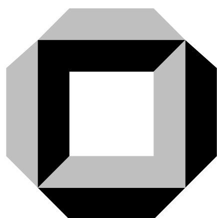

Universität Karlsruhe
Institut für Rechnerentwurf
und Fehlertoleranz
Prof. Dr.-Ing. D. Schmid

Am Zirkel 2
Postfach 6980
76128 Karlsruhe
Tel. +49 721/608-3960

Diplomarbeit

Implementierung eines Modellprüfers für die verzweigende temporale Logik CTL⋆

von

Mohamed Ghassen Ben Amor

Betreuer: Prof. Dr.-Ing. D. Schmid
Betreuender Mitarbeiter: Dr.rer.nat. Klaus Schneider

Tag der Anmeldung : 1.Dezember 1995
Tag der Abgabe: 31.Mai 1996

Erklärung

Hiermit versichere ich, daß ich die vorliegende Diplomarbeit selbständig verfaßt und keine anderen Quellen und Hilfsmittel als die im Literaturverzeichnis aufgeführten verwendet habe.

Karlsruhe, den 31. Mai 1996

Inhaltsverzeichnis

1 Einleitung **9**

2 Temporale Aussagenlogik **13**
2.1 Motivation . . . 13
2.2 Temporale Strukturen . . . 15
2.3 Syntax und Semantik der Temporallogiken . . . 16
2.3.1 Syntax und Semantik von CTL⋆ . . . 17
2.3.2 Syntax von LTL . . . 19
2.3.3 Syntax von CTL . . . 19
2.4 Fixpunktgleichungen . . . 23

3 Stand der Technik **27**
3.1 Motivation . . . 27
3.2 Binäre Entscheidungsdiagramme . . . 28
3.2.1 Binäre Entscheidungsbäume . . . 28
3.2.2 Binäre Entscheidungsdiagramme . . . 29
3.2.3 Verknüpfung von BDDs . . . 31
3.2.4 Symbolische Zustandstraversierung mittels BDDs . . . 32
3.3 CTL Modellprüfung . . . 34
3.4 LTL-Modellprüfung . . . 39
3.4.1 Tableau-Verfahren . . . 39
3.4.2 Verfahren von Schneider . . . 42
3.5 CTL⋆-Modellprüfung . . . 43

4 Übersetzung von CTL⋆ in CTL mit Fairneß **45**
4.1 Übersetzung von CTL⋆ in CTL+ . . . 45
4.2 Übersetzung von CTL+ in CTL mit Fairneß . . . 51
4.3 Bildung von Fairneßrestriktionen . . . 52

5 Experimentelle Ergebnisse **55**
5.1 Der CTL-Modellprüfer SMV . . . 55
5.2 Verifikationsbeispiele . . . 56
5.2.1 Identifikationsbaustein . . . 56
5.3 Verifikation eines Einzelimpulsbausteins . . . 58

Abbildungsverzeichnis

2.1 Kripke-Struktur und unendlicher Berechnungsbaum 16
2.2 Die Teilmengen der temporalen Aussagenlogik CTL$\star$ 19
2.3 Wirkungsweise der LTL-Operatoren 20
2.4 Eine lineare Struktur 20
2.5 Wirkungsweise der CTL-Operatoren 21
2.6 Berechnung von $\mathsf{E}[\varphi\ \mathsf{U}\ \psi]$ 24

3.1 Entwicklung der Funktion f nach der Variablen x_i 29
3.2 Binärer Entscheidungsbaum 30
3.3 Reduzierter Baum von Abbildung 3.2 31
3.4 Einfluß der Variablenordnung auf die ROBDD-Größe 32
3.5 Verknüpfung von zwei ROBDD ($i < k$) 33
3.6 Beispiel für eine Kripke-Struktur 33
3.7 Darstellung von Zustandsmengen 34
3.8 Darstellung der Zustandsübergangrelation 35
3.9 Syntaxbaum einer CTL-Formel und Ergebnisse der Modellprüfung 36
3.10 Eine Kripke-Struktur 36
3.11 Tableau für $g = a\ \mathsf{U}\ h$ 40
3.12 Eine Kripke-Struktur $\mathcal{M}$ 41
3.13 Produkt der Kripke-Struktur $\mathcal{M}$ mit dem Tableau $\mathcal{T}$ 41

