrightarrow{\alpha_1} \begin{bmatrix} 2 \\ 2 \end{bmatrix} \\[2ex]
\qquad \begin{matrix} x: \\ y: \end{matrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \xrightarrow{\alpha_3} \begin{bmatrix} 2 \\ 2 \end{bmatrix}
\end{array}
$$

Abbildung 3.4: Mögliche Interleavings von Ablauf σ_3.

Beispiel 8. Abbildung 3.4 zeigt mögliche Interleavings von Ablauf σ_3. Dabei wird die Nebenläufigkeit der Transitionen t_1 und t_2 aufgelöst, sodass für jede möglich Reihenfolge

der Transitionen t_1 und t_2 (t_1 schaltet vor t_2, t_2 schaltet vor t_1 und t_1 und t_2 schalten gleichzeitig) ein Interleaving existiert. Die Zustände $\left[\begin{smallmatrix}1\\2\end{smallmatrix}\right]$ und $\left[\begin{smallmatrix}2\\1\end{smallmatrix}\right]$ sind die beschriebenen Zwischenzustände. α_1–α_3 sind Aktionen der zugrunde liegenden Spezifikation Φ, auf die wir an dieser Stelle nicht weiter eingehen. ⌟

Trotz der beschriebenen Probleme haben Interleavingsemantiken jedoch 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. Eine Interleavingsemantik für die Logik TLDA ist somit die Grundlage für die Entwicklung eines expliziten Modelcheckers für die Verifikation von Systemen, die mit TLDA-Formeln spezifiziert worden sind.

3.2 Zustände und Aktionen in TLDA

Um eine Interleavingsemantik für die Logik TLDA anzugeben, muss definiert werden, wie aus TLDA-Spezifikationen – bzw. deren Abläufen – Interleavings abgeleitet werden können. In [Loh05] wurde eine schrittbasierte Interleavingsemantik mit folgendem Zustands- und Aktionsbegriff definiert:

- *Schnitte sind Zustände.* Ein Schnitt ordnet jeder Variablen einen Index in der Historie der jeweiligen Variablen zu. Wird der Zustand nur von der (endlichen) Menge der Systemvariablen $V(\Phi)$ einer Spezifikation $\Phi \triangleq \mathit{Init} \wedge \Box \mathit{Next}$ in Normalform beschrieben, legt $H_x(C(x))$ für alle $x \in V(\Phi)$ das durch den Schnitt C erzeugte Wertetupel fest. Die Wertetupel der Anfangsschnitte der Abläufe von Φ können dabei aus der Zustandsformel *Init* abgeleitet werden.
- *Schritte sind Aktionen.* Um Interleavings berechnen zu können, stehen nicht die Abläufe selbst, sondern nur die zugrunde liegende Spezifikation zur Verfügung. In jedem Ablauf, der Φ erfüllt, muss, beim Anfangsschnitt beginnend, jeder Schritt die Schrittformel *Next* erfüllen. Diese Schritte werden als Aktionen betrachtet.

  Um die 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. Die Menge aller Klauseln der disjunktiven Normalform von *Next* sei die Menge der Aktionen.

Es konnte gezeigt werden, dass die schrittbasierte Interleavingsemantik – im Gegensatz zur transitionsbasierten Interleavingsemantik [AR03] – implementiert werden kann. Im folgenden Abschnitt beschreiben wir einen Konstruktionsansatz, der für eine Spezifikation in Normalform das Transitionssystem berechnet.

3.3 Transitionssysteme für Spezifikationen

Aufbauend auf dem beschriebenen Zustands- und Aktionsbegriff kann nun ein Transitionssystem konstruiert werden, ein gerichteter Graph, dessen Knoten Zuständen, dessen Kanten Aktionen und dessen Pfade Interleavings 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 Φ das Transitionssystem $\mathcal{TS}$ über $V(\Phi)$ und $\mathcal{V}al$ konstruieren können:

Definition 29 (Konstruktion eines Transitionssystemes)
Sei $\Phi = Init \wedge \Box Next$ eine Spezifikation in Normalform. Zu Φ 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 Transitionssystem $\mathcal{TS}_\Phi$ wird nun induktiv über den Schritten der Abläufe von Φ konstruiert. Sei $\sigma = (H, T)$ mit $\sigma \models \Phi$ ein beliebiger Ablauf mit Anfangsschnitt C_0.

IA: 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: Seien C ein Schnitt in σ, der durch Schritte vom Anfangsschnitt C_0 in σ aus erreichbar ist und $S_C = (C, C', T_C)$ der bei C beginnende Schritt in σ. 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 Nachfolgeschnitt 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 σ. 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 Φ durchgeführt. ⌟

Anmerkung: Da es im Allgemeinen unendlich viele Abläufe σ mit $\sigma \models \Phi$ gibt, hat $\mathcal{TS}_\Phi$ in der Regel unendlich viele Interleavings und Zustände. Für widersprüchliche Spezifikationen ist $S = S_0 = R = \emptyset$, d. h. das Transitionssystem hat weder Zustände noch Zustandsübergänge.

3.4 Eigenschaften des Transitionssystemes

Wir wollen nun den Zusammenhang zwischen den Abläufen einer Spezifikation Φ und dessen Transitionssystem $\mathcal{TS}_\Phi$ betrachten. Dazu folgt aus der Konstruktion des Transitionssystemes einer Spezifikation (Def. 29) zunächst die folgende Beobachtung:

Korollar 3.1 (Abläufe und Interleavings)
Seien $\Phi = Init \wedge \Box Next$ eine Spezifikation und $\mathcal{TS}_\Phi = (S, S_0, Act, R, L)$ das Transitionssystem von Φ. Sei $\sigma = (H, T)$ mit $\sigma \models \Phi$ ein beliebiger Ablauf von Φ mit Anfangsschnitt C_0 und Schnitten C_1, C_2, C_3, ..., sodass $S_{C_i} = (C_i, C_{i+1}, T_{C_i})$ für $i = 0, 1, \ldots$ Schritte in σ sind. Sei $A_i \subseteq Act$ die Menge aller von S_{C_i} erfüllten Klauseln der disjunktiven Normalform von Next. Dann gilt:

i. In $\mathcal{TS}_\Phi$ existiert ein Anfangszustand s_0 und Zustände $s_1, s_2, s_3 \ldots$ mit den Beschriftungen $L(s_i)(x) = H_x(C_i(x))$ für alle $x \in V(\Phi)$ und $i = 0, 1, \ldots$

ii. Für alle $\alpha_i \in A_i$ existiert in $\mathcal{TS}_\Phi$ ein Zustandsübergang $s_i \xrightarrow{\alpha_i} s_{i+1}$.

⌟

Für einen Ablauf σ von Φ gibt es in $\mathcal{TS}_\Phi$ also mehrere Interleavings, deren Zustände die Schnitte von σ beschreiben, die durch Schritte vom Anfangsschnitt C_0 in σ aus erreichbar sind. Für den Ablauf σ_3 (vgl. Abb. 3.3) enthält $\mathcal{TS}_\Phi$ jedoch nur das Interleaving

$$\begin{matrix} x: \\ y: \end{matrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \xrightarrow{\alpha_3} \begin{bmatrix} 2 \\ 2 \end{bmatrix},$$

da die Schnitte C_1 und C_2 nicht durch Schritte vom Anfangsschnitt C_0 in σ_3 aus erreichbar sind. Allerdings gilt für umgebungsinvariante Spezifikationen:

Satz 3.2 (Jeder Schnitt ist durch Schritte erreichbar)
Sei Φ eine umgebungsinvariante Spezifikation. Sei C^* ein Schnitt in einem Ablauf σ mit $\sigma \models \Phi$. Dann existiert ein Ablauf σ_E mit $\sigma_E \models \Phi$, sodass Folgendes gilt:

1. σ_E stimmt auf den Systemvariablen von Φ mit σ überein, d. h. $\sigma_{E|V(\Phi)} = \sigma_{|V(\Phi)}$,

2. *es gibt ein Schnitt C_E in σ_E, sodass:*
   a) *C_E stimmt auf den Indizes der Systemvariablen von Φ mit C^* überein, d. h. $C_{E|V(\Phi)} = C^*_{|V(\Phi)}$,*
   b) *C_E ist in σ_E vom Anfangsschnitt aus durch Schritte erreichbar.*

⌟

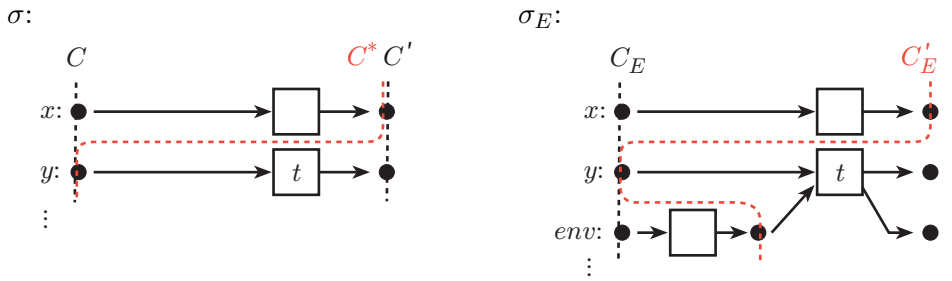
Um den Satz zu beweisen, benutzen wir folgendes Lemma aus [Ale05]:

Lemma 3.3 (Lemma 3.4 aus [Ale05])
Sei $\mathcal{A} \subset Var$ eine endliche Variablenmenge. Seien σ ein Ablauf, C ein Schnitt in σ und C^* ein nachfolgender Schnitt von C mit $C \leq C^* \leq C'$. Es gibt einen Ablauf σ_E, sodass Folgendes gilt:

1. *σ_E stimmt auf den Variablen aus $\mathcal{A}$ mit σ überein, d. h. $\sigma_{E|\mathcal{A}} = \sigma_{|\mathcal{A}}$,*
2. *es gibt einen Schnitt C_E in σ_E, sodass:*
   a) *C_E stimmt auf den Indizes der Variablen aus $\mathcal{A}$ mit C überein, d. h. $C_{E|\mathcal{A}} = C_{|\mathcal{A}}$,*
   b) *der Nachfolgeschnitt C'_E von C_E in σ_E stimmt auf den Indizes der Variablen aus $\mathcal{A}$ mit C^* überein, d. h. $C'_{E|\mathcal{A}} = C^*_{|\mathcal{A}}$.*

⌟

Wir werden an dieser Stelle nicht den Beweis von Lemma 3.3 wiedergeben, sondern lediglich die Beweisidee illustrieren (s. Abb. 3.5).



(a) Ablauf σ mit Schnitten $C \leq C^* \leq C'$.

(b) Ablauf σ_E mit Schnitten C_E und C'_E.

Abbildung 3.5: Illustration zur Beweisidee von Lemma 3.3: Sei $\mathcal{A} = \{x, y\}$. In σ gilt $C'(y) \neq C^*(y)$. Durch geeignete Synchronisation mit der Umgebungsvariablen *env* wird in σ_E das Schalten von Transition t aufgeschoben, sodass C'_E auf den Variablen aus $\mathcal{A}$ mit C^* übereinstimmt.

Die in Lemma 3.3 beschriebene Situation kann als Sonderfall von Satz 3.2 angesehen werden: Der in Satz 3.2 gesuchte Ablauf σ_E kann durch wiederholte Synchronisation mit Umgebungsvariablen konstruiert werden.

Beweis von Satz 3.2. Seien Φ eine umgebungsinvariante Spezifikation, σ ein Ablauf von Φ und C^* ein Schnitt in σ. Wir unterscheiden zwei Fälle:

1. Fall: Der Schnitt C^* ist durch Schritte vom Anfangsschnitt in σ aus erreichbar. Es ist nichts zu zeigen: $\sigma_E = \sigma$ und $C_E = C^*$ leisten das Verlangte.

2. Fall: Der Schnitt C^* ist in σ nicht durch Schritte vom Anfangsschnitt in σ aus erreichbar.

   Durch folgende Fixpunktoperation kann ein Ablauf σ_E konstruiert werden, der das Verlangte leistet:

   Sei $\sigma' := \sigma$. Wiederhole bis $\mathcal{A} = V(\Phi)$:

   1. Nach Lemma 2.1 existiert in σ' ein Schnitt C, der durch Schritte vom Anfangsschnitt aus erreichbar ist mit $C(x) \leq C^*(x)$ für alle $x \in V(\Phi)$ und $C(x) = C^*(x)$ für einige $x \in V(\Phi)$.
   2. Sei $\mathcal{A}$ die Menge aller Systemvariablen $x \in V(\Phi)$ mit $C(x) = C^*(x)$. Nach Lemma 3.3 gibt es einen Ablauf σ_E mit einem Schnitt C_E, sodass $\sigma_{E|\mathcal{A}} = \sigma'_{|\mathcal{A}}$, $C_{E|\mathcal{A}} = C_{|\mathcal{A}}$ und $C'_{E|\mathcal{A}} = C^*_{|\mathcal{A}}$.

      Dabei gilt für alle $x \in V(\Phi)$: $C'_E(x) = \begin{cases} C^*(x), & \text{falls } x \in \mathcal{A}, \\ C'(x), & \text{sonst.} \end{cases}$
   3. Sei für den nächsten Durchlauf $\sigma' := \sigma_E$.

   Die Konstruktion terminiert, da in jedem Durchlauf neue Systemvariablen zu $\mathcal{A}$ hinzugefügt werden, die Menge der Systemvariablen $V(\Phi)$ jedoch stets endlich ist.

   Für den im letzten Durchlauf konstruierten Ablauf σ_E mit Schnitt C_E gilt: $\sigma_{E|V(\Phi)} = \sigma_{|V(\Phi)}$ und $C_{E|V(\Phi)} = C_{E|\mathcal{A}} = C^*_{|\mathcal{A}} = C^*_{|V(\Phi)}$. Dabei ist C_E in σ_E durch Schritte vom Anfangsschnitt aus erreichbar. Da Φ umgebungsinvariant ist, gilt außerdem $\sigma_E \models \Phi$.

□

Beispiel 9. Der im Beweis von Satz 3.2 angegebene Konstruktionsansatz ist in Abbildung 3.6–3.8 illustriert.

Seien Φ eine umgebungsinvariante Spezifikation mit Systemvariablen $V(\Phi) = \{x, y, z\}$ und σ ein Ablauf von Φ. Seien C^* ein beliebiger Schnitt in σ und C der Schnitt in σ mit den in Lemma 2.1 beschriebenen Eigenschaften.

Abbildung 3.6 zeigt die Restriktion von σ auf die Systemvariablen $V(\Phi)$. Es gilt $C(z) = C^*(z)$ und somit $\mathcal{A} = \{z\}$.

Abbildung 3.7 zeigt die Situation nach dem 1. Durchlauf: Im konstruierten Ablauf σ_E mit Schnitten C_E und C^* gilt $C_E(z) = C^*(z)$, $C_E(y) = C^*(y)$ und somit $\mathcal{A} = \{y, z\}$.

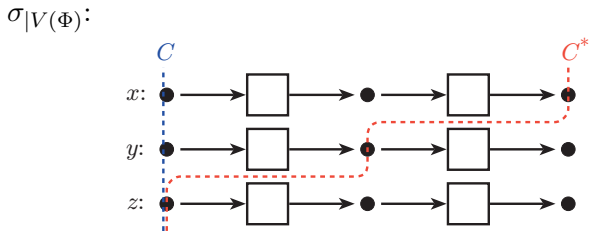


Abbildung 3.6: Restriktion des Ablaufes σ auf die Systemvariablen $V(\Phi)$.

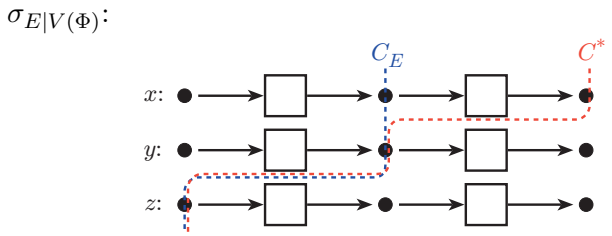


Abbildung 3.7: Situation nach dem 1. Durchlauf.

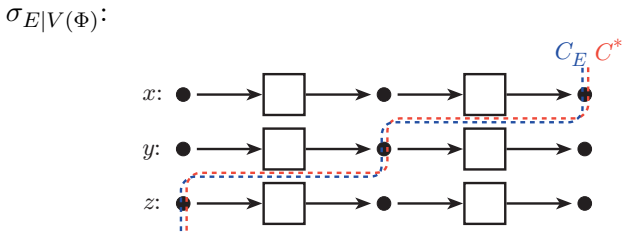


Abbildung 3.8: Situation nach dem 2. Durchlauf, Fixpunkt erreicht.

Abbildung 3.8 zeigt die Situation nach dem 2. Durchlauf: Im konstruierten Ablauf σ_E mit Schnitten C_E und C^* gilt $C_E(z) = C^*(z)$, $C_E(y) = C^*(y)$ und $C_E(x) = C^*(x)$. Somit gilt $\mathcal{A} = \{x, y, z\} = V(\Phi)$, sodass der Ablauf σ_E das Verlangte leistet. Es gilt: $C_{E|V(\Phi)} = C^*_{|V(\Phi)}$, und nach Lemma 3.3 ist C_E in σ_E durch Schritte vom Anfangsschnitt aus erreichbar. Außerdem gilt $\sigma_E \models \Phi$. ⌟

Dieser Satz ist für die konstruierten Transitionssysteme von entscheidender Bedeutung: Während in Korollar 3.1 nur Schnitte betrachtet wurden, die vom Anfangsschnitt eines Ablaufes aus durch Schritte erreichbar sind, kann mit Satz 3.2 gezeigt werden, dass zu *jedem* Schnitt in einem Ablauf ein Zustand mit der entsprechenden Beschriftung im Transitionssystem existiert.

Korollar 3.4 (Schnitte entsprechen Zuständen)
Seien Φ eine umgebungsinvariante Spezifikation und $\mathcal{TS}_\Phi = (S, S_0, Act, R, L)$ das Tran-

sitionssystem zu Φ. Für jeden Ablauf $\sigma = (H, T)$ mit $\sigma \models \Phi$ existiert zu jedem Schnitt C in σ ein Zustand $s_C \in S$ mit $L(s_C)(x) = H_x(C(x))$ für alle $x \in V(\Phi)$. ⌟

Aus Korollar 3.1 folgt mit Satz 3.2 und Korollar 3.4 außerdem:

Korollar 3.5 (Schritte entsprechen Zustandsübergängen)
Seien $\Phi = Init \wedge \Box Next$ eine umgebungsinvariante Spezifikation, $\sigma = (H, T)$ mit $\sigma \models \Phi$ ein beliebiger Ablauf von Φ und $S_C = (C, C', T_C)$ ein beliebiger Schritt in σ sowie A die Menge aller von S_C erfüllten Klauseln der disjunktiven Normalform der Schrittformel Next. Des Weiteren seien $\mathcal{TS}_\Phi = (S, S_0, Act, R, L)$ das Transitionssystem zu Φ und $s_C, s_{C'} \in S$ Zustände mit den Beschriftungen $L(s_C)(x) = H_x(C(x))$ und $L(s_{C'})(x) = H_x(C'(x))$ für alle $x \in V(\Phi)$. Dann existiert für alle $\alpha \in A$ im Transitionssystem $\mathcal{TS}_\Phi$ ein Zustandsübergang $s_C \xrightarrow{\alpha} s_{C'}$. ⌟

Nach Satz 3.2 existieren zum Ablauf σ_3 (vgl. Abb. 3.3) Abläufe σ_{E_1} bzw. σ_{E_2}, bei denen die Schnitte C_1 bzw. C_2 durch Schritte vom Anfangsschnitt C_0 aus erreichbar sind, sodass nach Korollar 3.5 auch die Interleavings

$$\begin{matrix} x: \\ y: \end{matrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \xrightarrow{\alpha_1} \begin{bmatrix} 2 \\ 1 \end{bmatrix} \xrightarrow{\alpha_2} \begin{bmatrix} 2 \\ 2 \end{bmatrix} \qquad \text{und} \qquad \begin{matrix} x: \\ y: \end{matrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \xrightarrow{\alpha_2} \begin{bmatrix} 1 \\ 2 \end{bmatrix} \xrightarrow{\alpha_1} \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

im Transitionssystem der zugrunde liegenden Spezifikation von σ_3 enthalten sind.

Damit ist der Zusammenhang zwischen den Abläufen einer Spezifikation Φ und den Interleavings im Transitionssystem $\mathcal{TS}_\Phi$ beschrieben. Informal lässt sich festhalten, dass die Schritte eines jeden Ablaufes σ mit $\sigma \models \Phi$ mehrere Interleavings in $\mathcal{TS}_\Phi$ beschreiben. Betrachten wir nun die umgekehrte Richtung, also vom Transitionssystem $\mathcal{TS}_\Phi$ zu den Abläufen von Φ:

Korollar 3.6 (Interleavings entsprechen Abläufen)
Sei $\mathcal{TS}_\Phi = (S, S_0, Act, R, L)$ ein Transitionssystem zu einer Spezifikation Φ. Sei $s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} s_2 \xrightarrow{\alpha_2} s_3 \ldots$ mit $s_0 \in S_0$ ein beliebiges Interleaving in $\mathcal{TS}_\Phi$.