5.1 REPEAT3 57

Kapitel 1

Einleitung

Heutige VLSI Schaltungen haben einen sehr hohen Komplexitätsgrad erreicht. Durch neue Anwendungsbereiche, insbesondere in sicherheitskritischen Systemen, stehen VLSI Schaltungen unter der Forderung, daß sie fehlerfrei sein müssen. Die steigende Entwurfskomplexität führt jedoch zunehmend dazu, daß schon Korrektheitsaussagen über eine Komponente eines großen und komplexen Systems immer schwieriger zu treffen sind.
Hinsichtlich der auftretenden Fehler unterscheidet man bei VLSI Schaltungen zwischen *Entwurfs-* und *Fertigungsfehlern*. Fertigungsfehler entstehen bei der Produktion von integrierten Schaltungen, indem physikalische Defekte zum Fehlverhalten einer (korrekt) entworfenen Schaltung führen. Das Problem der Fertigungsfehler ist heutzutage reichlich erforscht und man findet eine Vielzahl von *Testverfahren* [Wund91] zur Aufdeckung von Fertigungsfehlern, die auch in der Praxis erfolgreich eingesetzt werden.
Entwurfsfehler entstehen, wenn während der manuellen oder automatischen Implementierung einer Spezifikation die beabsichtigte Funktion verletzt wird. In dieser Arbeit werden nur Entwurfsfehler betrachtet.

Einen möglichen Ansatz zur Sicherstellung der Entwurfsfehlerfreiheit stellt die *Hardwareverifikation* dar [CaPr88, Yoel90, Gupt92]. Korrektheit wird hier in ihrer exakten mathematischen Bedeutung verstanden, indem durch einen formalen Beweis gezeigt wird, daß der Entwurf frei von Fehlern ist. Die Betrachtung aller Fälle ist implizit in der Methodik der formalen Verifikation enthalten [CaPr88], da Korrektheitsaussagen ohne Rücksicht auf Eingangswerte gelten. Die Hardwareverifikation ermöglicht es somit, die komplette Korrektheit zu überprüfen. Im gegensatz zur Simulation kann sie außerdem bereits während des Entwurfsprozesses eingesetzt werden [Gupt92], und erlaubt auch die Verifikation nachträglicher Optimierungen.

Es gibt vor allem zwei Gruppen von Ansätzen zur formalen Hardwareverifikation [Keut91, Gupt92, KuSK93a]: *automatische* und *interaktive* Ansätze.
Verwendet man Prädikatenlogik erster oder höherer Ordnung, so können aufgrund der Sprachmächtigkeit komplexe Sachverhalte auf höheren Abstraktionsebenen beschrieben werden. Da jedoch für Prädikatenlogik keine Entscheidungsverfahren existieren [Chur36],

ist eine Verifikation hier auf eine starke Interaktion mit dem Entwerfer angewiesen. Legt man hingegen z.B. Aussagenlogik oder temporale Aussagenlogik zugrunde, so lassen sich effizient implementierbare vollautomatische Verfahren angeben, welche auch derzeit in der Praxis eingesetzt werden. In dieser Arbeit werden temporale Aussagenlogiken verwendet.

Basierend auf einem Modell mit verzweigender Zeit wurde in [ClES83] ein Ansatz entwickelt, der sowohl temporale Aussagenlogik als auch die Automatentheorie zugrundelegt und auf dem Prinzip der *Modellprüfung* (engl. *model checking*) [BrCD85] beruht. Hierbei besteht die Modellprüfung aus dem Beweis, daß eine Spezifikation, gegeben als temporallogische Formel, bezüglich der Implementierung, welche als endlicher Automat gegeben ist, wahr ist.