Dann existiert ein Ablauf $\sigma = (H, T)$ mit $\sigma \models \Phi$, Anfangsschnitt C_0, Schnitten C_1, C_2, $C_3, \ldots$ und den Schritten $S_{C_i} = (C_i, C_{i+1}, T_{C_i})$ mit $H_x(C_i(x)) = L(s_i)(x)$ für alle $x \in V(\Phi)$ und $S_{C_i} \Vdash \alpha_i$ $(i = 0, 1, \ldots)$. ⌟

Die Existenz dieses Ablaufes σ folgt direkt aus der Konstruktionsvorschrift des Transitionssystemes zur Spezifikation Φ (Def. 29), in der wir zu jedem Ablauf mehrere Interleavings zu $\mathcal{TS}_\Phi$ hinzugefügt haben. Insbesondere lässt sich jedes Interleaving in $\mathcal{TS}_\Phi$ genau einem Ablauf von Φ – bzw. dessen Schritten – zuordnen.

Damit ist der Zusammenhang zwischen den Zuständen im Transitionssystem $\mathcal{TS}_\Phi$ einer Spezifikation Φ und den Abläufen von Φ beschrieben: Informal ausgedrückt beschreibt jeder Pfad, d. h. jedes Interleaving in $\mathcal{TS}_\Phi$ die Schritte eines Ablaufes σ mit $\sigma \models \Phi$.

Für eine umgebungsinvariante Spezifikation Φ erlaubt der in den Korollaren 3.4 und 3.5 beschriebene Zusammenhang zusammen mit Korollar 3.6, das Transitionssystem $\mathcal{TS}_\Phi$ analog zur Menge der Abläufe von Φ (vgl. Def. 21) als semantisches Modell anzusehen.

3.5 Interleavingsemantik

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

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

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

Wir können nun Systeme durch eine (widerspruchsfreie) umgebungsinvariante Spezifikation Φ in Normalform modellieren und ihr Verhalten sowohl durch die Menge der Abläufe von Φ, 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, und es wegen des beliebigen Verhaltens der Umgebung stets unendlich viele Abläufe gibt, die Φ erfüllen. 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 $\mathcal{TS}_\Phi = (S, S_0, Act, R, L)$ ein Transitionssystem einer umgebungsinvarianten Spezifikation Φ. Zwei Zustände $s_1, s_2 \in S$ sind genau dann äquivalent*, wenn ihre Beschriftungen gleich sind: $s_1 \sim s_2$ gdw. $L(s_1) = L(s_2)$.* ⌟

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

i. $S/\sim \subseteq S$ enthält aus jeder Äquivalenzklasse von S genau einen Repräsentanten.

ii. $S_0/\sim \subseteq S_0$ enthält aus jeder Äquivalenzklasse von S_0 genau einen Repräsentanten.

iii. $R/\sim \triangleq \{(s_1, \alpha, s_2) \mid s_1, s_2 \in S/\sim, (s_1', \alpha, s_2') \in R, s_1 \sim s_1', s_2 \sim s_2'\} \subseteq R$ ist eine Zustandsübergangsrelation.

⌟

Anmerkung: Das reduzierte Transitionssystem $\mathcal{TS}_\Phi/\sim$ hat für eine Spezifikation mit endlichen Wertebereichen der Systemvariablen $V(\Phi)$ stets endlich viele Zustände. Da

explizite Modelchecking-Algorithmen größtenteils auf Tiefensuche basieren, ist dies eine essentielle Eigenschaft.

Durch die beschriebene Reduktion ist der in Korollar 3.6 beschriebene Zusammenhang zwischen den Zuständen im reduzierten Transitionssystem $\mathcal{TS}_\Phi/\sim$ und den Abläufen von Φ nicht mehr eindeutig: Während zuvor jeder Zustand eindeutig einem Schnitt in einem Ablauf zuzuordnen war, enthält die Zustandsmenge des reduzierten Transitionssystems nun für *alle* Schnitte C, für die das Tupel $\{x \mapsto H_x(C(x)) \mid x \in V(\Phi)\}$ gleich ist, genau *einen* Repräsentanten s mit der Beschriftung $L(s)(x) = H_x(C(x))$ für alle $x \in V(\Phi)$.

Ebenso verhält es sich mit den Schritten in den Abläufen von Φ: Die Menge *aller* Schritte $S_C = (C, C', T_C)$ in den Abläufen von Φ, bei denen das Tupel $\{x \mapsto H_x(C(x)) \mid x \in V(\Phi)\}$ bzw. $\{x \mapsto H_x(C'(x)) \mid x \in V(\Phi)\}$ gleich ist und für die $S_C \Vdash \alpha$ gilt, werden im reduzierten Transitionssystem durch *genau einen* Zustandsübergang $s \xrightarrow{\alpha} s'$ mit den Zustandsbeschriftungen $L(s)(x) = H_x(C(x))$ und $L(s')(x) = H_x(C'(x))$ repräsentiert.

Da wir jedoch vorausgesetzt haben, dass es sich bei Φ um eine umgebungsinvariante Spezifikation handelt, gilt die in Korollar 3.6 beschriebene Beobachtung auch für reduzierte Transitionssysteme:

Lemma 3.7 (Interleavings in $\mathcal{TS}_\Phi/\sim$ entsprechen Abläufen)
Seien Φ eine umgebungsinvariante Spezifikation und $\mathcal{TS}_\Phi/\sim = (S/\sim, S_0/\sim, Act, R/\sim, L)$ das reduzierte Transitionssystem zu Φ. Sei $s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} s_2 \xrightarrow{\alpha_2} s_3 \ldots$ mit $s_0 \in S_0/\sim$ ein beliebiges Interleaving in $\mathcal{TS}_\Phi/\sim$.

Dann existiert ein Ablauf $\sigma = (H, T)$ mit $\sigma \models \Phi$, Anfangsschnitt C_0, Schnitten C_1, C_2, C_3,... und den Schritten $S_{C_i} = (C_i, C_{i+1}, T_{C_i})$ mit $H_x(C_i(x)) = L(s_i)(x)$ für alle $x \in V(\Phi)$ und $S_{C_i} \Vdash \alpha_i$ ($i = 0, 1, \ldots$). ⌟

Wir geben an dieser Stelle nicht den Beweis an, sondern motivieren lediglich die Beweisidee. Ein Ablauf, der die Eigenschaften aus Lemma 3.7 erfüllt, kann anhand des gegebenen Interleavings konstruiert werden:

- Da Φ umgebungsinvariant ist, hängt der Wahrheitswert der Schritte eines Ablaufes von Φ nur von den Belegungen der Systemvariablen ab. Diese Belegungen können aus den Beschriftungen der Zustände des Interleavings abgeleitet werden.
- Weiterhin geben wir in Abschnitt 4.2 ein Verfahren an, mit dem, gegeben eine Aktion von Φ, Informationen über die $\sim$-Variablen von Φ gewonnen werden können.

Die Informationen über die Werte und die Involviertheit der Systemvariablen von Φ reichen aus, um eine Historie H und eine Transitionsmenge T zu konstruieren, sodass der Ablauf $\sigma = (H, T)$ das Verlangte leistet.

4 Automatische Konstruktion von Transitionssystemen

Die in Kapitel 3 definierte Interleavingsemantik wurde in einem Brute-Force-Prototypen implementiert. Der entstandene Prototyp ist in der Lage, zu einfachen TLDA-Spezifikationen das Transitionssystem aufzubauen. Der Fokus dieser ersten Implementierung lag allerdings nicht in der Entwicklung leistungsfähiger Algorithmen und Datenstrukturen, sondern vielmehr darin, Erfahrungen mit der definierten Interleavingsemantik zu sammeln. In [Loh05] wurden bereits mehrere Probleme dieser ersten naiven Implementierung beschrieben:

- Die schrittbasierte Interleavingsemantik betrachtet die Schritte einer umgebungsinvarianten Spezifikation als Aktionen. Diese Aktionen lassen sich syntaktisch aus der disjunktiven Normalform der Schrittformel *Next* der Spezifikation ableiten. 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 sehr lange dauern, bzw. wegen Speicherüberlaufes nicht beendet werden.
- Von der großen Anzahl der Klauseln ist meist nur ein Bruchteil für das zu konstruierende Transitionssystem wesentlich, da sehr viele Klauseln widersprüchlich sind, daher von keinem Schritt der Abläufe der Spezifikation erfüllt werden können und somit als Aktionen ausscheiden.

  Der für die Analyse auf Widersprüchlichkeit notwendige Berechnungsaufwand bedeutet jedoch bei einer exponentiellen Klauselanzahl, dass die Rechenzeit der Informationsgewinnungsphase inakzeptabel hoch ist. Andererseits ist diese Berechnung notwendig, damit nicht zu viele widersprüchliche Aktionen „übersehen“ werden, da für jede Aktion Programmcode erzeugt und übersetzt werden muss (Speicherplatzproblem). Außerdem wird bei der späteren Konstruktion des reduzierten Transitionssystemes in jedem Zustand für jede Aktion überprüft werden, ob mit ihr ein Nachfolgezustand erreicht werden kann (Laufzeitproblem).
- 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, wodurch erneut ein Speicherplatzproblem bei der Codegenerierung auftritt.

In diesem Kapitel wollen wir Lösungen für diese Probleme vorstellen, durch die die Effizienz der automatischen Konstruktion des Transitionssystemes, bzw. der Analyse der Aktionen einer Spezifikation, erhöht werden kann. Dazu untersuchen wir in Abschnitt 4.1

den speziellen syntaktischen Aufbau komponierter Spezifikationen, der es ermöglicht, eine spezielle Datenstruktur für komponierte Spezifikationen, den DNF-Graphen, zu entwickeln. Mit ihm können Klauseln des spezifizierten Systemes mit geringerem Speicher- und Zeitaufwand berechnet werden, ohne die gesamte Spezifikation in disjunktive Normalform zu überführen. Weiterhin erlauben DNF-Graphen ein on-the-fly-Verfahren für die Berechnung der widerspruchsfreien Aktionen einer Spezifikation zu definieren. In Abschnitt 4.2 beschreiben wir $\sim$-Graphen, eine Datenstruktur, mit der Informationen über die $\sim$-Variablen einer Spezifikation effizient berechnet und minimal repräsentiert sowie aufgrund von $\sim$-Variablen widersprüchliche Klauseln erkannt werden können. Zuletzt fassen wir in Abschnitt 4.3 weitere Ansätze zusammen, mit denen die vorhandenen Algorithmen ergänzt und verbessert werden können.

Der bestehende Prototyp wurde (mit Ausnahme der in Abschnitt 4.3.5 beschriebenen Heuristik) mit den beschriebenen Algorithmen und Datenstrukturen erweitert. Zu einigen Algorithmen geben wir an entsprechender Stelle Leistungswerte der Implementierung an, während wir in Kapitel 5 das Zusammenspiel aller Techniken an mehreren Fallstudien untersuchen.

4.1 Der DNF-Graph

Klauseln bestehen meist aus mehreren Konjunktionsgliedern. Dadurch ergeben sich verschiedene Gründe, weswegen eine Klausel widersprüchlich sein kann: Zum einen kann sie ein widersprüchliches Konjunktionsglied (z. B. $x \neq x$ oder $x \cdot 0 = 1$) enthalten. Zum anderen kann eine Klausel widersprüchlich sein, da sie sich widersprechende Konjunktionsglieder enthält (z. B. $\widetilde{x} \wedge \neg \widetilde{x}$) – in diesem Falle führt erst die Konjunktion dieser Glieder zum Widerspruch. Wird bei der sequentiellen Betrachtung der Konjunktionsglieder einer Klausel einer dieser Fälle erkannt, ist diese Klausel – unabhängig von den restlichen Konjunktionsgliedern – in jedem Falle widersprüchlich. Bei der computergestützten Berechnung der Aktionen kann in diesem Fall die betrachtete Klausel verworfen und direkt mit der Überprüfung der nächsten Klausel fortgesetzt werden.

Wir stellen in diesem Abschnitt ein so genanntes on-the-fly-Verfahren für die Überprüfung von Klauseln auf Widersprüchlichkeit vor. Dazu nutzen wir einen speziellen syntaktischen Aufbau von Spezifikationen verteilter Systeme aus, den wir in Abschnitt 4.1.1 beschreiben. Anschließend definieren wir in Abschnitt 4.1.2 mit DNF-Graphen eine Datenstruktur, die es ermöglicht, Spezifikationen sehr platzsparend abzuspeichern und gleichzeitig die Aktionen des spezifizierten Systemes effizienter zu berechnen. In Abschnitt 4.1.3 zeigen wir, wie widersprüchliche Klauseln frühzeitig entdeckt und verworfen werden können und beschreiben das on-the-fly-Verfahren für die Berechnung widerspruchsfreier Aktionen.

4.1.1 Komposition von Spezifikationen

Die Temporal Logic of Distributed Actions ist eine kompositionale Logik. Dies bedeutet, dass die parallele Komposition nebenläufiger Komponenten durch Konjunktion der Spezifikationen der einzelnen Komponenten ausgedrückt werden kann: „Komposition ist Konjunktion.“ [Pnu79]. Diese Eigenschaft vereinfacht die Spezifikation verteilter Systeme, die meist aus mehreren Komponenten bestehen. Da eine reine parallele Komposition in der Regel nicht das gewünschte Verhalten beschreibt, müssen die Komponenten meist synchronisiert werden. Diese Synchronisierung wird durch zusätzliche Komponenten beschrieben, die ebenfalls mit dem Gesamtsystem komponiert werden. Insgesamt besteht die Spezifikation eines verteilten Systemes also aus der Konjunktion der Spezifikationen seiner Komponenten und geeigneten Synchronisationskomponenten.

Wir wollen den kompositionalen Charakter von TLDA anhand eines typischen Beispiels, der Stunden-/Minutenuhr (vgl. [Ale05]), beschreiben. Dieses Beispiel bildet die Grundlage für die folgenden Beispiele 11–14 in diesem Abschnitt.

Beispiel 10. Eine Stunden-/Minutenuhr kann als komponiertes System beschrieben werden, das aus einer Stundenuhr und einer Minutenuhr besteht.

Dabei spezifiziert $\Phi_{\text{hr}} \triangleq \mathit{Init}_{\text{hr}} \wedge \Box \mathit{Next}_{\text{hr}}$ mit

$$\mathit{Init}_{\text{hr}} \triangleq hr \in \{0, \ldots, 23\}, \qquad \mathit{Next}_{\text{hr}} \triangleq \Box \left(\widetilde{hr} \Rightarrow (hr' = (hr + 1) \bmod 24) \right)$$

eine Stundenuhr und $\Phi_{\text{min}} \triangleq \mathit{Init}_{\text{min}} \wedge \Box \mathit{Next}_{\text{min}}$ mit

$$\mathit{Init}_{\text{min}} \triangleq min \in \{0, \ldots, 59\}, \qquad \mathit{Next}_{\text{min}} \triangleq \Box \left(\widetilde{min} \Rightarrow (min' = (min + 1) \bmod 60) \right)$$

eine Minutenuhr. Eine parallele Komposition der beiden Spezifikationen, also

$$\Phi^*_{\text{hrmin}} \triangleq \mathit{Init}_{\text{hr}} \wedge \mathit{Init}_{\text{min}} \wedge \Box \mathit{Next}_{\text{hr}} \wedge \Box \mathit{Next}_{\text{min}},$$

würde jedoch nicht das gewünschte Verhalten beschreiben, da die Aktualisierung der Variable *hr* nicht unabhängig von der Aktualisierung der Variable *min* ist (vgl. Ablauf σ_4 in Abb. 4.1). Um dieses unerwünschte Verhalten auszuschließen, muss eine Synchronisationskomponente wie z. B. *Sync* mit der Spezifikation

$$\mathit{Sync} \triangleq \left(\widetilde{hr} \Rightarrow (\widetilde{hrmin} \wedge min = 59) \right) \wedge \left((\widetilde{min} \wedge min = 59) \Rightarrow \widetilde{hr} \right)$$

hinzugefügt werden, sodass

$$\Phi_{\text{hrmin}} \triangleq \mathit{Init}_{\text{hr}} \wedge \mathit{Init}_{\text{min}} \wedge \Box \mathit{Next}_{\text{hr}} \wedge \Box \mathit{Next}_{\text{min}} \wedge \Box \mathit{Sync}$$

letztlich das gewünschte Verhalten einer Stunden-/Minutenuhr spezifiziert, z. B. Ablauf σ_5 in Abbildung 4.2.

⌟

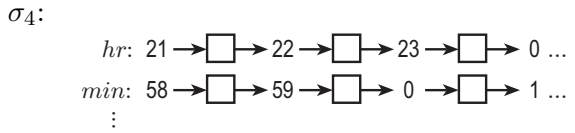


Abbildung 4.1: Ein Ablauf von Φ^*_{hrmin}: Die Variable *hr* ist nicht mit *min* synchronisiert.

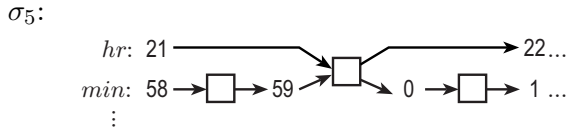


Abbildung 4.2: Ein Ablauf von Φ_{hrmin}: Die Variable *hr* ist korrekt mit *min* synchronisiert.

Wir betrachten in diesem Abschnitt nun ausschließlich Spezifikationen, die aus der Komposition mehrerer Teilspezifikationen bestehen. Syntaktisch haben diese Spezifikationen die Form

$$\Phi \triangleq \underbrace{\mathit{Init}_1 \wedge \mathit{Init}_2 \wedge \ldots \wedge \mathit{Init}_n}_{\triangleq \mathit{Init}} \wedge \Box \underbrace{(\mathit{Next}_1 \wedge \mathit{Next}_2 \wedge \ldots \wedge \mathit{Next}_n \wedge \mathit{Sync}_1 \wedge \ldots \wedge \mathit{Sync}_m)}_{\triangleq \mathit{Next}}.$$

Dabei beschreibt $\mathit{Init}_i \wedge \Box \mathit{Next}_i$ für $1 \leq i \leq n$ das Verhalten einer einzelnen Komponente i des Gesamtsystemes. Die Schrittformeln Sync_j für $1 \leq j \leq m$ spezifizieren die Synchronisation der einzelnen Komponenten. Eine Aktion des Gesamtsystemes ist nach dem Aktionsbegriff der schrittbasierten Interleavingsemantik ein Schritt des Systemes und wird durch eine Klausel der disjunktiven Normalform von *Next* mit

$$\mathit{Next} \triangleq \bigwedge_{i=1}^{n} \mathit{Next}_i \wedge \bigwedge_{j=1}^{m} \mathit{Sync}_j$$

beschrieben.

Notation: Wir beschreiben im Folgenden sowohl die Komponenten der Teilsysteme als auch die Synchronisationskomponenten mit Next_i $(1 \leq i \leq n+m)$ und vereinfachen dadurch die folgenden Definitionen.

4.1.2 DNF-Graphen als Datenstruktur

Um die Aktionen einer Spezifikation $\Phi \triangleq \mathit{Init} \wedge \Box \mathit{Next}$ analysieren zu können, muss die disjunktive Normalform der Schrittformel *Next* gebildet werden. In diesem Abschnitt definieren wir eine Datenstruktur, mit der wir den durch die Umformung verbundenen, bereits beschriebenen Problemen entgegenwirken und den kompositionalen Charakter der Spezifikationen ausnutzen: Anstelle der disjunktiven Normalform der Gesamtformel speichern wir nur die disjunktiven Normalformen der einzelnen Komponenten. Neben dem geringeren Verbrauch von Arbeitsspeicher kann auch die Rechenzeit um ein vielfaches

verringert werden. Außerdem ermöglicht diese Repräsentation ein on-the-fly-Verfahren für die Überprüfung von Aktionen auf Widersprüchlichkeit, das wir in Abschnitt 4.1.3 vorstellen.

Anmerkung: Der DNF-Graph setzt disjunktive Normalform voraus. Diese bilden wir mit der in Abschnitt 4.3.1 beschriebenen Regel.

Definition 33 (DNF-Graph, Komponentenindex, Klauselindex)
Sei $\Phi \triangleq \mathit{Init} \wedge \Box \mathit{Next}$ *eine umgebungsinvariante Spezifikation mit* $\mathit{Next} \triangleq \bigwedge_{i=1}^{n} \mathit{Next}_i$, *wobei* Next_i $(1 \leq i \leq n)$ *disjunktive Normalform hat. Sei* length_i *die Anzahl der Klauseln von* Next_i *und* k_j^i $(1 \leq j \leq \mathit{length}_i)$ *die* j*-te Klauseln in* Next_i, *d. h. Next hat die Form*

$$\mathit{Next} = \bigwedge_{i=1}^{n} \mathit{Next}_i = \bigwedge_{i=1}^{n} \bigvee_{j=1}^{\mathit{length}_i} k_j^i.$$

Der DNF-Graph für Φ *ist der gerichtete azyklische Graph* $\mathcal{D}_\Phi = (N, E)$ *mit*

i. $N = \{k_j^i \mid 1 \leq i \leq n \text{ und } 1 \leq j \leq \mathit{length}_i\} \cup \{\alpha, \omega\}$ *ist die Menge der Knoten.*

ii. $E = \{(k_j^i, k_l^{i+1}) \mid 1 \leq i \leq n-1, 1 \leq j \leq \mathit{length}_i \text{ und } 1 \leq l \leq \mathit{length}_{i+1}\} \cup \{(\alpha, k_j^1 \mid 1 \leq j \leq \mathit{length}_1)\} \cup \{(k_j^n, \omega) \mid 1 \leq j \leq \mathit{length}_n)\}$ *ist die Menge der Kanten.*

Für einen Knoten k_j^i *ist* i *der* Komponentenindex *und* j *der* Klauselindex. ⌟

Beispiel 11. Wir greifen nun wieder auf die Spezifikation der Stunden-/Minutenuhr aus Beispiel 10 zurück: $\Phi_{\text{hrmin}} \triangleq \mathit{Init} \wedge \Box \mathit{Next}$ mit

$$\begin{aligned}
\mathit{Init} \triangleq\ & (hr = 0 \wedge min = 0)\,, \\
\mathit{Next} \triangleq\ & \Big(\widetilde{hr} \Rightarrow (hr' = (hr + 1) \bmod 24)\Big) \wedge \\
& \Big(\widetilde{min} \Rightarrow (min' = (min + 1) \bmod 60)\Big) \wedge \\
& \Big(\widetilde{hr} \Rightarrow (\widetilde{hrmin} \wedge min = 59)\Big) \wedge \\
& \Big((\widetilde{min} \wedge min = 59) \Rightarrow \widetilde{hr}\Big)
\end{aligned}$$

ist eine umgebungsinvariante Spezifikation. Formt man die einzelnen Konjunktionsglieder von *Next* in disjunktive Normalform um, erhält man:

$$\begin{aligned}
\mathit{Next} \triangleq\ & \Big(\neg\widetilde{hr} \vee (\widetilde{hr} \wedge hr' = (hr + 1) \bmod 24)\Big) \wedge \\
& \Big(\neg\widetilde{min} \vee (\widetilde{min} \wedge min' = (min + 1) \bmod 60)\Big) \wedge \\
& \Big(\neg\widetilde{hr} \vee (\widetilde{hr} \wedge \widetilde{hrmin} \wedge min = 59)\Big) \wedge \\
& \Big(\neg\widetilde{min} \vee min \neq 59 \vee \widetilde{hr}\Big)
\end{aligned}$$

Es gilt: $length_1 = length_2 = length_3 = 2$ und $length_4 = 3$. Des Weiteren gilt bspw. $k_2^3 = \left(\widetilde{hr} \wedge \widetilde{hrmin} \wedge min = 59\right)$ und $k_3^4 = \widetilde{hr}$.

Abbildung 4.3 zeigt den DNF-Graphen $\mathcal{D}_{\Phi_{\text{hrmin}}}$. Die einzelnen Klauseln sind in der Spezifikationssprache TLDA$^+$ [Loh05] angegeben.

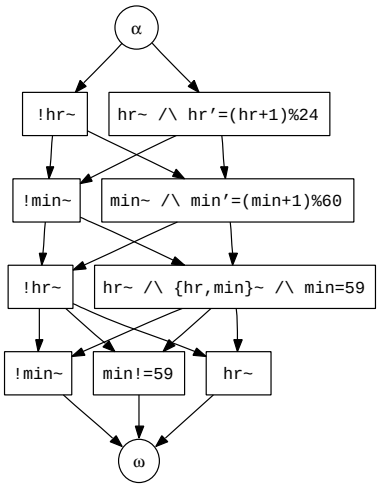


Abbildung 4.3: DNF-Graph für Spezifikation Φ_{hrmin}.

⌟

Die Knotenmenge des DNF-Graphen $\mathcal{D}_\Phi$ besteht also aus den Klauseln der disjunktiven Normalform der Konjunktionsglieder $Next_i$ von $Next$ sowie zwei weiteren Knoten α und ω. Für $1 \leq i \leq n-1$ ist jede Klausel aus $Next_i$ mit allen Klauseln aus $Next_{i+1}$ durch eine Kante verbunden. Des Weiteren ist der Knoten α mit allen Klauseln aus $Next_1$ und die Klauseln aus $Next_n$ mit dem Knoten ω verbunden. Jeder Pfad von α nach ω besucht für $1 \leq i \leq n$ genau eine Klausel aus $Next_i$. Weiterhin gibt es für jede Klausel k_j^i Pfade von α nach ω, die k_j^i enthalten.

Korollar 4.1 (Pfade und Aktionen)
Seien Φ eine umgebungsinvariante Spezifikation, $\mathcal{D}_\Phi = (N, E)$ der DNF-Graph für Φ und $w = \alpha k_{i_1}^1 k_{i_2}^2 \ldots k_{i_n}^n \omega$ ein Pfad in $\mathcal{D}_\Phi$ mit $(\alpha, k_{i_1}^1), (k_{i_1}^1, k_{i_2}^2), \ldots, (k_{i_{n-1}}^{n-1}, k_{i_n}^n), (k_{i_n}^n, \omega) \in E$ für $1 \leq i_j \leq length_j$ und $1 \leq j \leq n$.

Dann und nur dann ist $\varphi \triangleq \bigwedge_{i=j}^n k_{i_j}^j$ eine Aktion von Φ. ⌟

Beispiel 12. Abbildung 4.4 zeigt den Pfad $\alpha k_1^1 k_2^2 k_1^3 k_2^4 \omega$ im DNF-Graphen $\mathcal{D}_{\Phi_{\text{hrmin}}}$, der der Aktion $(\neg\widetilde{hr}) \wedge (\widetilde{min} \wedge min' = (min + 1) \mod 60) \wedge (\neg\widetilde{hr}) \wedge (min \neq 59)$ entspricht. ⌟

Der DNF-Graph einer Spezifikation enthält zu jeder Klausel der disjunktiven Normalform von $Next$ einen Pfad von α nach ω. Bei einer Spezifikation mit $Next \triangleq \bigwedge_{i=1}^n Next_i$ gibt es $\prod_{i=1}^n length_i$ verschiedene Pfade von α nach ω. Durch Ausnutzen des komposi-

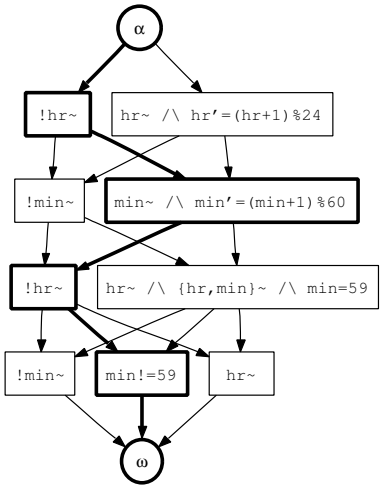


Abbildung 4.4: Pfad $\alpha k_1^1 k_2^2 k_1^3 k_2^4 \omega$ im DNF-Graphen der Spezifikation Φ_{hrmin}.

tionalen Charakters von Φ enthält der DNF-Graph jedoch nur $2 + \sum_{i=1}^{n} \mathit{length}_i$ Knoten. Somit wächst die Knotenanzahl nicht exponentiell in der Gesamtlänge von *Next*, sondern nur in der Länge der Teilspezifikationen Next_i. Neben der Einsparung von Speicherplatz (vgl. Tabelle 4.1, bei der α und ω nicht mitgerechnet sind) bietet der DNF-Graph außerdem die Grundlage für ein sehr effizientes Verfahren bei der Überprüfung von Aktionen auf Widersprüchlichkeit.

Spezifikation	Aktionen	Knoten im DNF-Graphen	Komponenten
hrmin	24	9	4
mutex2	4 320	24	7
mutex4	35 831 808	52	16
river	20 480	32	16
prodcons	16 875	30	8
3phils	64 000	33	9
5phils	102 400 000	60	20
7phils	631 242 752	84	28
10phils	1 073 741 824	120	40

Tabelle 4.1: Größe des DNF-Graphen.

Anmerkung: Wir beziehen uns in den Tabellen 4.1–4.5 auf folgende Spezifikationen:

- hrmin: Stunden-/Minutenuhr (s. Beispiel 10)
- mutex2, mutex4: wechselseitiger Ausschluss [Dij65] mit zwei bzw. vier Agenten (s. [Loh05])
- river: Flussproblem (s. Abschnitt 5.1)
- prodcons: Produzenten-/Konsumentensystem (s. Abschnitt 5.2)

- 3phils, 5phils, 7phils, 10phils: Philosophenproblem mit verschiedener Anzahl von Philosophen (s. Abschnitt 5.3)

4.1.3 On-thy-fly-Erkennung widersprüchlicher Pfade

Wie eingangs beschrieben ist eine Aktion, die zueinander widersprüchliche Konjunktionsglieder enthält, widersprüchlich. Auf den DNF-Graphen bezogen bedeutet dies, dass ein Pfad, der zueinander widersprüchliche Klauseln besucht, eine widersprüchliche Aktion beschreibt. Dieser Widerspruch kann jedoch schon erkannt werden, bevor der Knoten ω erreicht wird.

Korollar 4.2 (Pfade und Widersprüche)
Seien $\Phi \triangleq Init \wedge \Box Next$ mit $Next \triangleq \bigwedge_{j=1}^{n} Next_j$ eine umgebungsinvariante Spezifikation, $\mathcal{D}_\Phi$ der DNF-Graph für Φ, $\varphi \triangleq \bigwedge_{j=1}^{n} k_{i_j}^j$ eine widersprüchliche Aktion von Φ und $\alpha k_{i_1}^1 \dots k_{i_n}^n \omega$ der Pfad in $\mathcal{D}_\Phi$, der φ beschreibt.

Es gibt ein kleinstes l mit $1 \leq l \leq n$, sodass $\varphi^* \triangleq \bigwedge_{j=1}^{l} k_{i_j}^j$ widersprüchlich ist und alle Pfade $\alpha k_{i_1}^1 \dots k_{i_l}^l w'$ mit $w' = k_{i'_{l+1}}^{l+1} k_{i'_{l+2}}^{l+2} \dots k_{i'_n}^n \omega$ ebenfalls widersprüchliche Aktionen beschreiben. Dabei ist $1 \leq i'_j \leq length_j$ für $l < j \leq n$ ein Klauselindex. ⌟

Dies bedeutet, dass in Pfaden, die eine widersprüchliche Aktion beschreiben, eine Klausel $k_{i_j}^j$ existiert, ab der jede Fortsetzung des Pfades w' zu ω wiederum eine widersprüchliche Aktion beschreibt.

Beispiel 13. Die Spezifikation Φ_{hrmin} enthält mehrere widersprüchliche Aktionen, z. B. $k_2^1 \wedge k_1^2 \wedge k_1^3 \wedge k_1^4$ mit $k_2^1 = (\widetilde{hr} \wedge hr' = (hr + 1) \bmod 24)$, $k_1^2 = \neg\widetilde{min}$, $k_1^3 = \neg\widetilde{hr}$ und $k_1^4 = \neg\widetilde{min}$.

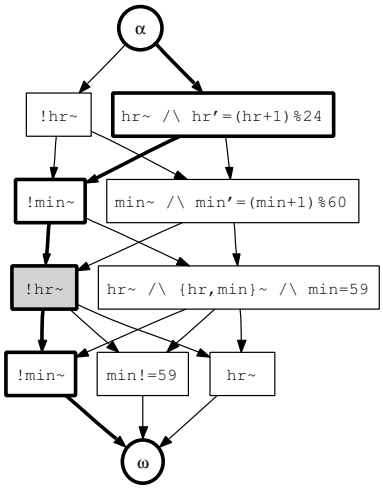


Abbildung 4.5: Pfad in $\mathcal{D}_{\Phi_{\text{hrmin}}}$, der eine widersprüchliche Aktion beschreibt.

Abbildung 4.5 zeigt den entsprechenden Pfad $\alpha k_2^1 k_1^2 k_1^3 k_1^4 \omega$ im DNF-Graphen $\mathcal{D}_{\Phi_{\text{hrmin}}}$. Dabei ist $k_1^3 = \neg\widetilde{hr}$ (in Abb. 4.5 grau hervorgehoben) die Klausel, ab der der Pfad $\alpha k_2^1 k_1^2 k_1^3$ nur so zu ω fortgesetzt werden kann, dass er eine widersprüchliche Aktion beschreibt. ⌟

Das Ergebnis aus Korollar 4.2 kann nun verwendet werden, um eine Reihe von widersprüchlichen Aktionen zu verwerfen, ohne sie explizit betrachtet zu haben: Wird auf dem Pfad im DNF-Graphen von α nach ω ein Widerspruch im Teilpfad $\alpha k_{i_1}^1 \dots k_{i_l}^l$ gefunden, kann dieser Pfad nicht so zu ω fortgesetzt werden, dass er eine Aktion beschreibt, die nicht widersprüchlich ist. Demnach können all jene Pfade mit dem Präfix $\alpha k_{i_1}^1 \dots k_{i_l}^l$ ignoriert werden und die Klauselindizes i_j für $1 \leq j \leq n$ durch i'_j mit

$$i'_j \triangleq \begin{cases} 1, & \text{falls } j > l, \\ 1, & \text{falls } j = l \text{ und } i_j = \mathit{length}_l, \\ 1, & \text{falls } j < l \text{ und } i_j = \mathit{length}_j \text{ und } i'_{j-1} = 1, \\ i_j + 1, & \text{falls } j = l \text{ und } i_j < \mathit{length}_l, \\ i_j + 1, & \text{falls } j < l \text{ und } i_j < \mathit{length}_j \text{ und } i'_{j-1} = 1, \\ i_j, & \text{sonst} \end{cases}$$

ersetzt werden.

Bei dem beschriebenen Ansatz handelt es sich um eine on-the-fly-Lösung: Anstatt alle Aktionen vollständig aufzubauen und erst anschließend zu überprüfen, wird hier jede Aktion *während* der Konstruktion auf Widersprüchlichkeit hin überprüft. Falls ein Widerspruch entdeckt wird, wird die bisher konstruierte Aktion nicht vollständig verworfen, sondern versucht, nur die Klausel, die den Widerspruch verursacht hat, auszutauschen und mit der Konstruktion fortzufahren.

Beispiel 14. Die disjunktive Normalform der Schrittformel *Next* der Spezifikation der Stunden-/Minutenuhr Φ_{hrmin} enthält 24 Klauseln. Von diesen potentiellen Aktionen sind jedoch 20 widersprüchlich und müssen verworfen werden.

Bei der Brute-Force-Implementierung wird zunächst die vollständige Schrittformel *Next* in disjunktive Normalform umgeformt und anschließend jede einzelne Aktion auf Widersprüchlichkeit überprüft. Bei der on-the-fly-Implementierung hingegen müssen nicht alle Pfade (Aktionen) bis zum Ende überprüft werden.

Abbildung 14 zeigt den DNF-Graphen $\mathcal{D}_{\Phi_{\text{hrmin}}}$, der zu einen Baum entfaltet wurde. Die vier Pfade, die nur aus weißen Knoten bestehen, beschreiben die widerspruchsfreien Aktionen von Φ_{hrmin}. Wie in Abbildung 4.5 sind die Knoten, bei denen eine Aktion als widersprüchlich erkannt wurde, grau hervorgehoben. Da die jeweiligen Pfade ab diesen Knoten nicht zu einer widerspruchsfreien Aktion fortgesetzt werden können, werden die nachfolgenden Knoten (schwarz dargestellt) nicht überprüft.

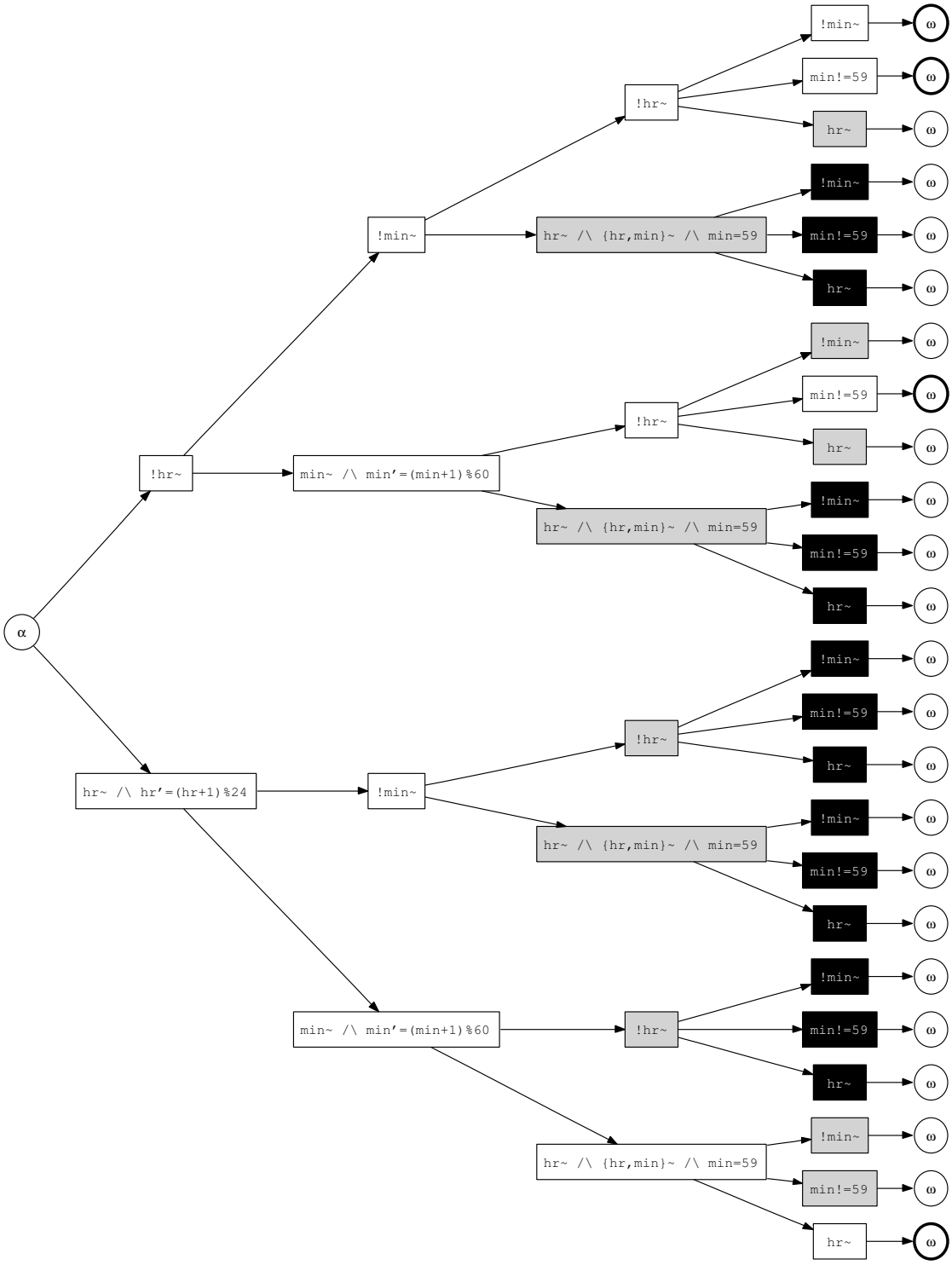


Abbildung 4.6: Alle Pfade/Aktionen der Spezifikation Φ_{hrmin}.

So kann in fünf Fällen die Überprüfung einer Aktion vorzeitig beendet werden, und anstelle von 24 Aktionen werden nur neun Aktionen vollständig (d. h. in voller Länge) überprüft. Insgesamt werden also nur 14 der insgesamt 24 Pfade überprüft. ⌟

Der im Beispiel für die Spezifikation der Stunden-/Minutenuhr beschriebene Effekt des on-the-fly-Ansatzes ist im Vergleich zu anderen Fallstudien noch gering. In der Regel muss nur ein Bruchteil (1–9 Prozent) der Pfade überprüft werden (vgl. Tabelle 4.2). Dabei werden Widersprüche in Pfaden meist sehr früh, d. h. nach Überprüfen weniger Knoten, entdeckt (vgl. Tabelle 4.3).

Die beschriebenen Daten hängen jedoch in großem Maße von den Algorithmen ab, die entscheiden, ob eine Aktion widersprüchlich ist oder nicht. Für Widersprüche, die aufgrund der $\sim$-Variablen auftreten, ist mit $\sim$-Graphen (s. Abschnitt 4.2) dieses Problem effizient gelöst. Für andere Konstellationen, die zu Widersprüchen führen, sind bisher nur einige wenige Ansätze entwickelt und implementiert, die in Abschnitt 4.3 diskutiert werden.

Spezifikation	Pfade gesamt	Pfade untersucht	Rate
hrmin	24	14	58,33 %
mutex2	4 320	387	8,96 %
mutex4	35 831 808	28 061	0,07 %
river	20 480	99	0,48 %
prodcons	16 875	1 407	8,34 %
3phils	64 000	556	0,87 %
5phils	102 400 000	9 984	0,01 %

Tabelle 4.2: Anzahl der überprüften Pfade im DNF-Graphen.

Spezifikation	maximale Pfadlänge	durchschnittliche Pfadlänge bei Abbruch
hrmin	4	3,3
mutex2	7	4,8
mutex4	16	7,7
river	16	8,8
prodcons	8	6,2
3phils	15	5,1
5phils	20	15,2
7phils	28	21,3

Tabelle 4.3: Durchschnittliche Pfadlänge der widersprüchlichen Klauseln.

4.2 Der $\sim$-Graph

Viele der Klauseln der disjunktiven Normalform von *Next* sind 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. Ein Großteil der Klauseln ist dabei wegen der ∼-Variablen widersprüchlich. Die Überprüfung solcher Klauseln auf Widerspruchsfreiheit ist jedoch nicht trivial, da Widersprüche aufgrund der ∼-Variablen nicht direkt syntaktisch zu erkennen sind.