Eine temporale Aussagenlogik ist CTL⋆ (*Computation Tree Logic*). Sie enthält die verzweigende Aussagenlogik CTL und die lineare Aussagenlogik LTL als echte Teilmengen.
In CTL kann man jeder Teilformel eine Menge von Zuständen zuordnen, in denen die jeweilige Formel gilt. Wenn man Zustände als Vektoren von Variablen $\vec{v}$ kodiert, kann man diese Zustandsmengen wiederum durch eine aussagenlogische Formel repräsentieren, wobei ein Zustand $\vec{v}$ in dieser Menge enthalten ist, wenn $\Phi(\vec{v})$ den Wert $true$ liefert. Dabei stellt $\vec{v}$ einen Vektor von Variablenbelegungen in einem bestimmten Zustand dar. In der Praxis speichert man diese Formeln sehr kompakt als BDD [Aker78].
Die somit mögliche symbolische Zustandstraversierung ermöglicht für CTL sehr effiziente Modellprüfungsverfahren: der Aufwand der CTL-Modellprüfung ist linear zur Modellgröße $|cM|$ und zur Formellänge d.h. $O(|\mathcal{M}| \times |\varphi|)$.
Der Aufwand der LTL-Modellprüfung ist dagegen linear zur Modellgröße $|cM|$ und exponentiell zur Formellänge d.h. $O(|\mathcal{M}| \times exp(|\varphi|))$. In LTL kann man die Gültigkeit einer Teilformel nicht mehr durch Zustandsmengen beschreiben. Daher ist keine direkte symbolische Zustandstraversierung möglich. Es gibt jedoch bereits Algorithmen zur LTL-Modellprüfung [ClGH94, Schn96b], die ein LTL-Modellprüfungsproblem in ein CTL-Modellprüfungsproblem übersetzten, um dadurch symbolische Traversierungsverfahren nutzbar zu machen.

LTL-Spezifikationen können kürzer sein als CTL-Spezifikationen und erlauben zusätzlich hierarchisches Vorgehen, indem Module separat beschrieben und entsprechend intantiiert werden. Dagegen erzwingen die CTL-Spezifikationen oft Einblick in die Implementierung und verhindern dadurch die Modularisierung. Um die Vorteile der beiden Logiken CTL und LTL zu vereinigen, betrachtet man CTL⋆. Auch hier ist die symbolische Traversierung direkt nicht anwendbar und es gibt auch bis jetzt keine Werkzeuge für CTL⋆-Modellprüfung.
Emerson & Lei [EmLe85] entwickelten einen Algorithmus zur CTL⋆-Modellprüfung, der einen LTL- und eine CTL-Modellprüfer benötigt. Als Alternative wird in dieser Arbeit ein neues Verfahren entwickelt, bei dem ein CTL⋆-Modellprüfungsproblem auf ein CTL-Modellprüfungsproblem reduziert wird.

Diese Arbeit ist wie folgt gegliedert: Kapitel 2 und 3 erarbeiten den Stand der Technik. In Kapitel 2 werden die Temporallogiken CTL$\star$, LTL und CTL vorgestellt. Kapitel 3 setzt sich mit dem Problem der Modellprüfung dieser Logiken auseinander. Dabei wird die symbolische Darstellung von Zustandsmengen mittels BDDs erläutert und anschließend werden verschiedene Verfahren zur CTL- und LTL-Modellprüfung vorgestellt. In Kapitel 4 wird die in dieser Arbeit neu entwickelte Übersetzung von CTL$\star$-Formeln in CTL-Modellprüfungsproblem mit Fairneß detailliert beschrieben. Abschließend werden in Kapitel 5 exemplarisch zwei kleine Schaltungen verifiziert. Dabei werden für jede Schaltung zwei Spezifikationen angegeben: eine in CTL$\star$ und die andere in CTL. Dabei wird deutlich, daß CTL$\star$-Spezifikationen kürzer gegenüber den CTL-Spezifikationen sein kann.

Kapitel 2

Temporale Aussagenlogik

2.1 Motivation

Für die Korrektheit gibt es zwei konventionelle Definitionen [CaPr88,StBE88]:

- Korrektheit wird in Bezug auf eine Eigenschaft einer Hardware definiert. Demnach ist eine Hardware korrekt, wenn bewiesen werden kann, daß eine Schaltung diese Eigenschaft erfüllt. Es gibt zwei wichtige Klassen [CaPr88]: Sicherheit, welche sicherstellt, daß ein System sich niemals in einem undefinierten Zustand befinden darf, und Lebendigkeit, die besagt, daß es für jeden Zustand, in dem sich das System befindet, einen Nachfolgezustand gibt, der auch erreicht werden kann.
- Korrektheit wird im Zusammenhang mit zwei Beschreibungen der Schaltung auf unterschiedlichen Abstraktionsebenen definiert, wobei eine als Spezifikation der anderen verwendet wird.