In diesem Abschnitt beschäftigen wir uns mit der Erkennung von Aktionen, die aufgrund der ∼-Variablen widersprüchlich sind. Dazu definieren wir in Abschnitt 4.2.1 zunächst syntaktische Kriterien, mit denen Informationen über ∼-Variablen (Informationsmengen) direkt aus der Aktion abgeleitet werden können. Anschließend beschreiben wir in Abschnitt 4.2.2 ∼-Graphen, die als Datenstruktur für Informationsmengen benutzt werden können. In Abschnitt 4.2.3 geben wir ein Verfahren an, das die Eigenschaften des ∼-Graphen ausnutzt und Informationsmengen effizient berechnet. Diese Informationsmengen verwenden wir in Abschnitt 4.2.4 um widersprüchliche Aktionen zu erkennen. Zuletzt beschreiben wir in Abschnitt 4.2.5, wie Informationsmengen für die Codegenerierung zum Aufbau des Transitionssystemes und das anschließende Modelchecking platzsparend gespeichert werden können.

4.2.1 Informationsmengen

Die aus ∼-Variablen gebildeten Schrittformeln haben in Schritten gewisse Eigenschaften. Wir werden im Rest dieses Abschnittes diese Eigenschaften benutzen, um Widersprüche in Aktionen – also Schrittformeln – zu erkennen. Die angegebenen Regeln sind dabei nicht vollständig, d. h. mit ihnen können nicht Informationen über die Involviertheit aller ∼-Variablen einer Spezifikation abgleitet werden. Allerdings können diese Regeln in syntaktische Kriterien überführt und effizient implementiert werden.

Lemma 4.3 (Eigenschaften von ∼-Variablen in Schritten)
Seien S_C ein Schritt und $\widetilde{a}, \widetilde{b}, \widetilde{c} \in \widetilde{\mathcal{V}ar}$. Es gilt:

i. $S_C \models \widetilde{a}$, *falls* $S_C \models \widetilde{b}$ *und* $a \subseteq b$ *(Zerlegung)*

ii. $S_C \models \widetilde{c}$, *falls* $S_C \models (\widetilde{a} \wedge \widetilde{b})$, $c = a \cup b$ *und* $a \cap b \neq \emptyset$ *(Transitivität)*

iii. $S_C \models \neg\widetilde{b}$, *falls* $S_C \models \neg\widetilde{a}$ *und* $a \subseteq b$ *(Erweiterung)*

⌟

Beweis. Seien σ ein Ablauf und $S_C = (C, C', T_C)$ ein Schritt in σ.

Zerlegung: Wegen $S_C \models \widetilde{b}$ existiert eine Transition $t \in T_C$ mit $b \subseteq dom(t)$. Wegen $a \subseteq b$ gilt $a \subseteq dom(t)$ und somit $S_C \models \widetilde{a}$.

Transitivität: Wegen $S_C \models (\widetilde{a} \wedge \widetilde{b})$ existierten Transitionen $t_1, t_2 \in T_C$ mit $a \subseteq dom(t_1)$ und $b \subseteq dom(t_2)$. Wegen $a \cap b \neq \emptyset$ existiert eine nichtleere Variablenmenge $x = a \cap b$ mit $x \subseteq dom(t_1)$ und $x \subseteq dom(t_2)$. Nach Definition des Ablaufes ist jede Variable

in einem Schnitt höchstens von einer Transition involviert. Somit ist $t_1 = t_2$ und $a \subseteq dom(t_1)$ und $b \subseteq dom(t_1)$. Für $c = a \cup b$ ist $c \subseteq dom(t_1)$, und es gilt $S_C \parallel\!\models \widetilde{c}$.

Erweiterung: Wegen $S_C \parallel\!\models \neg\widetilde{a}$ existiert in T_C keine Transition t mit $a \subseteq dom(t)$. Somit existiert auch für $b \supseteq a$ keine Transition $t' \in T_C$ mit $b \subseteq dom(t')$. Daher gilt $S_C \parallel\!\models \neg\widetilde{b}$. □

Die in Lemma 4.3 angegebenen Regeln erlauben es, Informationen über $\sim$-Variablen eines Schrittes abzuleiten. Allerdings ist dazu stets die explizite Kenntnis eines Schrittes mit den darin schaltenden Transitionen notwendig. Für die Überprüfung von Aktionen auf Widersprüchlichkeit müssen Informationen über $\sim$-Variablen jedoch syntaktisch, d. h. direkt aus einer Schrittformel, abgeleitet werden, da bei der Analyse einer Spezifikation die Abläufe nicht explizit bekannt sind. Dazu definieren wir zunächst den Begriff der Informationsmenge:

Definition 34 (Informationsmenge)
Seien Φ eine umgebungsinvariante Spezifikation und α eine widerspruchsfreie Aktion von Φ. Die Informationsmenge I_α für α*, eine Menge von negierten und unnegierten $\sim$-Variablen, sei durch folgende Fixpunktoperation konstruiert:*

1. *Initialisierung:*

   $I_\alpha := \{\widetilde{a} \mid \widetilde{a}$ *kommt in* α *unnegiert vor*$\} \cup \{\neg\widetilde{a} \mid \widetilde{a}$ *kommt in* α *negiert vor*$\}$

2. *Solange bis kein neues Element zu I_α hinzukommt:*

$$\begin{aligned} I_\alpha := I_\alpha &\cup \{\widetilde{a} \mid \widetilde{b} \in I_\alpha \text{ und } \emptyset \neq a \subseteq b\} && \text{(Zerlegung)} \\ &\cup \{\widetilde{c} \mid \widetilde{a}, \widetilde{b} \in I_\alpha,\ c = a \cup b \text{ und } a \cap b \neq \emptyset\} && \text{(Transitivität)} \\ &\cup \{\neg\widetilde{b} \mid \neg\widetilde{a} \in I_\alpha \text{ und } a \subseteq b \subseteq V(\Phi)\} && \text{(Erweiterung)} \end{aligned}$$

⌟

Lemma 4.4 *Die Konstruktion aus Definition 34 terminiert.*

Beweis. Φ ist eine umgebungsinvariante Spezifikation und α eine widerspruchsfreie Aktion von Φ. In jedem Konstruktionsschritt werden I_α entweder Elemente hinzugefügt oder I_α bleibt gleich und die Konstruktion terminiert.

Im 1. Konstruktionsschritt werden nur $\sim$-Variablen hinzugefügt, die in α vorkommen. Da α endlich ist, ist auch I_α nach dem 1. Konstruktionsschritt endlich.

In jedem Durchlauf des 2. Konstruktionsschrittes werden Teilmengen (Zerlegung) von Elementen aus I_α, Vereinigungen von Elementen aus I_α (Transitivität) oder Teilmengen von $V(\Phi)$ (Erweiterung) zu I_α hinzugefügt. Da I_α und $V(\Phi)$ endlich sind, können nur endlich viele Elemente hinzugefügt werden. Es gilt:

$$I_\alpha \subseteq (\{\widetilde{a} \mid \emptyset \neq a \subseteq V(\Phi)\} \cup \{\neg\widetilde{a} \mid \emptyset \neq a \subseteq V(\Phi)\})\,.$$

□

Die Konstruktionsregeln geben also an, wie ohne Kenntnis eines Schrittes rein syntaktisch Informationen über ∼-Variablen abgeleitet werden können. Dennoch besteht ein Zusammenhang zwischen den Schritten, die α erfüllen, und der Informationsmenge I_α.

Lemma 4.5 (Informationsmengen und Schritte)
Seien Φ eine umgebungsinvariante Spezifikation, α eine widerspruchsfreie Aktion von Φ und I_α die Informationsmenge für α. Seien σ ein Ablauf von Φ und S_C ein Schritt in σ mit $S_C \Vdash \alpha$. Dann gilt: $S_C \Vdash \bigwedge_{r \in I_\alpha} r$. ⌟

Beweis. Sei $I_\alpha = \{r_1, \ldots, r_n\}$. Der Beweis wird induktiv über den Aufbau der Formel $(\bigwedge_{r \in I_\alpha} r) = (\bigwedge_{i=1}^{n} r_i) = r_1 \wedge \ldots \wedge r_n$ geführt.

Sei r_i ein beliebiges Element aus I_α und S_C ein Schritt mit $S_C \Vdash \alpha$.

Falls r_i im Initialisierungsschritt zu I_α hinzugefügt wurde, kommt r_i in α negiert bzw. unnegiert vor, und es gilt offensichtlich $S_C \Vdash r_i$.

Falls $r_i = \widetilde{a}$ im 2. Konstruktionsschritt mit der Zerlegungsregel zu I_α hinzugefügt wurde, existiert ein $\widetilde{b} \in I_\alpha$ mit $a \subseteq b$. Nach Induktionsvoraussetzung gilt $S_C \Vdash \widetilde{b}$. Nach Lemma 4.3.i gilt $S_C \Vdash \widetilde{a}$ und somit $S_C \models r_i$.

Falls $r_i = \widetilde{c}$ im 2. Konstruktionsschritt mit der Transitivitätsregel zu I_α hinzugefügt wurde, existieren $\widetilde{a}, \widetilde{b} \in I_\alpha$ mit $c = a \cup b$ und $a \cap b \neq \emptyset$. Nach Induktionsvoraussetzung gilt $S_C \models (\widetilde{a} \wedge \widetilde{b})$. Nach Lemma 4.3.ii gilt $S_C \Vdash \widetilde{c}$ und somit $S_C \models r_i$.

Falls $r_i = \neg\widetilde{b}$ im 2. Konstruktionsschritt mit der Erweiterungsregel zu I_α hinzugefügt wurde, existiert ein $\neg\widetilde{a} \in I_\alpha$ mit $a \subseteq b \subsetneq V(\Phi)$. Nach Induktionsvoraussetzung gilt $S_C \Vdash \neg\widetilde{a}$. Nach Lemma 4.3.iii gilt $S_C \Vdash \neg\widetilde{b}$ und somit $S_C \models r_i$.

□

Mit dem in Definition 34 beschriebenem Verfahren haben wir einen Algorithmus entwickelt, mit dem auf syntaktischem Wege Informationen über die ∼-Variablen hergeleitet werden können. Dabei beschränken wir uns jedoch auf die Informationen, die direkt aus den ∼-Variablen abgeleitet werden können. Regeln, die bspw. aus $x' \neq x$ die Involviertheit von x ableiten, betrachten wir in Abschnitt 4.3.

4.2.2 Definition und Beschriftung des ∼-Graphen

Wir wollen in diesem Abschnitt eine Datenstruktur beschreiben, mit der Informationsmengen gespeichert werden können. Sie bildet die Grundlage für weitere Algorithmen (z. B. den on-the-fly-Ansatz mithilfe des DNF-Graphen, vgl. Abschnitt 4.1) und zur effizienteren Berechnung und Speicherung von Informationsmengen.

Definition 35 (∼-Graph, Elternknoten, Kindknoten)
Sei $\emptyset \neq V \subset \mathcal{V}ar$ eine endliche Variablenmenge. $\mathcal{G}_V = (N, E)$ ist der ∼-Graph für V, *falls:*

i. $N \triangleq \mathfrak{P}(V)\backslash\emptyset$ *ist die Menge der Knoten,*

ii. $E \triangleq \{\{u, v\} \mid u, v \in N, u \subset v, card(v) = card(u) + 1\}$ *ist die Menge der Kanten.*

Für einen Knoten $u \in N$ *bezeichnet* ${}^\bullet u \triangleq \{v \mid \{u, v\} \in E$ *und* $v \subset u\}$ *die Menge der* Elternknoten von u *und* $u^\bullet \triangleq \{v \mid \{u, v\} \in E$ *und* $u \subset v\}$ *die Menge der* Kindknoten von u. ⌟

Notation: Wir lassen – wie auch bei ∼-Variablen – Mengenklammern weg und schreiben für den Knoten $\{x, y\}$ kurz xy. Entsprechend schreiben wir für die Kante $\{\{x, y\}, \{x, y, z\}\}$ kurz $\{xy, xyz\}$.

Die Ableitungsregeln in Def. 34 für Informationen über ∼-Variablen basieren auf Mengenbeziehungen zwischen Mengen von Systemvariablen: Die Zerlegung betrachtet für eine Menge b alle Untermengen $a \subseteq b$, die Erweiterung für eine Menge a alle Obermengen $b \supseteq a$ und die Transitivität für zwei Elemente a und b alle nicht-disjunkten Vereinigungen $a \cup b$, d. h. erneut Obermengen.

Bei der Definition des ∼-Graphen ist diese Hierarchie zwischen einer Menge, ihren Unter- und Obermengen durch Kanten ausgedrückt: Für jede Teilmenge v der Variablenmenge V existiert im ∼-Graphen $\mathcal{G}_V = (N, E)$ ein Knoten $v \in N$. Dieser Knoten ist mit all seinen Untermengen u mit $card(u) = card(v) - 1$ und seinen Obermengen w mit $card(w) = card(v) + 1$ durch eine Kante verbunden. Diese Kanten werden im Folgenden dazu benutzt, die beschriebenen Mengenbeziehungen in konstanter Zeit zu bestimmen (s. Lemma 4.7).

Zur Illustration geben wir ein Beispiel für einen ∼-Graphen an, der Grundlage für die folgenden Beispiele 16–18 in diesem Abschnitt ist:

Beispiel 15. Abbildung 4.7 zeigt den ∼-Graphen für die Variablenmenge $V = \{w, x, y, z\}$.

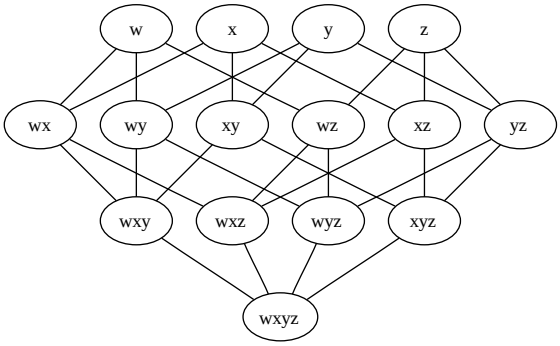


Abbildung 4.7: ∼-Graph $\mathcal{G}_V$ für $V = \{w, x, y, z\}$.

Dabei gilt bspw. ${}^\bullet xyz = \{xy, xz, yz\}$, ${}^\bullet w = \emptyset$ und $xy^\bullet = \{wxy, xyz\}$. ⌟

Die einzelnen Knoten des ∼-Graphen $\mathcal{G}_V$ können als ∼-Variablen aufgefasst werden, d. h. die Teilmenge $u \subseteq V$ steht für die ∼-Variable $\widetilde{u}$. Um Informationsmengen zu speichern,

können die Knoten des ∼-Graphen entsprechend der Informationen in der Informationsmenge beschriftet werden:

Definition 36 (Beschriftung des ∼-Graphen mit einer Informationsmenge)
Seien Φ eine umgebungsinvariante Spezifikation, α eine widerspruchsfreie Aktion von Φ, I_α die Informationsmenge für α und $\mathcal{G}_{V(\Phi)} = (N, E)$ der ∼-Graph für die Menge der Systemvariablen von Φ.

Die Abbildung $L_{I_\alpha} : N \to \{pos, neg, ?\}$ ist die Beschriftung des ∼-Graphen $\mathcal{G}_{V(\Phi)}$ mit der Informationsmenge I_α, *falls für alle $u \in N$ gilt:*

$$L_{I_\alpha}(u) = \begin{cases} pos, & \text{falls } \widetilde{u} \in I_\alpha, \\ neg, & \text{falls } \neg\widetilde{u} \in I_\alpha, \\ ?, & \text{sonst.} \end{cases}$$

⌟

Beispiel 16. Seien Φ eine umgebungsinvariante Spezifikation mit den Systemvariablen $V(\Phi) = \{w, x, y, z\}$ und α eine Aktion von Φ mit der Informationsmenge

$$I_\alpha = \{\neg\widetilde{w}, \widetilde{x}, \widetilde{y}, \widetilde{z}, \neg\widetilde{wx}, \neg\widetilde{wy}, \widetilde{xy}, \neg\widetilde{wz}, \widetilde{xz}, \widetilde{yz}, \neg\widetilde{wxy}, \neg\widetilde{wxz}, \neg\widetilde{wyz}, \widetilde{xyz}, \neg\widetilde{wxyz}\}\,.$$

Abbildung 4.8 zeigt den ∼-Graphen $\mathcal{G}_{V(\Phi)}$ mit der Beschriftung L_{I_α}.

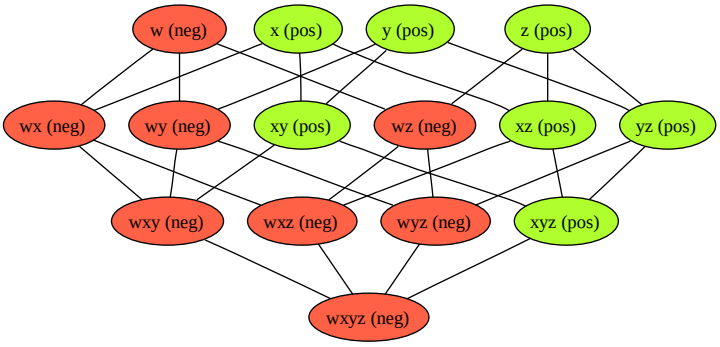


Abbildung 4.8: ∼-Graph $\mathcal{G}_{V(\Phi)}$ mit der Beschriftung L_{I_α}.

Es gilt bspw. $L_{I_\alpha}(xyz) = pos$ und $L_{I_\alpha}(w) = neg$. ⌟

Anhand der Beschriftung eines ∼-Graphen kann die Informationsmenge rekonstruiert werden. Diesen Ansatz greifen wir in Abschnitt 4.2.5 auf, in dem wir ein Verfahren angeben, mit dem nur die Beschriftung einiger weniger Knoten ausreicht, um die Beschriftung der restlichen Knoten – und somit alle Elemente der zugrunde liegenden Informationsmenge – zu rekonstruieren.

4.2.3 Graphbasierte Berechnung von Informationsmengen

Bisher haben wir beschrieben, wie sich, gegeben eine widerspruchsfreie Aktion α einer umgebungsinvarianten Spezifikation Φ, die Informationsmenge I_α berechnen lässt, der $\sim$-Graph $\mathcal{G}_{V(\Phi)}$ konstruiert werden kann und sich dessen Knoten mit den Informationen aus I_α beschriften lassen. Dabei haben wir im letzten Abschnitt gezeigt, dass die Informationsmenge aus der Beschriftung rekonstruiert werden kann. Wir wollen nun die Beschriftung des $\sim$-Graphen nicht anhand der Informationsmenge I_α vornehmen, sondern sie direkt anhand der Aktion α ableiten. Dazu definieren wir ein Verfahren, das den Aufbau des $\sim$-Graphen ausnutzt und die Regeln aus Definition 34 neu formuliert, d. h. Mengenbeziehungen durch Kantenbeziehungen ersetzt.

Definition 37 (Graphbasierte Berechnung einer Beschriftung)
Seien Φ eine umgebungsinvariante Spezifikation, α eine widerspruchsfreie Aktion von Φ und $\mathcal{G}_{V(\Phi)} = (N, E)$ der $\sim$-Graph für die Menge der Systemvariablen von Φ. Sei L eine Beschriftung des $\sim$-Graphen $\mathcal{G}_{V(\Phi)}$, die durch folgende Fixpunktoperation konstruiert wird:

1. *Initialisierung:*

$$L(a) = \begin{cases} pos, & \text{falls } \widetilde{a} \text{ in } \alpha \text{ unnegiert vorkommt,} \\ neg, & \text{falls } \widetilde{a} \text{ in } \alpha \text{ negiert vorkommt} \\ ?, & \text{sonst.} \end{cases}$$

2. *Solange bis sich L nicht mehr ändert:*

$$L(c) = \begin{cases} pos, & \text{falls } L(c) = ?,\ \exists b \in N, L(b) = pos \text{ und } c \in {}^\bullet b, \\ pos, & \text{falls } L(c) = ?,\ \exists a, b \in N,\ a \neq b,\ L(a) = L(b) = pos,\ \{a, b\} \subseteq {}^\bullet c \\ & \quad \text{und } card(c) > 2, \\ neg, & \text{falls } L(c) = ?,\ \exists a \in N,\ L(a) = neg \text{ und } c \in a^\bullet. \end{cases}$$

⌟

Anmerkung: Die Regeln im 2. Konstruktionsschritt entsprechen der Zerlegung, Transitivität und Erweiterung aus Definition 34, wurden jedoch hier nicht durch Kantenbeziehungen anstelle von Mengenbeziehungen ausgedrückt.

Lemma 4.6 *Die Konstruktion aus Definition 37 terminiert.*

Beweis. Angenommen, die Konstruktion terminiert nicht, d. h. die Beschriftung L ändert sich unendlich oft. Da die Knotenanzahl von $\mathcal{G}_{V(\Phi)}$ endlich ist, muss es einen Knoten $v \in N$ geben, der mehrfach beschriftet wird, damit sich L unendlich oft ändert. $L(v)$ kann jedoch nur geändert werden, falls $L(v) = ?$. Nach der Änderung gilt $L(v) \neq ?$, sodass $L(v)$ nicht mehr geändert werden kann. Somit terminiert die Konstruktion, was im Widerspruch zur Annahme, die Konstruktion terminiere nicht, steht. □

Lemma 4.7 (Berechnete Beschriftung entspricht Informationsmenge)
Seien Φ eine umgebungsinvariante Spezifikation, α eine widerspruchsfreie Aktion von Φ, I_α die Informationsmenge für α. und $\mathcal{G}_{V(\Phi)} = (N, E)$ der $\sim$-Graph für die Menge der Systemvariablen von Φ. Sei L die in Definition 37 konstruierte Beschriftung von $\mathcal{G}_{V(\Phi)}$. Es gilt für alle Knoten $u \in N$: $L(u) = L_{I_\alpha}(u)$. ⌟

Beweis von Lemma 4.7. Der Beweis wird über die Konstruktionsschritte von Definition 37 geführt:

1. - Falls $\widetilde{a}$ in α unnegiert vorkommt, gilt $\widetilde{a} \in I_\alpha$ (Def. 34) und $L_{I_\alpha}(a) = pos$ (Def. 36).
   - Falls $\widetilde{a}$ in α negiert vorkommt, gilt $\neg\widetilde{a} \in I_\alpha$ (Def. 34) und $L_{I_\alpha}(a) = neg$ (Def. 36).
   - Ansonsten gilt $\widetilde{a} \notin I_\alpha$ (Def. 34) und $L_{I_\alpha}(a) = ?$ (Def. 36).

   Nach der Initialisierung gilt somit $L_{I_\alpha}(a) = L(a)$.
2. - Sei $b \in N$ mit $L(b) = L_{I_\alpha}(b) = pos$. Es gilt $\widetilde{b} \in I_\alpha$ (Def. 36). Für alle $c \in {}^\bullet b$ gilt $c \subset b$ (Def. 35) und $\widetilde{c} \in I_\alpha$ (Def. 34) sowie $L_{I_\alpha}(c) = pos$ (Def. 36).
   - Sei $a, b \in N$ mit $L_{I_\alpha}(a) = L_{I_\alpha}(b) = pos$. Es gilt $\{\widetilde{a}, \widetilde{b}\} \subseteq I_\alpha$ (Def. 36). Wegen $\{a, b\} \subseteq {}^\bullet c$ gilt $a \subset c$, $b \subset c$ und $card(a) = card(b) = card(c) - 1$ (Def. 35). Mit $a \neq b$ gilt: $a \cup b = c$. Aus $card(c) > 2$ folgt $card(a) = card(b) > 1$ und $a \cap b \neq \emptyset$. Somit gilt $c \in I_\alpha$ (Def. 34) und $L_{I_\alpha}(c) = pos$ (Def. 36).
   - Sei $b \in N$ mit $L(b) = L_{I_\alpha}(b) = neg$. Es gilt $\neg\widetilde{b} \in I_\alpha$ (Def. 36). Für alle $c \in b^\bullet$ gilt $b \subset c$ (Def. 35) und $\neg\widetilde{c} \in I_\alpha$ (Def. 34) sowie $L_{I_\alpha}(c) = neg$ (Def. 36).

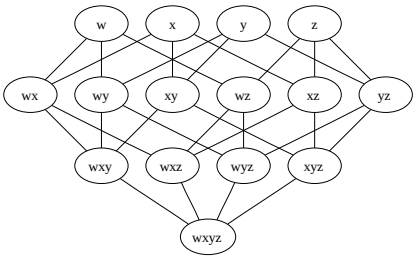
   Nach Durchlauf des 2. Konstruktionsschrittes gilt somit $L_{I_\alpha}(c) = L(c)$.

□

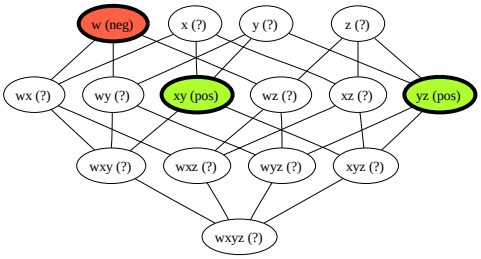
Beispiel 17. Seien Φ eine umgebungsinvariante Spezifikation mit den Systemvariablen $V(\Phi) = \{w, x, y, z\}$ und $\alpha = \neg\widetilde{w} \wedge \widetilde{xy} \wedge \widetilde{yz}$ eine Aktion von Φ. Abbildung 4.9 illustriert, wie die Informationsmenge I_α mithilfe des $\sim$-Graphen $V_{V(\Phi)}$ bestimmt wird.

Dabei ist der aktuell betrachtete Knoten jeweils fett hervorgehoben. Die Pfeile deuten die Propagierung der Informationen entsprechend der Induktionsschritte von Lemma 4.7 an. ⌟

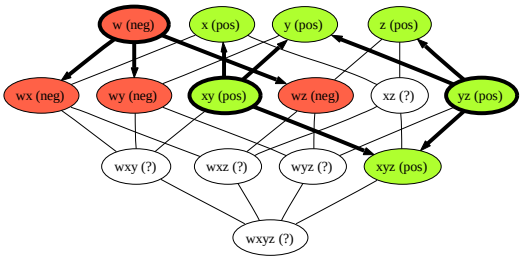
Wie in Abbildung 4.9 deutlich wird, werden Informationen (Beschriftungen mit *pos* und *neg*) durch den $\sim$-Graphen propagiert. Dabei wird kein Knoten mehrfach beschriftet. Durch diese minimale Propagierung von Informationen wird verhindert, dass Teile des $\sim$-Graphen erneut – also überflüssigerweise – beschriftet werden. Dieses effizientere Verfahren zeigte in der Praxis eine große Geschwindigkeitssteigerung gegenüber der Brute-Force-Implementierung. Tabelle 4.4 zeigt Ergebnisse der neuen Implementierung mit $\sim$-Graphen bei verschiedenen Spezifikationen: Insgesamt lassen sich 30–70 % aller Operationen auf dem $\sim$-Graphen vermeiden.



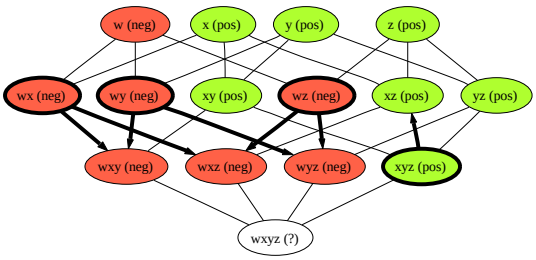
(a) ∼-Graph ohne Beschriftung.



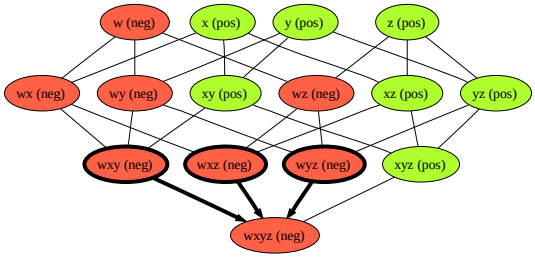
(b) Initialisierung.



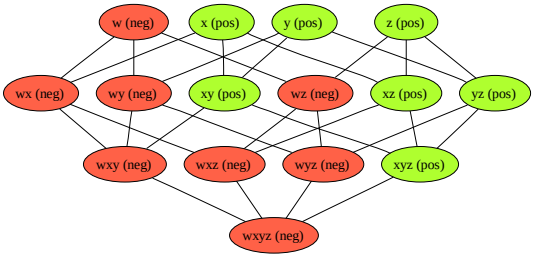
(c) 1. Durchlauf.



(d) 2. Durchlauf.



(e) 3. Durchlauf.



(f) 4. Durchlauf, Fixpunkt erreicht.

Abbildung 4.9: Bestimmung der Informationsmenge I_α anhand des ∼-Graphen $\mathcal{G}_{V(\Phi)}$.

Spezifikation	Operationen auf Informationsmengen	Vermeidung von Propagierungen	Rate
hrmin	70	21	30,0 %
mutex2	4 956	1 649	33,3 %
mutex4	1 808 862	996 787	55,1 %
river	3 461	1 776	51,3 %
prodcons	17 426	5 595	32,1 %
3phils	79 571	49 077	61,7 %
5phils	45 478 283	36 120 071	79,4 %

Tabelle 4.4: Vermeidung der Propagierung bekannter Informationen.

4.2.4 Widersprüchliche Informationsmengen und Beschriftungen

In den bisherigen Abschnitten haben wir uns stets auf widerspruchsfreie Aktionen und damit auch widerspruchsfreie Informationsmengen und Beschriftungen beschränkt. Allerdings soll der ∼-Graph gerade dafür verwendet werden, Aktionen, die aufgrund von ∼-Variablen widersprüchlich sind, zu erkennen.

Unter einer widersprüchlichen Aktion verstehen wir in diesem Abschnitt also eine Aktion α, deren Informationsmenge I_α widersprüchlich ist, d. h. dass für eine ∼-Variable $\widetilde{a}$ sowohl $\widetilde{a} \in I_\alpha$ als auch $\neg\widetilde{a} \in I_\alpha$ gilt. Wird diese Informationsmenge benutzt, um den ∼-Graphen zu beschriften, müsste der Knoten a sowohl mit *pos*, als auch mit *neg* beschriftet werden.

Vergleichbar mit dem im vorherigen Abschnitt beschrieben Verfahren kann mit der Beschriftung des ∼-Graphen auch in dem Moment abgebrochen werden, in dem ein bereits beschrifteter Knoten mit der entsprechend widersprüchlichen Information beschriftet werden soll. Eine Propagierung von Informationen ist einerseits nicht möglich, andererseits auch überflüssig, da die Informationsmenge der Aktion und somit die Aktion selbst als widersprüchlich erkannt wurde. Die aktuelle Aktion kann sofort verworfen werden, sodass sich das beschriebene Verfahren mit dem on-the-fly-Ansatz des DNF-Graphen (s. Abschnitt 4.1) kombinieren lässt.

Mit dem ∼-Graphen können jedoch nur Widersprüche in der Informationsmenge einer Aktion entdeckt werden. Andere Gründe für Widersprüchlichkeit können nicht gefunden werden: So hat beispielsweise die widersprüchliche Aktion $\alpha_1 \triangleq \widetilde{x} \wedge x = 0 \wedge x = 1$ eine widerspruchsfreie Informationsmenge. Ebenso wird die Aktion $\alpha_2 \triangleq \neg\widetilde{x} \wedge x' \neq x$ nicht als widersprüchlich erkannt, obwohl $x \neq x'$ die Information $\widetilde{x}$ (und damit einen Widerspruch zu $\neg\widetilde{x}$) impliziert. Widersprüche dieser Art werden in Abschnitt 4.3.4 untersucht.

4.2.5 Minimale Repräsentation von Informationsmengen

Falls eine Aktion nicht wie im letzten Abschnitt als widersprüchlich erkannt wurde, muss die Informationsmenge für die spätere Verwendung gespeichert werden.

Einerseits wird sie beim Aufbau des Transitionssystemes, also bei der Berechnung der Nachfolgezustände eines Zustandes, verwendet. Dabei kann für jede einelementige ∼-Variable $\widetilde{x}$, die negiert in der Informationsmenge vorkommt, abgeleitet werden, dass sich der Wert der Systemvariable x im Nachfolgezustand nicht ändert, d. h. es gilt $x' = x$. Andererseits ist jeder Zustandsübergang des Transitionssystems mit einer Aktion beschriftet. Die Informationen über die ∼-Variablen der Aktion werden für das explizite Modelchecking verwendet und können nicht aus den Zuständen (Wertetupeln) abgeleitet werden.

Die Informationsmengen müssen also vollständig gespeichert werden. Da jedoch zu jeder Aktion Quellcode generiert wird, muss die Repräsentation der Informationsmengen so kompakt wie möglich sein, weil der erzeugte Quelltext sonst nicht übersetzt werden kann.

Allerdings können wir erneut die Struktur des $\sim$-Graphen ausnutzen, wegen der die Beschriftung des $\sim$-Graphen mit einer Informationsmenge gewisse Eigenschaften hat:

Korollar 4.8 (Eigenschaften der Beschriftung des $\sim$-Graphen)
Seien Φ eine umgebungsinvariante Spezifikation, α eine widerspruchsfreie Aktion von Φ, I_α die Informationsmenge für α, $\mathcal{G}_{V(\Phi)} = (N, E)$ der $\sim$-Graph für die Menge der Systemvariablen von Φ, $u \in N$ und L_{I_α} die Beschriftung des $\sim$-Graphen $\mathcal{G}_{V(\Phi)}$ mit I_α. Es gilt:

i. Für alle Knoten $v \in {}^\bullet u$ mit $L_{I_\alpha}(u) = pos$ gilt: $L_{I_\alpha}(v) = pos$.

ii. Für alle Knoten $v \in u^\bullet$ mit $L_{I_\alpha}(u) = neg$ gilt: $L_{I_\alpha}(v) = neg$.

⌟

Anmerkung: Diese Eigenschaften ähneln zwar denen der Zerlegungs- bzw. Erweiterungsregel aus Definition 37, werden hier jedoch anders interpretiert: Wurden bei der graphbasierten Berechnung einer Informationsmenge Regeln angewandt, um Informationen über die $\sim$-Variablen zu gewinnen, betrachten wir nun einen bereits vollständig beschrifteten Graphen.

Die Beschriftung einiger Knoten lässt sich also aus der Beschriftung anderer Knoten herleiten, z. B. impliziert die Beschriftung eines Knotens u mit *pos* die Beschriftung all seiner Kindknoten mit *pos*. Um die Informationsmenge abzuspeichern, müssten so die Beschriftungen der Kindknoten von u nicht explizit gespeichert werden.

Definition 38 (Repräsentantenmenge, minimale Repräsentantenmenge)
Seien Φ eine umgebungsinvariante Spezifikation, α eine widerspruchsfreie Aktion von Φ und I_α die Informationsmenge für α.

Eine Menge R von negierten und unnegierten $\sim$-Variablen ist eine Repräsentantenmenge von I_α, *falls für $\varphi \triangleq \bigwedge_{r \in R} r$ gilt: $I_\varphi = I_\alpha$. R ist die* minimale Repräsentantenmenge von I_α, *falls für alle $r \in R$ gilt: $R \backslash \{r\}$ ist keine Repräsentantenmenge von I_α.* ⌟

Um die minimale Repräsentantenmenge zu bestimmen, dürfen also insbesondere nicht solche negierten oder unnegierten $\sim$-Variablen hinzugefügt werden, die mit Regeln aus Lemma 4.7 aus bestehenden $\sim$-Variablen abgeleitet werden können. Die Beobachtungen aus Korollar 4.8 ermöglichen – gegeben ein mit einer Informationsmenge beschrifteter $\sim$-Graph – eine graphbasierte Bestimmung der minimalen Repräsentantenmenge dieser Informationsmenge.

Korollar 4.9 (Graphbasierte Bestimmung der minimalen Repräsentantenmenge)
Seien Φ eine umgebungsinvariante Spezifikation, α eine widerspruchsfreie Aktion von Φ, $\mathcal{G}_{V(\Phi)} = (N, E)$ der ∼-Graph für die Menge der Systemvariablen von Φ und L_{I_α} die Beschriftung von $\mathcal{G}_{V(\Phi)}$ mit der Informationsmenge I_α. Sei

$$R \triangleq \{\widetilde{u} \mid u \in N, L_{I_\alpha}(u) = pos \text{ und } L_{I_\alpha}(v) \neq pos \text{ für alle } v \in u^\bullet\} \cup \{\widetilde{u} \mid u \in N, L_{I_\alpha}(u) = neg \text{ und } L_{I_\alpha}(v) \neq neg \text{ für alle } v \in {}^\bullet u\}.$$

Dann ist R die minimale Repräsentantenmenge von I_α. ⌟

Beispiel 18. Für die Informationsmenge I_α aus Beispiel 17 ist offensichtlich $\{\neg\widetilde{w}, \widetilde{xy}, \widetilde{yz}\}$ eine Repräsentantenmenge. Die minimale Repräsentantenmenge von I_α ist $\{\neg\widetilde{w}, \widetilde{xyz}\}$. Abbildung 4.10 illustriert die graphbasierte Bestimmung dieser minimalen Repräsentantenmenge.

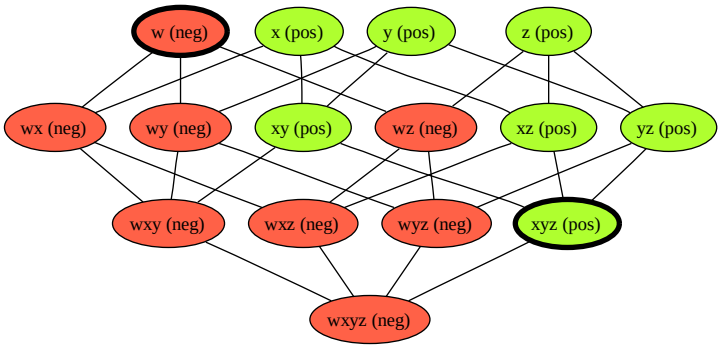


Abbildung 4.10: Minimale Repräsentantenmenge $\{\neg\widetilde{w}, \widetilde{xyz}\}$ für die Informationsmenge aus Beispiel 17.

Dabei ist ${}^\bullet w = \emptyset$ und für den Knoten $wxyz \in xyz^\bullet$ gilt: $L_{I_\alpha}(wxyz) \neq pos$. Der Knoten xyz repräsentiert alle unnegierten und w alle negierten Elemente der Informationsmenge I_α. ⌟

Wie bereits beschrieben, spielt bei der Implementierung eine große Rolle, da für jede Aktion Code generiert werden muss, der Informationen über die Involviertheit der Systemvariablen von Φ enthält: Für verschiedene Spezifikationen konnte mit dem Brute-Force-Prototyp zwar Quellcode für die automatische Konstruktion des Transitionssystemes generiert werden, allerdings war dieser wegen der expliziten Repräsentation der Informationsmengen in jeder Aktion so umfangreich, dass er nicht übersetzt werden konnte. Die Reduktion auf die minimale Repräsentation der Informationsmengen bei der Speicherung reichte in einigen Fällen aus, um den resultierenden Code übersetzen zu können.

Dabei können die minimalen Repräsentantenmengen im ungünstigsten Fall zwar noch immer exponentiell in der Anzahl der ∼-Variablen sein, allerdings sind diese Konstellationen der Informationen eher theoretischer Natur und dürften in realen Spezifikationen

nicht auftreten. Tabelle 4.5 zeigt die maximale Größe, d. h. die maximale Anzahl der Elemente der minimalen Repräsentantenmengen der widerspruchsfreien Aktionen verschiedener Spezifikationen.

Spezifikation	Anzahl Systemvariablen	Anzahl Knoten in $\mathcal{G}$	max. Größe der min. Repräsentantenmengen
hrmin	2	3	2
mutex2	3	7	3
mutex4	5	31	7
river	4	15	4
prodcons	3	7	3
3phils	6	63	12
5phils	10	1 023	25
7phils	14	16 383	42
10phils	20	1 048 575	?

Tabelle 4.5: Maximale Größe der minimalen Repräsentantenmengen.

4.3 Weitere Verbesserungsansätze

Zusätzlich zu den in den vorherigen beiden Abschnitten beschriebenen Datenstrukturen und Algorithmen wollen wir in diesem Abschnitt kurz auf weitere Verbesserungsansätze für die Berechnung der Aktionen einer Spezifikation eingehen. Dabei wollen wir kurz in das jeweilige Problem einführen und die Lösung und ohne formale Beweise skizzieren. Sofern es möglich ist, den Effekt einer Verbesserung zu isolieren, geben wir außerdem noch Leistungsdaten der Implementierung an.

4.3.1 Disjunktive Normalform umgebungsinvarianter Spezifikationen

Im Zusammenhang mit der schrittbasierten Interleavingsemantik betrachten wir lediglich umgebungsinvariante Spezifikationen. In [Ale05] wurde diese Klasse von Spezifikationen untersucht, konnte jedoch syntaktisch nicht exakt charakterisiert werden. Nur für sog. schwach umgebungsinvariante Schrittformeln konnte ein notwendiges und hinreichendes syntaktisches Kriterium formuliert werden. Die Komponenten der Schrittformeln *Next* der Spezifikationen dieser Arbeit sind meist schwach umgebungsinvariant und haben daher oft die Form $\varphi \triangleq (\widetilde{a} \Rightarrow \psi)$, wobei a eine Menge von Systemvariablen ist, die in ψ gestrichen oder ungestrichen vorkommen.

Um die Aktionen der zugrunde liegenden Spezifikation zu bestimmen, muss φ in disjunktive Normalform überführt werden: $(\neg\widetilde{a} \vee \psi')$. Dabei sei ψ' die disjunktive Normalform von ψ. Bei kompositionalen Spezifikationen enthält eine Aktionen des Gesamtsystemes also entweder $\neg\widetilde{a}$ oder eine Klausel von ψ'. Um diese Aktionen auf Widersprüchlichkeit zu überprüfen, sind insbesondere Informationen über die $\sim$-Variablen von entscheidender Bedeutung (vgl. Abschnitt 4.2). Bei der Umformung in disjunktive Normalform von Aus-

drücken der oben beschriebenen Form gehen jedoch Informationen über die $\sim$-Variable $\widetilde{a}$ in ψ' verloren.

Allerdings kann ψ in den äquivalenten Ausdruck in disjunktiver Normalform umgeformt werden: $(\neg\widetilde{a} \vee (\widetilde{a} \wedge \psi'))$. Alle Klauseln dieses Ausdruckes enthalten nun Informationen über die Involviertheit der Variablenmenge a. Durch diese zusätzliche Regel bei der Umformung in disjunktive Normalform wird die Effektivität der Erkennung widersprüchlicher Klauseln mithilfe $\sim$-Graphen stark erhöht, da durch mehr Informationen über die $\sim$-Variablen widersprüchliche Klauseln evtl. früher entdeckt werden können und so zusammen mit dem on-the-fly-Ansatz die Anzahl der zu überprüfenden Klauseln drastisch verringert wird.