Für die Verifikation braucht man eine geeignete Spezifikationssprache, eine Modellierung der Implementierung und ein effizientes Beweisverfahren.
Das Verhalten eines Automaten kann man als Funktion von Eingabesequenzen auf Ausgabesequenzen auffassen. Der Begriff der Automatenäquivalenz entspricht somit einer speziellen Aussage über Sequenzen: Zwei Automaten sind dann gleich, wenn sie auf beliebige Eingabesequenzen mit gleichen Ausgabesequenzen reagieren.
Bei der Spezifikation digitaler Schaltungen kann man entweder nur das zeitliche Verhalten oder nur das funktionale Verhalten der Schaltung betrachten. Aber meistens werden diese zwei Verfahren zusammen eingesetzt. In dieser Arbeit wird nur das zeitliche Verhalten betrachtet. Das funktionale Verhalten kann in aussagenlogische Formeln umkodiert werden. Um das zeitliche Verhalten zu spezifizieren, benötigt man neben der Aussagenlogik noch Zeitoperatoren: Möchte man Aussagen über Sequenzen angeben, so muß man in der Lage sein, auch über verschiedene Elemente der Sequenz, d.h. über verschiedene Zeitpunkte zu argumentieren. Dies führt zur temporalen Aussagenlogik, die je nach Art der zur Verfügung stehenden Operatoren, verschiedene Ausdrucksstärke besitzt und daher auch einen unterschiedlichen Beweisaufwand erfordert. Der Einsatz temporaler Logik zur Hardware-Spezifikation und Verifikation wurde zuerst von Pnueli [Pnue77] eingeführt. Bei

dieser Logik handelte es sich um eine Erweiterung der klassischen Aussagenlogik um die temporalen Operatoren Next, Always, Sometimes und Until. Abhängig von der Ordnung der Zeitpunkte oder Intervalle der Zukunft unterscheidet man zwischen Ansätzen mit linearer [BaVe86, FuFu89] oder verzweigender [ClES83, ClLM89b] temporaler Logik. Die lineare Zeitdarstellung ermöglicht bessere Ausdrucksmöglichkeiten, während verzweigende temporale Aussagenlogik bessere Entscheidungsmöglichkeiten bietet. In diesem Abschnitt wird die Sprache CTL⋆ vorgestellt, welche die zwei zumeist eingesetzten Logiken CTL und LTL als Teilmengen beinhaltet. Im Unterschied zu klassischer Aussagenlogik sind hier Belegungen der Variablen Folgen von Werten.
Die zu verifizierende Implementierung kann auf zwei Arten modelliert werden:

- Man kann sie als Verschaltung von Basiselementen auffassen (Bibliothekselemente wie Gatter, Flipflops,...). In diesem Fall können die Grundelemente einzeln beschrieben werden. Diese Formeln können dann entsprechend ihrer Verschaltung verknüpft werden, was zu einer Beschreibung der Implementierung durch eine temporallogische Formel führt.

- Eine andere Möglichkeit ist die Betrachtung der Implementierung insgesamt als "Black-Box". Ohne Details ihres Aufbaus zu betrachten, interessieren hier nur alle möglichen Sequenzen, die sie erzeugen kann. Hier wird also die Implementierung nicht durch eine Formel, sondern als Menge möglicher Sequenzen beschrieben. Die Menge dieser Sequenzen muß natürlich möglichst effizient dargestellt werden, und es muß auch eine strukturellen Implementierungsbeschreibung der Menge dieser Sequenzen geeignet extrahiert werden.

Je nach Art der Implementierungsbeschreibung gibt es zwei verschiedene Beweisaufgaben:

- Liegen Spezifikation und Implementierung als temporallogische Formel vor, so muß die Gültigkeit einer Formel gezeigt werden. Man benötigt hier einen Tautologietest für temporale Aussagelogik, bei dem überprüft wird, ob die formulierte Korrektheitsaussage für alle möglichen Sequenzen gilt.

- Wenn die Implementierung als Menge von Sequenzen vorliegt, muß gezeigt werden, daß alle möglichen Sequenzen die gegebene temporallogische Spezifikation erfüllen. Da eine Sequenz den Wahrheitswert einer temporallogischen Formel bestimmt, muß gezeigt werden, daß alle Sequenzen der Implementierung ein Modell der Spezifikation sind. Dieses Beweisproblem heißt daher Modellprüfung (*engl. Model checking*).