4.3.2 Implizierte Aktionen

Bei umfangreichen Spezifikationen steigt neben der Klauselanzahl der disjunktiven Normalform meist auch die Zahl der Aktionen, die keine Widersprüche enthalten. Dabei kann es vorkommen, dass die Aktionsmenge *Act* Aktionen α_1 und α_2 enthält, von denen eine Aktion die andere impliziert, d. h. es gilt bspw. $\alpha_1 \Rightarrow \alpha_2$. In diesem Fall ist α_1 allgemeiner als α_2, d. h. zu jedem Zustandsübergang $s \xrightarrow{\alpha_2} s'$ gibt es auch einen Zustandsübergang $s \xrightarrow{\alpha_1} s'$. Insbesondere existiert kein Zustand s^*, der nur mit Aktion α_2 nicht jedoch mit α_1 erreicht werden kann. Die konkretere Aktion α_2 ist somit für den Aufbau des Transitionssystemes nicht wesentlich und kann daher verworfen werden.

Da ein vollständiger Test auf Implikation einer Schrittformel durch eine andere nicht effizient, d. h. ohne Belegung der Systemvariablen mit Werten, implementiert werden kann, beschränken wir uns auf ein syntaktisches Kriterium: α_1 und α_2 sind Klauseln einer disjunktiven Normalform, d. h. sie bestehen aus einer Konjunktion von Konjunktionsgliedern. Falls die Menge der Konjunktionsglieder von α_1 eine nichtleere echte Teilmenge der Menge der Konjunktionsglieder von α_2 ist, ist α_1 allgemeiner als α_2, und jeder Schritt, der α_2 erfüllt, erfüllt auch α_1.

Für eine Aktion, die nicht als widersprüchlich erkannt wurde, muss somit überprüft werden, ob die Aktionsmenge *Act* bereits eine allgemeinere Aktion enthält. Nur wenn dies nicht der Fall ist, ist die aktuelle Aktion für den Aufbau des Transitionssystemes wesentlich und wird zu *Act* hinzugefügt.

Für das Konsumenten-/Produzentensystem (s. Abschnitt 5.2) konnten so 198 der 224 widerspruchsfreien Aktionen (88 %) verworfen werden, die ansonsten beim Aufbau des Transitionssystemes berücksichtigt worden wären.

4.3.3 Leere Aktionen

Die schrittbasierte Interleavingsemantik ist ausschließlich für umgebungsinvariante Spezifikationen definiert. Diese Spezifikationen sind invariant bzgl. ihrer Umgebung, d. h. ihr Wahrheitswert hängt nur von der Belegung der Systemvariablen ab. Im Bezug auf

Abläufe bedeutet dies, dass sich die Umgebung stets beliebig mit der Systemvariablen synchronisieren kann und so insbesondere das Schalten beliebiger Transitionen aufgeschoben werden kann (vgl. Abb. 3.5).

Im Transitionssystem einer umgebungsinvarianten Spezifikation Φ existieren demnach stets Aktionen, die keine der Systemvariablen involvieren und den Wert aller Systemvariablen beibehalten. All diese Aktionen werden von einer allgemeinsten Aktion α_0 mit $\alpha_0 \triangleq \bigwedge_{x \in V(\Phi)} \neg\widetilde{x}$, der *leeren Aktion*, impliziert. Fügen wir diese Aktion initial zur Menge der Aktionen *Act* hinzu, können alle konkreteren Aktionen verworfen werden: Falls während der Analyse einer Aktion alle einelementigen Teilmengen der Systemvariablen im $\sim$-Graphen mit *neg* beschriftet werden, kann diese Aktion direkt verworfen werden, ohne sie bis zum Ende zu analysieren, da sie entweder von der leeren Aktion impliziert wird, oder durch eine Fortsetzung mit weiteren Konjunktionsgliedern widersprüchlich wird. Diese Reduktion nutzt die Beobachtungen aus Abschnitt 4.3.2, das on-the-fly-Verfahren und den $\sim$-Graphen aus und kann sehr effizient implementiert werden.

Beispiel 19. Mit der Brute-Force-Implementierung wurden für eine Spezifikation Φ_{mutex2} eines wechselseitigen Auschlusses [Dij65] u. a. folgende Aktionen berechnet:

$$\alpha_0 \triangleq \neg\widetilde{agentL} \wedge \neg\widetilde{agentR} \wedge \neg\widetilde{key}$$
$$\alpha_1 \triangleq \neg\widetilde{agentL} \wedge \neg\widetilde{agentR} \wedge \neg\widetilde{key} \wedge (agentR = \mathsf{quiet})$$
$$\alpha_2 \triangleq \neg\widetilde{agentL} \wedge \neg\widetilde{agentR} \wedge \neg\widetilde{key} \wedge (agentL = \mathsf{quiet})$$
$$\alpha_3 \triangleq \neg\widetilde{agentL} \wedge \neg\widetilde{agentR} \wedge \neg\widetilde{key} \wedge (agentL = \mathsf{quiet}) \wedge (agentR = \mathsf{quiet})$$

Dabei ist die Aktion α_0 die leere Aktion und allgemeiner als die Aktionen α_1–α_3, sodass nur α_0 beibehalten werden muss und α_1–α_3 verworfen werden können. Die Analyse von α_1–α_3 kann dabei jeweils nach Lesen des Konjunktionstermes $\neg\widetilde{key}$ abgebrochen werden. ⌟

4.3.4 Weitere Gründe für Widersprüchlichkeit

Neben den $\sim$-Variablen als Grund für Widersprüche in Aktionen wurden zu Beginn von Abschnitt 4.1 und in Abschnitt 4.2.4 bereits weitere Gründe angegeben, weswegen eine Aktion widersprüchlich sein kann:

- Eine Aktion kann ein in sich widersprüchliches Konjunktionsglied enthalten. Sofern solche Widersprüche syntaktisch zu erkennen sind, kann dies bereits während dem Einlesen der Spezifikation erkannt werden. Typische Beispiele sind Typfehler bei Relationen und Funktionen oder Über- bzw. Unterschreiten von Wertebereichen. Allerdings sind solche Widersprüche tatsächliche Spezifikationsfehler, d. h. entspringen nicht der Umformung in disjunktive Normalform.
- Enthält eine Aktion kein in sich widersprüchliches Konjunktionsglied, kann sie erst durch die Kombination mehrerer Konjunktionsglieder widersprüchlich werden.

Beim Aufbau der Pfade im DNF-Graphen können eine Reihe solcher Widersprüche erkannt werden, indem überprüft wird, ob der aktuelle Knoten gewisse syntaktische Kriterien erfüllt, die die bisher aufgebaute Aktion widersprüchlich werden lässt.

Wird bspw. ein Konjunktionsglied der Form $x = c$, wobei x eine Systemvariable und c eine Konstante ist, zum Pfad hinzugefügt, muss überprüft werden, ob der Pfad bereits Knoten der Form $x \neq c$, $x > c$, $x < c$ oder $x = d$ mit $d \neq c$ enthält. Eine Reihe solcher Tests sind bereits implementiert und sind bei vielen Spezifikationen sehr effizient: Für das Konsumenten-/Produzentensystem (s. Abschnitt 5.2) werden so zusätzliche 78 Aktionen als widersprüchlich erkannt und durch das on-the-fly-Verfahren die Laufzeit dennoch um den Faktor 14 verringert.

- Ein Sonderfall der Widersprüche durch mehrere Konjunktionsglieder sind Klauseln, die für eine Systemvariable x sowohl $\neg\widetilde{x}$ als auch Konjunktionsglieder der Form $x' \neq x$ enthält. Wenn jedoch zu jeder Klausel, die $\neg\widetilde{x}$ enthält, das Konjunktionsglied $x' = x$ hinzugefügt wird, kann die widersprüchliche Involviertheit erkannt werden.

4.3.5 Reihenfolge der Komponenten

Im DNF-Graphen wird jede Aktion durch einen Pfad beschrieben. Die Effizienz des on-the-fly-Verfahrens hängt dabei von der durchschnittlichen Länge dieser Pfade ab, bis ein Widerspruch erkannt wird. Dabei durchläuft jeder Pfad von α nach ω aus jeder Komponente i jeweils eine Klausel der disjunktiven Normalform von $Next_i$. Die Länge des Pfades bis ein Widerspruch erkannt wird, hängt somit von der Reihenfolge ab, in der die Komponenten besucht wurden.

Durch Ausarbeiten von Heuristiken für die Reihenfolge der Komponenten könnte diese durchschnittliche Pfadlänge evtl. verringert werden. Erste Tests ergaben eine Veränderung der durchschnittlichen Pfadlänge bis zu einem gefundenen Widerspruch von $\pm 30\,\%$. Eine nähere Analyse solcher Heuristiken verfolgen wir in dieser Arbeit jedoch nicht.

5 Fallstudien

In [Loh05] wurde die schrittbasierte TLDA-Interleavingsemantik bereits prototypisch implementiert. Dieser Prototyp wurde im Rahmen dieser Arbeit mit den im letzten Kapitel beschriebenen Algorithmen und Datenstrukturen ergänzt. In diesem Kapitel wollen wir die Leistungsfähigkeit der automatischen Konstruktion von Transitionssystemen demonstrieren und legen den Fokus auf die Analyse der Klauseln einer Spezifikation auf Widersprüchlichkeit.

Dazu betrachten wir drei Fallstudien, zu denen wir jeweils in das zugrunde liegende Problem einführen, eine TLDA- bzw. TLDA$^+$-Spezifikation [Loh05] angeben und wichtige Leistungsdaten der einzelnen Algorithmen und Datenstrukturen diskutieren. Dabei untersuchen wir insbesondere das Zusammenspiel aller in Kapitel 4 beschriebenen Verbesserungen. Des Weiteren illustrieren wir noch einmal den Zusammenhang zwischen der Halbordnungssemantik und der schrittbasierten Interleavingsemantik, indem wir Beispielabläufe mit entsprechenden Interleavings sowie die Transitionssysteme der Spezifikationen angeben.

5.1 Wolf, Kohl und Ziege

5.1.1 Problem und Spezifikation

Die erste Fallstudie, das Problem mit Wolf, Kohl und Ziege, ist ein bekanntes Logikrätsel, das bereits im Frühmittelalter von Alkuin von York formuliert wurde:

> Ein Bauer will mit seinem Kohlkopf, einer Ziege und einem Wolf ans andere Ufer eines Flusses. Er hat aber nur ein kleines Boot zur Verfügung, mit dem er nur ein Teil (also Kohl, Ziege oder Wolf) mitnehmen kann. Er muss also mehrmals übersetzen. Das Problem ist allerdings, dass die Ziege ohne den Bauern den Kohl fräße und der Wolf sich über die Ziege hermachte, wenn der Bauer sie aus den Augen ließe.
>
> Wie muss der Bauer die Überfahrten gestalten, dass für keinen seiner Besitztümer eine Gefahr droht?

Das Problem kann als verteilter Algorithmus mit vier Komponenten (Bauer, Kohlkopf, Ziege und Wolf) aufgefasst werden. Wir modellieren jede der Komponenten als eine Variable mit dem Wertebereich $\{\mathsf{west}, \mathsf{ost}\}$. Dabei müssen die Zustandsübergänge (von west nach ost bzw. umgekehrt) der einzelnen Komponenten so synchronisiert werden, dass sie die Anforderungen (z. B. niemals Ziege mit Kohlkopf unbeaufsichtigt lassen) erfüllen.

Abbildung 5.1 zeigt eine kommentierte TLDA$^+$-Spezifikation Φ_{fluss} des Flussproblemes.

```
VARIABLES
  bauer \in {"ost", "west"},
  ziege \in {"ost", "west"},
  wolf \in {"ost", "west"},
  kohl \in {"ost", "west"}

// Anfangs stehen alle auf der Ostseite des Flusses.
bauer = "ost" /\ ziege = "ost" /\ wolf = "ost" /\ kohl = "ost"

// Mit dem Boot kommt man auf die andere Flussseite.
/\ [](bauer~ => bauer' != bauer)
/\ [](ziege~ => ziege' != ziege)
/\ [](wolf~ => wolf' != wolf)
/\ [](kohl~ => kohl' != kohl)

// Nur der Bauer kann das Boot steuern.
/\ [](ziege~ => {ziege,bauer}~)
/\ [](wolf~ => {wolf,bauer}~)
/\ [](kohl~ => {kohl,bauer}~)

// Wer das Boot benutzen will, muss auf der gleichen Seite wie der Bauer stehen.
/\ []({ziege,bauer}~ => ziege=bauer)
/\ [](({wolf,bauer}~ => wolf=bauer)
/\ [](({kohl,bauer}~ => kohl=bauer)

// Das Boot kann nur den Bauern und ein weiteres Ding transportieren.
/\ [] !{bauer,ziege,wolf}~
/\ [] !{bauer,ziege,kohl}~
/\ [] !{bauer,wolf,kohl}~

// Ziege und Kohl dürfen nicht alleine gelassen werden.
/\ [](({bauer,wolf}~ => ziege != kohl)

// Ziege und Wolf dürfen nicht alleine gelassen werden.
/\ [](({bauer,kohl}~ => wolf != ziege)

// Der Bauer darf nur alleine fahren, wenn er eine sichere Situation hinterlässt.
/\ []( (bauer~ /\ !{bauer,ziege}~ /\ !{bauer,wolf}~ /\ !{bauer,kohl}~)
        => !(kohl=ziege \/ ziege=wolf) )
```

Abbildung 5.1: TLDA$^+$-Spezifikation für das Flussproblem. Die Spezifikation enthält vier Systemvariablen und 16 Komponenten.

5.1.2 Benchmarks

Bevor wir Abläufe, Interleavings und das Transitionssystem des Flussproblemes eingehen, wollen wir zunächst auf die Ergebnisse der Analyse der Aktionen eingehen. Dabei zeigt sich, dass mit dem DNF-Graphen definierte on-the-fly-Verfahren eine zentrale Rolle spielt:

Die Pfade im DNF-Graphen der Spezifikation haben eine Maximallänge von 16 Knoten (wegen der 16 Komponenten), jedoch wurden widersprüchliche Klauseln durchschnittlich nach Analyse von 8,77 Konjunktionsgliedern entdeckt. So erlaubt das on-the-fly-Verfahren, die Berechnung der Aktionen schon nach Analyse von nur 99 der 20 480 Klauseln (0,483 %) und weniger als einer Sekunde abzuschließen. Dadurch ergibt sich

eine enorme Leistungssteigerung gegenüber dem Brute-Force-Prototypen, der für die Analyse derselben Spezifikation mehrere Minuten benötigt.

Von den analysierten 99 Klauseln können 94 verworfen werden: 91 Klauseln sind widersprüchlich und drei Klauseln wurden von anderen impliziert. Insgesamt sind nur fünf Klauseln bzw. Aktionen für das Transitionssystem $\mathcal{TS}_{\Phi_{\text{fluss}}}$ wesentlich:

$$
\begin{aligned}
\alpha_0 \triangleq\ & \neg\widetilde{bauer} \wedge \neg\widetilde{ziege} \wedge \neg\widetilde{wolf} \wedge \neg\widetilde{kohl},\\
\alpha_1 \triangleq\ & \widetilde{bauer} \wedge \neg\widetilde{ziege} \wedge \neg\widetilde{wolf} \wedge \neg\widetilde{kohl}\\
& \wedge bauer' \neq bauer \wedge kohl \neq ziege \wedge ziege \neq wolf,\\
\alpha_2 \triangleq\ & \widetilde{bauerkohl} \wedge \neg\widetilde{ziege} \wedge \neg\widetilde{wolf}\\
& \wedge bauer = kohl \wedge ziege \neq wolf \wedge bauer' \neq bauer \wedge kohl' \neq kohl,\\
\alpha_3 \triangleq\ & \widetilde{bauerwolf} \wedge \neg\widetilde{ziege} \wedge \neg\widetilde{kohl}\\
& \wedge bauer = wolf \wedge ziege \neq kohl \wedge bauer' \neq bauer \wedge wolf' \neq wolf,\\
\alpha_4 \triangleq\ & \widetilde{bauerziege} \wedge \neg\widetilde{wolf} \wedge \neg\widetilde{kohl}\\
& \wedge bauer = ziege \wedge bauer' \neq bauer \wedge ziege' \neq ziege.
\end{aligned}
$$

5.1.3 Abläufe und Interleavings

Für das Flussproblem gibt es mehrere Lösungen (d. h. mehrere Sequenzen von Überfahrten). Abbildung 5.2 zeigt einen Ablauf von Φ_{fluss}, der eine Lösung beschreibt, d. h. einen Schnitt C_7 erreicht mit $H_x(C_7(x)) = \mathsf{west}$ für alle Systemvariablen $V(\Phi_{\text{fluss}})$.

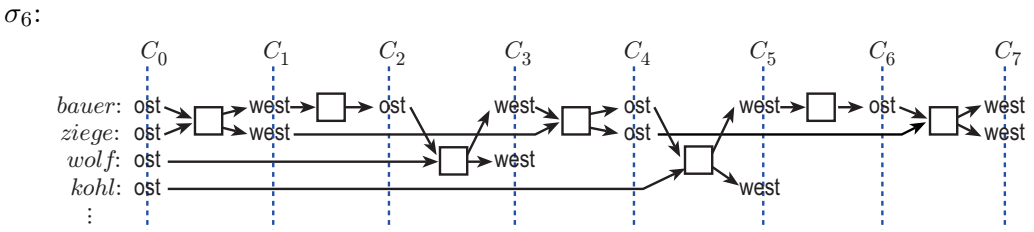


Abbildung 5.2: Ablauf einer Lösung des Flussproblemes. Die Schnitte sind vereinfacht dargestellt.

In σ_6 erfüllt jeder Schritt genau eine Aktion: $S_{C_0} \Vdash \alpha_4$, $S_{C_1} \Vdash \alpha_1$, $S_{C_2} \Vdash \alpha_3$, $S_{C_3} \Vdash \alpha_4$, $S_{C_4} \Vdash \alpha_2$, $S_{C_5} \Vdash \alpha_1$ und $S_{C_6} \Vdash \alpha_4$. Zu Ablauf σ_6 gibt es nach Korollar 3.5 in $\mathcal{TS}_{\Phi_{\text{fluss}}}$ das in Abbildung 5.3 gezeigte Interleaving.

Der Zielzustand $[\mathsf{west}, \mathsf{west}, \mathsf{west}, \mathsf{west}]$ wird ebenfalls erreicht, wenn im beschriebenen Interleaving die Aktionen α_2 und α_3 vertauscht werden. Abbildung 5.4 zeigt das Transitionssystem des Flussproblemes. Dabei ist der Anfangszustand fett gezeichnet und die Wertetupel haben dieselbe Variablenordnung wie in der Spezifikation.

$$\begin{array}{l} \textit{bauer}: \\ \textit{ziege}: \\ \textit{wolf}: \\ \textit{kohl}: \end{array} \begin{bmatrix} \text{ost} \\ \text{ost} \\ \text{ost} \\ \text{ost} \end{bmatrix} \xrightarrow{\alpha_4} \begin{bmatrix} \text{west} \\ \text{west} \\ \text{ost} \\ \text{ost} \end{bmatrix} \xrightarrow{\alpha_1} \begin{bmatrix} \text{ost} \\ \text{west} \\ \text{ost} \\ \text{ost} \end{bmatrix} \xrightarrow{\alpha_3} \begin{bmatrix} \text{west} \\ \text{west} \\ \text{west} \\ \text{ost} \end{bmatrix} \xrightarrow{\alpha_4} \begin{bmatrix} \text{ost} \\ \text{ost} \\ \text{west} \\ \text{ost} \end{bmatrix} \xrightarrow{\alpha_2} \begin{bmatrix} \text{west} \\ \text{ost} \\ \text{west} \\ \text{west} \end{bmatrix} \xrightarrow{\alpha_1} \begin{bmatrix} \text{ost} \\ \text{ost} \\ \text{west} \\ \text{west} \end{bmatrix} \xrightarrow{\alpha_4} \begin{bmatrix} \text{west} \\ \text{west} \\ \text{west} \\ \text{west} \end{bmatrix}$$

Abbildung 5.3: Interleaving für eine Lösung des Flussproblemes.

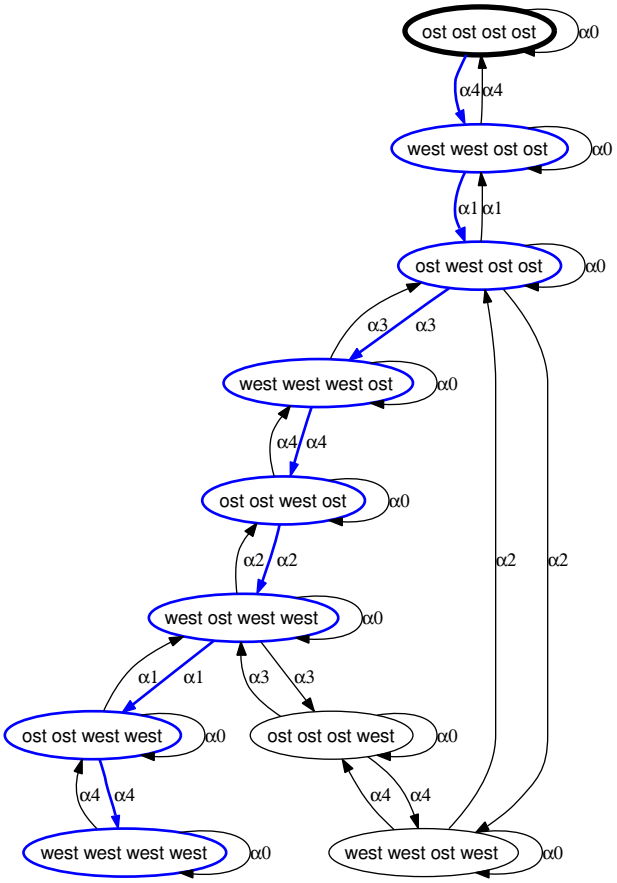


Abbildung 5.4: Transitionssystem des Flussproblemes.

5.2 Produzenten-/Konsumentensystem

5.2.1 Problem und Spezifikation

Als zweite Fallstudie betrachten wir ein Konsumenten-/Produzentensystem (vgl. [Rei98]) mit einem Puffer. In [Ale04] wird das System wie folgt beschrieben:

> The producer spontaneously produces an item and retains it as long as the buffer is not empty. After the buffer gets empty, the item can be delivered to the buffer

> and the producer is ready to produce a new item. If there is an item in the buffer, the consumer can remove it from the buffer. Afterwards, he eventually consumes the item and is then ready to remove a new item from the buffer.

Wie in Abschnitt 4.1.1 beschrieben, kann das System in mehrere Komponenten dekomponiert werden, die einzeln spezifiziert werden. Die Komponenten des Systemes agieren dabei teilweise nebenläufig und teilweise synchronisiert:

Die Zustandsübergänge "ready to deliver"→"ready to produce" des Produzenten und $0 \to 1$ des Puffers müssen dabei synchronisiert stattfinden. Ebenso soll der Übergang "ready to remove"→"ready to consume" des Konsumenten und $1 \to 0$ des Puffers synchron ablaufen.

Wir übernehmen folgende Spezifikation $\Phi_{\text{prodcons}} \triangleq \mathit{Producer} \wedge \mathit{Consumer} \wedge \mathit{Buffer} \wedge \mathit{Sync}$ aus [Ale04] mit

$$
\begin{aligned}
\mathit{Producer} &\triangleq \mathit{prod} = \mathsf{rp} \wedge \Box(\widetilde{\mathit{prod}} \Rightarrow \mathit{Produce} \vee \mathit{Deliver}),\\
\mathit{Produce} &\triangleq \mathit{prod} = \mathsf{rp} \wedge \mathit{prod}' = \mathsf{rd},\\
\mathit{Deliver} &\triangleq \mathit{prod} = \mathsf{rd} \wedge \mathit{prod}' = \mathsf{rp},\\
\mathit{Consumer} &\triangleq \mathit{cons} = \mathsf{rr} \wedge \Box(\widetilde{\mathit{cons}} \Rightarrow \mathit{Remove} \vee \mathit{Consume}),\\
\mathit{Remove} &\triangleq \mathit{cons} = \mathsf{rr} \wedge \mathit{cons}' = \mathsf{rc},\\
\mathit{Consume} &\triangleq \mathit{cons} = \mathsf{rc} \wedge \mathit{cons}' = \mathsf{rr},\\
\mathit{Buffer} &\triangleq \mathit{buff} = 0 \wedge \Box(\widetilde{\mathit{buff}} \Rightarrow \mathit{Filling} \vee \mathit{Emptying}),\\
\mathit{Filling} &\triangleq \mathit{buff} = 0 \wedge \mathit{buff}' = 1,\\
\mathit{Emptying} &\triangleq \mathit{buff} = 1 \wedge \mathit{buff}' = 0,\\
\mathit{Sync} &\triangleq \quad \Box\neg\widetilde{\mathit{prodcons}}\\
&\qquad \wedge \Box(\mathit{Deliver} \Leftrightarrow \mathit{Filling}) \wedge \Box(\mathit{Remove} \Leftrightarrow \mathit{Emptying})\\
&\qquad \wedge \Box(\mathit{Filling} \Leftrightarrow \widetilde{\mathit{prodbuff}}) \wedge \Box(\mathit{Emptying} \Leftrightarrow \widetilde{\mathit{consbuff}}).
\end{aligned}
$$

Dabei ignorieren wir die dort angegebenen Lebendigkeitseigenschaften. Um diese Spezifikation computergestützt verarbeiten zu können, müssen wir sie in Normalform bringen. Abbildung 5.5 zeigt eine TLDA$^+$-Variante von Φ_{prodcons}.

5.2.2 Abläufe und Interleavings

Die Restriktion eines unendlichen Ablaufes σ_7 des Produzenten-/Konsumentensystemes auf die Systemvariablen zeigt Abbildung 5.6.

Wie im Beweis zu Satz 3.2 beschrieben, kann durch Synchronisation mit der Umgebung ein Ablauf σ_8 (vgl. Abb 5.7) konstruiert werden, in dem der Schnitt C^* durch Schritte vom Anfangsschnitt aus erreichbar ist. Außerdem gilt $\sigma_{7|V(\Phi_{\text{prodcons}})} = \sigma_{8|V(\Phi_{\text{prodcons}})}$ und $\sigma_8 \models \Phi_{\text{prodcons}}$.

```
VARIABLES
  prod \in {"rp", "rd"},
  cons \in {"rr", "rc"},
  buff \in {0, 1}

prod="rp" /\ cons="rr" /\ buff=0

/\ [](prod~ => ((prod="rp" /\ prod'="rd") \/ (prod="rd" /\ prod'="rp")))

/\ [](cons~ => ((cons="rr" /\ cons'="rc") \/ (cons="rc" /\ cons'="rr")))

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

/\ [](!{prod,cons}~)
/\ [](  (prod="rd" /\ prod'="rp") <=> (buff=0 /\ buff'=1) )
/\ [](  (cons="rr" /\ cons'="rc") <=> (buff=1 /\ buff'=0) )
/\ [](  (buff=0 /\ buff'=1) <=> {prod,buff}~ )
/\ [](  (buff=1 /\ buff'=0) <=> {cons,buff}~ )
```

Abbildung 5.5: TLDA$^+$-Spezifikation für das Produzenten-/Konsumentensystem.

$\sigma_{7|V(\Phi_{\text{prodcons}})}$:

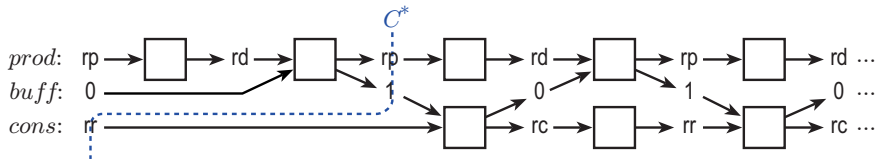


Abbildung 5.6: Unendlicher Ablauf des Produzenten-/Konsumentensystemes.

σ_8:

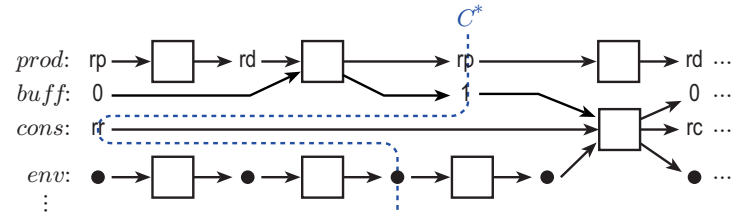


Abbildung 5.7: Ein Ablauf des Produzenten-/Konsumentensystemes mit Umgebungsvariable *env*.

Zwei Interleavings des Produzenten-/Konsumentensystemes sind in Abbildung 5.8 abgebildet. Dabei entspricht Interleaving (a) Ablauf σ_7 ohne Synchronisation mit der Umgebung und Interleaving (b) die in Abbildung 5.7 beschriebene Situation.

Abbildung 5.9 zeigt das Transitionssystem des Produzenten-/Konsumentensystemes.

(a)

$$\begin{matrix} prod: \\ cons: \\ buff: \end{matrix} \begin{bmatrix} \mathsf{rp} \\ \mathsf{rr} \\ 0 \end{bmatrix} \xrightarrow{\alpha_{17}} \begin{bmatrix} \mathsf{rd} \\ \mathsf{rr} \\ 0 \end{bmatrix} \xrightarrow{\alpha_{25}} \begin{bmatrix} \mathsf{rp} \\ \mathsf{rr} \\ 1 \end{bmatrix} \xrightarrow{\alpha_{19}} \begin{bmatrix} \mathsf{rd} \\ \mathsf{rc} \\ 0 \end{bmatrix} \xrightarrow{\alpha_{26}} \begin{bmatrix} \mathsf{rp} \\ \mathsf{rr} \\ 1 \end{bmatrix} \xrightarrow{\alpha_{19}} \begin{bmatrix} \mathsf{rd} \\ \mathsf{rc} \\ 0 \end{bmatrix} \ldots$$

(b)

$$\begin{matrix} prod: \\ cons: \\ buff: \end{matrix} \begin{bmatrix} \mathsf{rp} \\ \mathsf{rr} \\ 0 \end{bmatrix} \xrightarrow{\alpha_{17}} \begin{bmatrix} \mathsf{rd} \\ \mathsf{rr} \\ 0 \end{bmatrix} \xrightarrow{\alpha_{25}} \begin{bmatrix} \mathsf{rp} \\ \mathsf{rr} \\ 1 \end{bmatrix} \xrightarrow{\alpha_{2}} \begin{bmatrix} \mathsf{rd} \\ \mathsf{rc} \\ 0 \end{bmatrix} \ldots$$

Abbildung 5.8: Interleavings des Produzenten-/Konsumentensystemes.

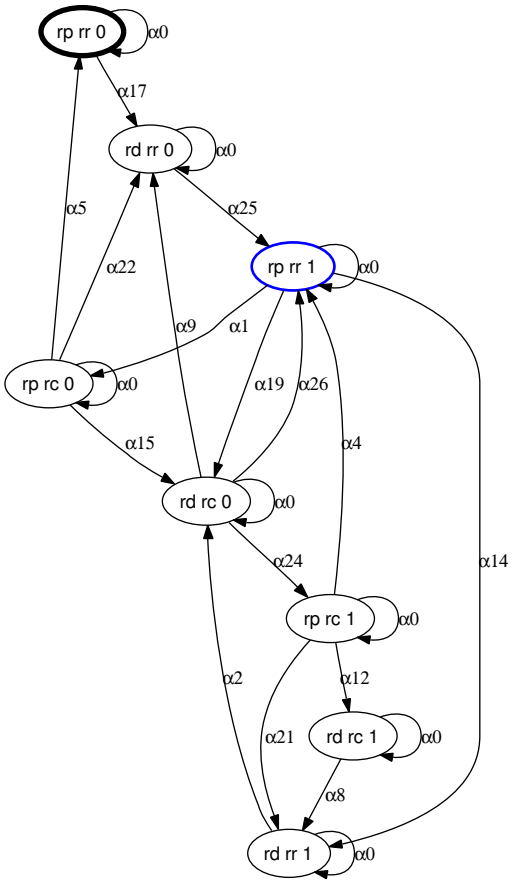


Abbildung 5.9: Transitionssystem des Produzenten-/Konsumentensystemes.

5.2.3 Benchmarks

Die Spezifikation ist im Vergleich zum Flussproblem des vorherigen Abschnittes komplexer. Sie besteht zwar „nur“ aus 16 875 Klauseln, allerdings muss davon ein größerer Anteil (1 407, d. h. 8,337 %) analysiert werden. Von diesen Klauseln werden 1 381 aus unterschiedlichen Gründen verworfen:

- 577 Klauseln enthalten widersprüchliche Informationsmengen über die $\sim$-Variablen der Spezifikation, die mithilfe des $\sim$-Graphen erkannt wurden.

- 606 Klauseln sind widersprüchlich, obwohl ihre Informationsmengen konsistent sind. Mit den in Abschnitt 4.3.4 vorgestellten Techniken konnten jedoch Widersprüche der Form $x \neq a \wedge x = a$ erkannt werden.
- 198 Klauseln werden von anderen Klauseln impliziert (s. Abschnitt 4.3.2) und können daher ignoriert werden.

Dieses Beispiel zeigt, wie erst das Zusammenspiel der in Kapitel 4 vorgestellten Algorithmen alle widersprüchlichen Klauseln der Spezifikation erkennen kann.

Bereits dieses Beispiel mit einer noch recht einfachen Spezifikation konnte mit dem Brute-Force-Prototypen nicht analysiert werden, da durch die Abwesenheit des on-the-fly-Verfahrens und effizienter Analyse der $\sim$-Variablen für zu viele Aktionen Quellcode generiert wurde, der nicht übersetzt werden konnte.

Der Rechenaufwand kann jedoch durch eine alternative Spezifizierung des Produzenten-/Konsumentensystemes weiter verringert werden: Bei der angegebenen Spezifikation aus Abbildung 5.5 handelt es sich um eine direkte Übersetzung der Spezifikation aus [Ale04], die jedoch für den manuellen Beweis von Eigenschaften und nicht für die maschinelle Verarbeitung formuliert wurde und viele Redundanzen enthält. Werden diese Redundanzen entfernt und die Wertebereiche der Systemvariablen berücksichtigt, ergibt sich eine kürzere und einfachere Spezifikation (vgl. Abb 5.10).

```
VARIABLES
  prod \in {"rp", "rd"},
  buff \in {0, 1},
  cons \in {"rr", "rc"}

prod = "rp" /\ cons = "rr" /\ buff = 0

/\ [](prod~ => prod' != prod)
/\ [](cons~ => cons' != cons)
/\ [](buff~ => buff' != buff)

/\ [](prod~ => ({prod,buff}~ /\ buff = 0) \/ prod = "rp")
/\ [](cons~ => ({cons,buff}~ /\ buff = 1) \/ cons = "rc")
/\ [](buff~ => ({buff,prod}~ /\ prod = "rd") \/ ({buff,cons}~ /\ cons = "rr"))
```

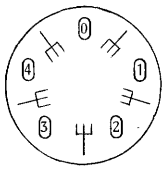
Abbildung 5.10: Eine alternative TLDA$^+$-Spezifikation für das Produzenten-/Konsumentensystem.

Ihre disjunktive Normalform besteht aus lediglich 216 Klauseln, von denen 62 analysiert und 55 verworfen werden müssen. Die resultierenden Aktionen reichen aus, um ein zu Abbildung 5.9 äquivalentes Transitionssystem zu konstruieren. Ob eine solche Umformulierung stets möglich ist und welcher Rechenaufwand dafür nötig ist, wird in dieser Arbeit nicht weiter betrachtet.

5.3 Die speisenden Philosophen

5.3.1 Problem und Spezifikation

Die dritte Fallstudie, die wir betrachten, ist das Problem der speisenden Philosophen. Es wurde zuerst 1971 von Edsger W. Dijskstra [Dij71] wie folgt formuliert:

> Five philosophers, numbered from 0 through 4 are living in a house where the table laid for them, each philosopher having his own place at the table:
>
> 
>
> Their only problem—besides those of philosophy—is that the dish served is a very difficult kind of spaghetti, that has to be eaten with two forks. There are two forks next to each plate, so that presents no difficulty: as a consequence, however, no two neighbours may be eating simultaneously.

Das beschriebene Szenario ist ein anschauliches Beispiel für das Problem der verteilten Ressourcenverwaltung, das bei verteilten Algorithmen auftreten kann. In [Mas04] wurden verschiedene Spezifikationen für das Philosophenproblem angegeben. Wie verwenden die folgende Spezifikation $\Phi_{5\text{phils}} \triangleq \mathit{Init}_{5\text{phils}} \wedge \Box \mathit{Next}_{5\text{phils}}$ mit

$$
\begin{aligned}
\mathit{Init}_{5\text{phils}} &\triangleq \textstyle\bigwedge_{i\in\{0,\ldots,4\}} (p_i = \mathsf{d} \wedge g_i = \mathsf{v}),\\
\mathit{Next}_{5\text{phils}} &\triangleq \quad (\widetilde{p_i} \Rightarrow \mathit{WerdeH}_i \vee \mathit{WerdeE}_i \vee \mathit{WerdeD}_i\\
&\qquad \wedge\ \widetilde{p_i} \Rightarrow \textstyle\bigwedge_{j\in\{0,\ldots,4\}, j\neq i} \neg\widetilde{p_i p_j}\\
&\qquad \wedge\ \widetilde{g_{i+1}} \Rightarrow \mathit{WerdeE}_i \vee \mathit{WerdeE}_{i+1} \vee \mathit{WerdeD}_i \vee \mathit{WerdeD}_{i+1}\\
&\qquad \wedge\ \textstyle\bigwedge_{j\in\{0,\ldots,4\}, j\neq i} (\widetilde{g_i g_j} \Rightarrow i = j+1 \vee j = i+1)),\\
\mathit{WerdeH}_i &\triangleq p_i = \mathsf{d} \wedge p_i' = \mathsf{h},\\
\mathit{WerdeE}_i &\triangleq p_i = \mathsf{h} \wedge p_i' = \mathsf{e} \wedge g_i = g_{i+1} = \mathsf{v} \wedge g_i' = g_{i+1}' = \mathsf{b} \wedge \widetilde{g_i p_i g_{i+1}},\\
\mathit{WerdeD}_i &\triangleq p_i = \mathsf{e} \wedge p_i' = \mathsf{d} \wedge g_i = g_{i+1} = \mathsf{b} \wedge g_i' = g_{i+1}' = \mathsf{v} \wedge \widetilde{g_i p_i g_{i+1}}.
\end{aligned}
$$

Dabei ignorieren wir die dort angegebene Lebendigkeitseigenschaften. Die Anzahl der Philosophen in der Spezifikation lässt sich leicht anpassen. Um die Illustrationen der Abläufe, Interleavings und nicht zuletzt des Transitionssystemes übersichtlich zu halten, betrachten wir in nächsten Abschnitt das Philosophenproblem mit drei Philosophen. Abbildung 5.11 zeigt die TLDA$^+$-Spezifikation für drei Philosophen.

5.3.2 Abläufe und Interleavings

Abb. 5.12 zeigt die Restriktion eines Ablaufes von $\Phi_{3\text{phils}}$ für drei Philosophen auf die Systemvariablen. Ein resultierendes Interleaving zeigt Abb. 5.13.

```
VARIABLES
  p0 \in {"d", "h", "e"},
  p1 \in {"d", "h", "e"},
  p2 \in {"d", "h", "e"},
  g0 \in {"v", "b"},
  g1 \in {"v", "b"},
  g2 \in {"v", "b"}

p0="d" /\ p1="d" /\ p2="d" /\ g0="v" /\ g1="v" /\ g2="v"

/\ [] ( p0~ => ( (p0="d" /\ p0'="h")
              \/ (p0="h" /\ p0'="e" /\ g0="v" /\ g1="v" /\ g0'="b" /\ g1'="b" /\ {g0,p0,g1}~)
              \/ (p0="e" /\ p0'="d" /\ g0="b" /\ g1="b" /\ g0'="v" /\ g1'="v" /\ {g0,p0,g1}~) ) )

/\ [] ( p1~ => ( (p1="d" /\ p1'="h")
              \/ (p1="h" /\ p1'="e" /\ g1="v" /\ g2="v" /\ g1'="b" /\ g2'="b" /\ {g1,p1,g2}~)
              \/ (p1="e" /\ p1'="d" /\ g1="b" /\ g2="b" /\ g1'="v" /\ g2'="v" /\ {g1,p1,g2}~) ) )

/\ [] ( p2~ => ( (p2="d" /\ p2'="h")
              \/ (p2="h" /\ p2'="e" /\ g2="v" /\ g0="v" /\ g2'="b" /\ g0'="b" /\ {g2,p2,g0}~)
              \/ (p2="e" /\ p2'="d" /\ g2="b" /\ g0="b" /\ g2'="v" /\ g0'="v" /\ {g2,p2,g0}~) ) )

/\ [] ( p0~ => (!{p0,p1}~ /\ !{p0,p2}~) )
/\ [] ( p1~ => (!{p1,p0}~ /\ !{p1,p2}~) )
/\ [] ( p2~ => (!{p2,p0}~ /\ !{p2,p1}~) )

/\ [] ( g1~ => ( (p0="h" /\ p0'="e" /\ g0="v" /\ g1="v" /\ g0'="b" /\ g1'="b" /\ {g0,p0,g1}~)
              \/ (p1="h" /\ p1'="e" /\ g1="v" /\ g2="v" /\ g1'="b" /\ g2'="b" /\ {g1,p1,g2}~)
              \/ (p0="e" /\ p0'="d" /\ g0="b" /\ g1="b" /\ g0'="v" /\ g1'="v" /\ {g0,p0,g1}~)
              \/ (p1="e" /\ p1'="d" /\ g1="b" /\ g2="b" /\ g1'="v" /\ g2'="v" /\ {g1,p1,g2}~) ) )

/\ [] ( g2~ => ( (p1="h" /\ p1'="e" /\ g1="v" /\ g2="v" /\ g1'="b" /\ g2'="b" /\ {g1,p1,g2}~)
              \/ (p2="h" /\ p2'="e" /\ g2="v" /\ g0="v" /\ g2'="b" /\ g0'="b" /\ {g2,p2,g0}~)
              \/ (p1="e" /\ p1'="d" /\ g1="b" /\ g2="b" /\ g1'="v" /\ g2'="v" /\ {g1,p1,g2}~)
              \/ (p2="e" /\ p2'="d" /\ g2="b" /\ g0="b" /\ g2'="v" /\ g0'="v" /\ {g2,p2,g0}~) ) )

/\ [] ( g0~ => ( (p2="h" /\ p2'="e" /\ g2="v" /\ g0="v" /\ g2'="b" /\ g0'="b" /\ {g2,p2,g0}~)
              \/ (p0="h" /\ p0'="e" /\ g0="v" /\ g1="v" /\ g0'="b" /\ g1'="b" /\ {g0,p0,g1}~)
              \/ (p2="e" /\ p2'="d" /\ g2="b" /\ g0="b" /\ g2'="v" /\ g0'="v" /\ {g2,p2,g0}~)
              \/ (p0="e" /\ p0'="d" /\ g0="b" /\ g1="b" /\ g0'="v" /\ g1'="v" /\ {g0,p0,g1}~) ) )
```

Abbildung 5.11: TLDA$^+$-Spezifikation für das Philosophenproblem mit drei Philosophen. Die Spezifikation enthält sechs Systemvariablen und neun Komponenten.

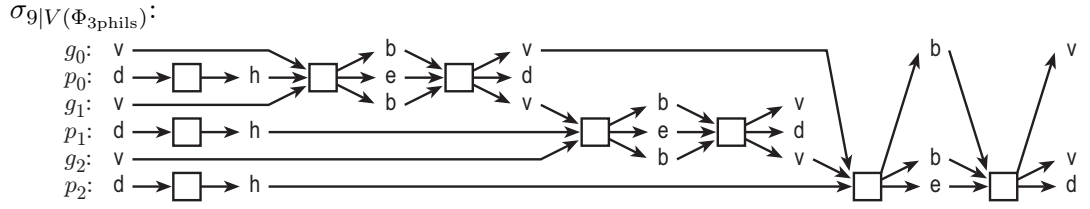


Abbildung 5.12: Ein Ablauf der speisenden Philosophen.

Abb. 5.14 zeigt das Transitionssystem der Philosophenproblemes mit drei Philosophen.

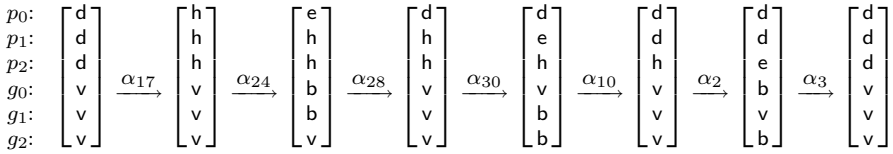


Abbildung 5.13: Interleaving der speisenden Philosophen.

5.3.3 Benchmarks

Da sich die Zahl der Philosophen in der Spezifikation leicht anpassen lässt, können wir verschiedene Instanzen des Philosophenproblemes betrachten. Dabei wächst die Größe der Spezifikation exponentiell mit der Anzahl der Philosophen. Tabelle 5.1 zeigt eine Übersicht über die betrachteten Leistungsdaten.

	3 Philosophen	5 Philosophen	7 Philosophen	10 Philosophen
Systemvariablen	6	10	14	20
Größe $\sim$-Graph	63	1 023	16 383	1 048 575
Größe DNF-Graph	33	60	84	120
Komponenten	9	20	28	40
Durchschnittslänge	7,10	15,17	?	?
Aktionen	64 000	102 400 000	631 242 752	1 073 741 824
Aktionen analysiert	556	9 984	?	?
Aktionen behalten	31	351	?	?
Rechenzeit	1 Sekunde	6 Minuten	$\approx$ 55 Stunden	?
Speicherauslastung	2 MB	4 MB	30 MB	$\gg$ 1,5 GB

Tabelle 5.1: Benchmarks für das Philosophenproblem.

Systeme mit mehr als sieben Philosophen können nicht vollständig analysiert werden, da sowohl die Laufzeit enorm ansteigt (mehr als 55 Stunden für sieben Philosophen), als auch der Speicherplatz für eine Analyse der Spezifikation nicht mehr ausreicht: Für zehn Philosophen benötigt alleine die Konstruktion des $\sim$-Graphen mehrere Minuten und mehr als ein Gigabyte Speicherplatz.

Das Philosophenproblem zeigt klar die Grenzen der vorgestellten Algorithmen: Durch den enormen Anstieg der Klauselanzahl ist selbst die on-the-fly-Lösung nicht mehr effizient genug. Wir werden die Ergebnisse der Fallstudien in Abschnitt 6.2 diskutieren.

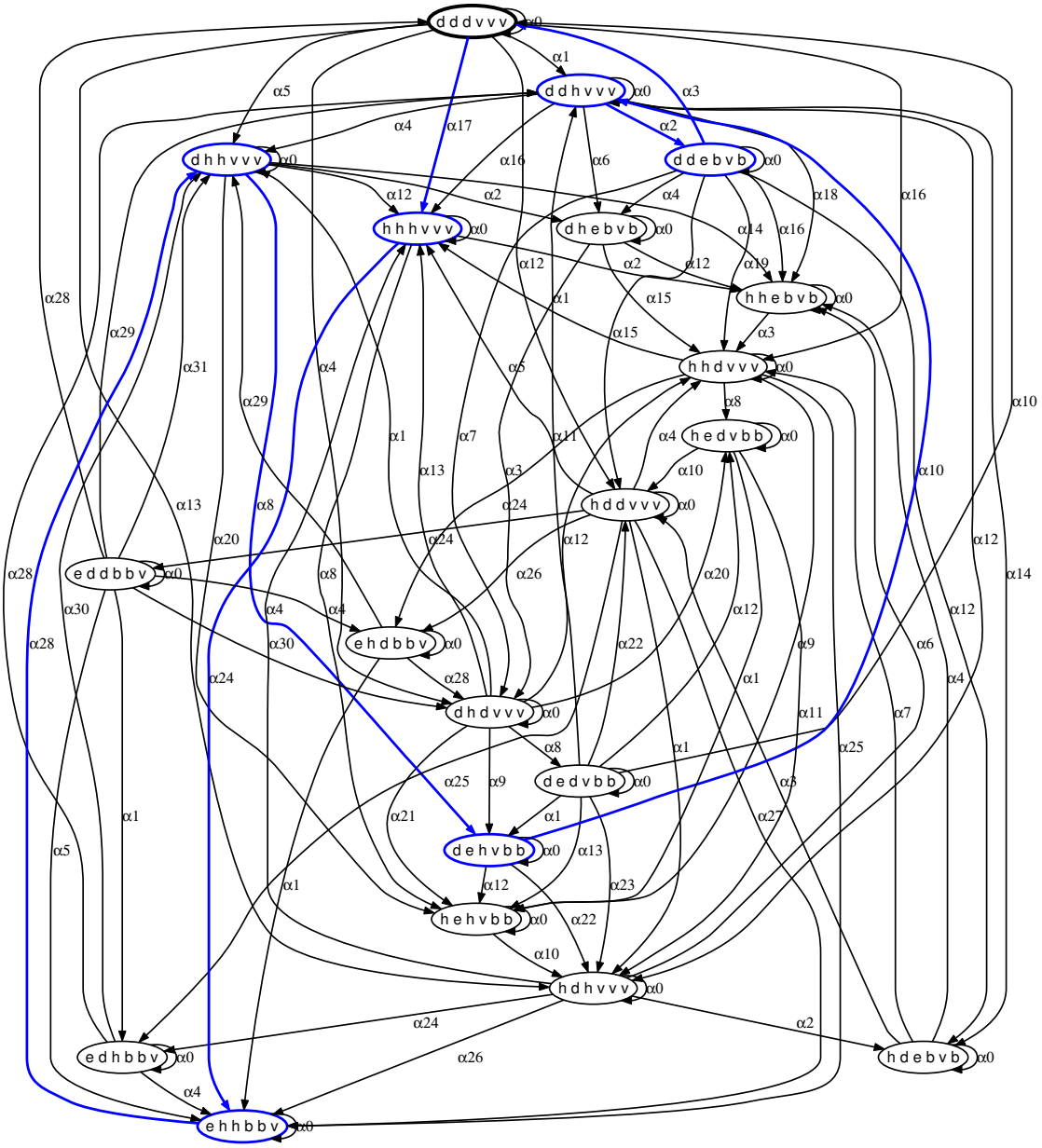


Abbildung 5.14: Transitionssystem der speisenden Philosophen.

6 Zusammenfassung und Ausblick

6.1 Zusammenfassung

Ziel dieser Arbeit war es, eine formale Fundierung für die in [Loh05] definierte schrittbasierte TLDA-Interleavingsemantik zu geben und den bestehenden Prototyp durch effizientere Algorithmen zu erweitern.

Dazu untersuchten wir in Kapitel 3 den Zusammenhang zwischen dem Transitionssystem einer Spezifikation und der Menge seiner Abläufe. Wir konnten zeigen, dass für die Klasse der umgebungsinvarianten Spezifikationen das Transitionssystem als semantisches Modell aufgefasst werden kann. Spezifizierte Systeme können somit anhand der Pfade des Transitionssystemes, d. h. der Menge der Interleavings, untersucht werden. Weiterhin zeigten wir, dass dieser Zusammenhang auch durch die Reduktion des Transitionssystemes erhalten bleibt. Für Systeme mit endlichen Wertebereichen der Systemvariablen kann so ein endliches Transitionssystem konstruiert werden, das Grundlage für die Entwicklung von expliziten Modelchecking-Algorithmen für die Logik TLDA ist.

Durch den gewählten Aktionsbegriff der schrittbasierten Interleavingsemantik und die Eigenschaften von TLDA ergaben sich jedoch mehrere Probleme, die die Leistungsfähigkeit der prototypischen Brute-Force-Implementierung stark einschränkten: Nur für einfache Spezifikationen konnte die Menge der Aktionen analysiert werden.

In Kapitel 4 untersuchten wir die Probleme, die durch die Umformung in disjunktive Normalform entstanden. Wir konnten dabei in mehreren Fällen eine Struktur in den Probleme ausarbeiten und ausnutzen. So half der spezielle Aufbau komponierter Spezifikationen bei der Definition des DNF-Graphen – einer Datenstruktur, mit der mit geringerem Rechen- und Speicheraufwand Klauseln der disjunktiven Normalform des Gesamtsystemes berechnet werden können. Weiterhin konnten wir den Aufbau der Klauseln ausnutzen und ein on-the-fly-Verfahren entwickeln, mit dem die Analyse der Aktionen eines Systemes in der Regel früher abgebrochen und die Anzahl der zu berechnenden Aktionen meist drastisch verringern werden konnte.

Bei der Entdeckung widersprüchlicher Klauseln gingen wir auch auf die hervorgehobene Rolle der $\sim$-Variablen ein und zeigten, dass sie durch Mengenbeziehungen hierarchisch geordnet sind. Diese Hierarchie ließen wir in die Konstruktion des $\sim$-Graphen einfließen, mit dem wir eine effiziente Datenstruktur für die Informationsberechnung und -speicherung entwickelt haben.

Zuletzt untersuchten wir in Kapitel 5 die Leistungsfähigkeit der entwickelten Algorithmen und Datenstrukturen, indem wir mehrere Fallstudien mit dem erweiterten Prototypen untersuchten und die resultierenden Leistungsdaten diskutierten. Dabei zeigte sich,

dass insbesondere das on-the-fly-Verfahren eine große Leistungssteigerung gegenüber der ersten Brute-Force-Implementierung ermöglicht und die Rechenzeit des neuen Prototypen trotz der verschiedenen Analysetechniken dennoch geringer war.

Allerdings zeigte die Analyse des Philosophenproblemes, dass noch einige Probleme offen geblieben sind, die wir im folgenden Abschnitt untersuchen werden.

6.2 Offene Probleme

Mit den in Kapitel 4 beschriebenen Algorithmen und Datenstrukturen wurde das zentrale Problem des gewählten Aktionsbegriffes nicht gelöst – die Komplexität der disjunktiven Normalform. Zwar konnte mit dem DNF-Graphen der für eine effiziente Klauselberechnung notwendige Speicherplatz verringert werden, allerdings zeigen insbesondere die Ergebnisse des Philosophenproblemes, dass das on-the-fly-Verfahren nicht in der Lage ist, der exponentiell wachsenden Klauselanzahl entgegenzuwirken. Weiterhin steigt auch die Größe des $\sim$-Graphen exponentiell in der Anzahl der Systemvariablen, sodass selbst einfache Spezifikationen mit vielen Variablen nicht analysiert werden können.

Bei den untersuchten Fallstudien handelte es sich um sehr einfache theoretische Beispiele, die nicht mit der Komplexität realer Systeme verglichen werden können. Während bei den meisten Modellierungsmethoden in der Regel die große Anzahl der Zustände des Transitionssystemes die computergestützte Verifikation unmöglich machen, scheitert der beschriebene Ansatz bereits bei der Vorbereitung auf die Zustandsberechnung.

Beim Modelchecker TLC [YML99] der Temporal Logic of Actions müssen Aktionen nicht aus der Spezifikation berechnet werden, sondern werden direkt aus der Spezifikation entnommen. Dabei verfolgt TLC folgenden Aktionsbegriff:

> *Next* is the next-state relation, which specifies all possible steps (pairs of successive states) in a behavior of the system. It is a disjunction of actions that describe the different system operations. An action is a mathematical formula in which unprimed variables refer to the first state of a step and primed variables refer to its second state.

Dieser Ansatz ähnelt den Guarded Commands [Dij75]: Jede Aktion besteht aus Vorbedingungen und einem Effekt. Die Vorbedingung entspricht dabei Formeln, die nur aus rigiden und flexiblen Variablen besteht, z. B. $x = 1$. Der Effekt einer Aktion wird durch Formeln, die gestrichene flexible Variablen enthalten, beschrieben, z. B. $x' = x + 1$. Diese intuitive Beziehung zwischen flexiblen Variablen und aktuellem Zustand bzw. gestrichen flexiblen Variablen und dem Nachfolgezustand wurde auch bei der Implementierung der Interleavingsemantik in [Loh05] aufgegriffen: Die Klauseln der disjunktiven Normalform umgebungsinvarianten TLDA-Spezifikationen haben denselben Aufbau, enthalten jedoch zusätzlich Informationen über die $\sim$-Variablen.

Wie bereits in Abschnitt 4.3.1 beschrieben, sind die Komponenten umgebungsinvarianter Spezifikationen meist mit schwach umgebungsinvarianten Formeln spezifiziert, die oft

die Form $(\widetilde{a} \Rightarrow \varphi)$ haben. Bei der Überführung in disjunktive Normalform haben die Komponenten nach Eliminierung der Implikationen somit die Form $(\neg\widetilde{a} \vee \varphi)$, sodass die Gesamtspezifikation nahezu konjunktive Normalform hat. Somit ist der syntaktische Aufbau umgebungsinvarianter Spezifikationen der Grund für die exponentielle Anzahl zu überprüfender Klauseln.

Um die schrittbasierte Interleavingsemantik dennoch als Grundlage für einen expliziten TLDA-Modelchecker nutzen zu können, ist die Ausarbeitung eines alternativen Spezifikationsschemas notwendig: Systeme sollten dabei nicht durch eine Konjunktion schwach umgebungsinvarianter Teilspezifikationen in der beschriebenen syntaktischen Form, sondern direkt durch eine Disjunktion ihrer Aktionen spezifiziert werden. Dabei muss jedoch gewährleistet werden, dass die resultierende Spezifikation umgebungsinvariant ist, da nur so eine formale Analyse des Systems anhand seines Transitionssystemes möglich ist.

6.3 Ausblick

Wie zuvor beschrieben, kann die schrittbasierte TLDA-Interleavingsemantik für beliebige umgebungsinvariante Spezifikationen nicht effizient implementiert werden. Einen alternativen Ansatz zum expliziten Modelchecking durch Transitionssysteme bietet das symbolische Modelchecking. Dabei könnte insbesondere das SAT-basierte Modelchecking [ABE00] ein erfolgversprechender Ansatz sein, bei dem das Verifikationsproblem in ein aussagenlogisches Erfüllbarkeitsproblem überführt wird, das anschließend mit SAT-Solvern gelöst werden kann. Obwohl das Erfüllbarkeitsproblem $\mathcal{NP}$-vollständig ist, lassen sich mit modernen SAT-Solvern in der Praxis mittlerweile sehr große Formeln mit mehreren Millionen Variablen auf Erfüllbarkeit testen. Für die Logik TLA ist die Entwicklung eines SAT-basierten Modelcheckers, BTLC, angekündigt, allerdings liegen noch keine Ergebnisse vor. Eventuell könnten Erfahrungen auf die Logik TLDA übertragen und ein symbolischer TLDA-Modelchecker entwickelt werden.

Für temporale Logiken existiert neben den beschriebenen Modelchecking-Varianten ein weiteres Verfahren der computergestützten Verifikation: Da sowohl das zu verifizierende System als auch die zu überprüfende Eigenschaft im selben Formalismus spezifiziert sind, kann die Eigenschaft direkt aus der Spezifikation des Systemes *bewiesen* werden. Diese Eigenschaft, „Implementierung ist Implikation“ [Pnu79], reduziert das Verifikationsproblem auf den Beweis, dass die Systemspezifikation Φ die zu überprüfende Eigenschaft Ψ impliziert, d. h., dass die Formel $\Phi \Rightarrow \Psi$ erfüllt ist.

Die für die Logik bekannten Beweisregeln können in einem Theorembeweiser integriert werden, der dann automatisch oder interaktiv – d. h. nach Vorgabe von Zwischenzielen – den Beweis führt. Für die Logik TLA wurde ein Theorembeweiser TLP entwickelt, mit dem erfolgreiche Fallstudien durch halbautomatische Beweise verifiziert wurden [KL93]. Seit 1998 wird die Logik TLA auch vom Theorembeweiser Isabelle [NPW02] unterstützt. Durch den modularen Aufbau von Isabelle könnten die bestehenden Beweisregeln von TLDA aus [Ale05] eingearbeitet werden.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[ABE00] Abdulla, Parosh A. ; Bjesse, Per ; Eén, Niklas: Symbolic Reachability Analysis Based on SAT-Solvers. In: *TACAS '00: Proceedings of the 6th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, Springer-Verlag, April 2000 (LNCS 1785). – ISBN 3–540–67282–6, S. 411–425

[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. Spezifikation und Verifikation von Systemen mit Temporal Logic of Actions*, Humboldt-Universität zu Berlin, Mathematisch-Naturwissenschaftliche Fakultät II, Diss., Juli 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

[CES86] Clarke, Edmund M. ; Emerson, E. A. ; Sistla, A. P.: Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 8 (1986), April, Nr. 2, S. 244–263. – ISSN 0164–0925

[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

[Dij71] Dijkstra, Edsger W.: Hierarchical Ordering of Sequential Processes. In: *Acta Informatica* 1 (1971), S. 115–138. – ISSN 0001–5903

[Dij75] Dijkstra, Edsger W.: Guarded Commands, Nondeterminacy and Formal Derivation of Programs. In: *Communications of the ACM (CACM)* 18 (1975), August, Nr. 8, S. 453–457. – ISSN 0001–0782

[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

[KL93] Kurshan, Robert P. ; Lamport, Leslie: Verification of a Multiplier: 64 Bits and Beyond. In: Courcoubetis, Costas (Hrsg.): *CAV '93: Proceedings of the 5th International Conference on Computer Aided Verification*, Springer-Verlag, April 1993 (LNCS 697). – ISBN 3–540–56922–7, S. 166–179

[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

[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

[Loh05] Lohmann, Niels: *Implementierung einer schrittbasierten Interleavingsemantik für die Temporal Logic of Distributed Actions (TLDA)*, Humboldt-Universität zu Berlin, Studienarbeit, Juni 2005

[Mas04] Massuthe, Peter: *Verfeinerungstechniken der Temporal Logic of Distributed Actions TLDA*, Humboldt-Universität zu Berlin, Diplomarbeit, März 2004

[NPW02] Nipkow, Tobias ; Paulson, Lawrence C. ; Wenzel, Markus: *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*. Springer-Verlag, 2002 (LNCS 2283)

[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

[Rei86] 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

[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

Danksagung

Die Arbeit an der Interleavingsemantik für die Temporal Logic of Distributed Actions hat mir viel Freude gemacht, mich jedoch gleichermaßen des öfteren an den Rand der Verzweifelung gebracht, da ich mehr als einmal die Komplexität des Themas unterschätzte und an mir selbst zu zweifeln begann.

Daher möchte ich mich bei allen, die mir durch Rat und Tat zur Seite gestanden haben, bedanken. Besonders danke ich dem gesamten Lehrstuhl für Theorie der Programmierung, der mich so freundlich aufgenommen und mir während der Kaffeerunden und Luhme-Workshops so viel Anregung und konstruktive Kritik gegeben hat. Ausdrücklich möchte ich mich bei meinem Betreuer Peter Massuthe bedanken, der mich auf meinem Weg mit der temporalen Logik begleitet hat und stets ein offenes Ohr für Ideen und Fragen hatte. Er hat es immer verstanden, durch rechtzeitiges Bremsen oder motivierendes Anfeuern meine Arbeit in die richtige Richtung zu leiten.

Außerdem danke ich Karsten Schmidt, der mich am Ende meines Studiums doch noch für die theoretische Informatik begeistern konnte, Adrianna Alexander für die Logik TLDA, Jan Bretschneider für die Hilfe bei LaTeX, C++, etc. und Alex Julius für viele Fragen, die mir halfen, das Thema immer wieder mit anderen Augen zu betrachten.

Ganz besonders bin ich meiner Familie für alle Unterstützung, die man sich nur wünschen kann, zu großem Dank verpflichtet. Zuletzt danke ich meinem Großvater, der in mir das Interesse für die Informatik geweckt hat.

Danke!

Erklärung

Hiermit erkläre ich, die vorliegende Arbeit „*Formale Fundierung und effizientere Algorithmen für die schrittbasierte TLDA-Interleavingsemantik*“ selbstständig und ohne fremde Hilfe verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet zu haben.

Weiterhin erkläre ich hiermit mein Einverständnis, dass die vorliegende Arbeit in der Bibliothek des Instituts für Informatik der Humboldt-Universität zu Berlin ausgestellt werden darf.

Berlin, den 7. September 2005