Es gibt viele Varianten von temporalen Logiken. Einen guten Überblick hierfür gibt beispielsweise [Emer90, AlHe91, WiPn89].
In dieser Arbeit werden nur die temporalen Logiken CTL, LTL und CTL⋆ betrachtet. Zuerst werden Syntax und Semantik der Temporallogiken CTL⋆, CTL und LTL vorgestellt. Danach werden verschiedene Basissysteme vorgestellt. Abschließend wird eine Eigenschaft der verzweigenden temporalen Logik CTL - Fixpunktgleichungen - erläutert.

2.2 Temporale Strukturen

Zur Modellierung der möglichen Ausgabesequenzen, die ein System erzeugt, müssen die Wertebelegungen der Variablen zu den verschiedenen Zeitpunkten dargestellt werden.
Dazu gibt es verschiedene Modellierungsansätze der Zeit. Die Klassifikationskriterien für Zeit umfassen u.a. folgende Möglichkeiten: [Emer90]

- Lineare versus verzweigende Zeit: Hier gibt es zwei Sichtweisen: die erste geht von einem linearen Zeitmodell aus, d.h es gibt in jedem Zeitpunkt genau einen Folgezeitpunkt. Bei der anderen Sichtweise kann sich die Zeit in jedem Zeitpunkt aufspalten. Jeder der mehreren möglichen Folgezeitpunkte repräsentiert dann eine der möglichen zukünftigen Entwicklungen, z.B. verschiedene Berechnungsabläufe eines Systems. Die Auswahl zwischen verschiedenen Entwicklungsmöglichkeiten erfolgt in der Regel indeterministisch.
- Zeitpunkte versus Zeitintervalle: Die meisten temporalen Operatoren beziehen sich auf einzelne Zeitpunkte, einige temporale Operatoren beziehen sich jedoch auf Zeitintervalle, was die Spezifikation einiger Systeme erleichtert.
- Diskrete versus kontinuierliche Zeit: Die meisten Temporallogiken verwenden ein diskretes Zeitmodell. Meist entspricht ein diskreter Zeitpunkt dem augenblicklichen Systemzustand, der unmittelbare Folgezustand des Systems entspricht dann dem nächsten Zeitpunkt. So liegen dieser Zeitmodellierung meist die natürlichen Zahlen zugrunde. Andere Temporallogiken verwenden eine kontinuierliche (oder dichte) Zeit mit den reellen Zahlen als Modellierungsgrundlage.
- Vergangenheit versus Zukunft: Je nachdem, ob man nur die zukünftigen Zeitpunkte betrachtet, oder sich die temporalen Operatoren auch auf vergangene Zeitpunkte beziehen können, ergeben sich unterschiedliche Logiken. Wird bei der Systemmodellierung von einem definierten Startzustand ausgegangen, so genügen Temporallogiken zum Einsatz, bei denen nur Operatoren für Folgezeitpunkte vorhanden sind.

Weil in dieser Arbeit nur digitale Schaltungen betrachtet werden, werden im folgenden nur Logikvarianten verwendet, die auf einem diskreten Zeitmodell mit Zeitpunkten beruhen. Allerdings kommen sowohl Zeitmodelle mit linearer als auch verzweigender Zeit zum Einsatz. Um das jeweils verwendete Zeitmodell zu formalisieren, verwendet man temporale Strukturen. Da ein lineares Zeitmodell als Spezialfall verzweigender Zeit betrachtet werden kann, wird zuerst die Struktur für verzweigende Zeit definiert. Strukturen beruhen auf markierten Zustandsübergangsgraphen, die aus historischen Gründen auch als Kripke-Strukturen bezeichnet werden [HuGr68]. Die Semantik der Temporallogiken wird bezüglich dieser Kripke-Strukturen definiert.

Definition 2.2.1 (Kripke-Struktur)
Eine Kripke-Struktur ist gegeben durch ein Tupel $\mathcal{M} = \langle \mathcal{S}, \mathcal{R}, \mathcal{L} \rangle$, wobei

- $\mathcal{S}$ *eine endliche Menge von Zuständen,*
- $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$ *eine totale Zustandsübergangsrelation und*

- $\mathcal{L} : \mathcal{S} \longrightarrow \wp(\mathcal{V})$ *eine Markierungsfunktion, wobei* $\mathcal{V}$ *die Menge der aussagenlogischen Variablen (atomaren Formeln) ist,*

darstellt.

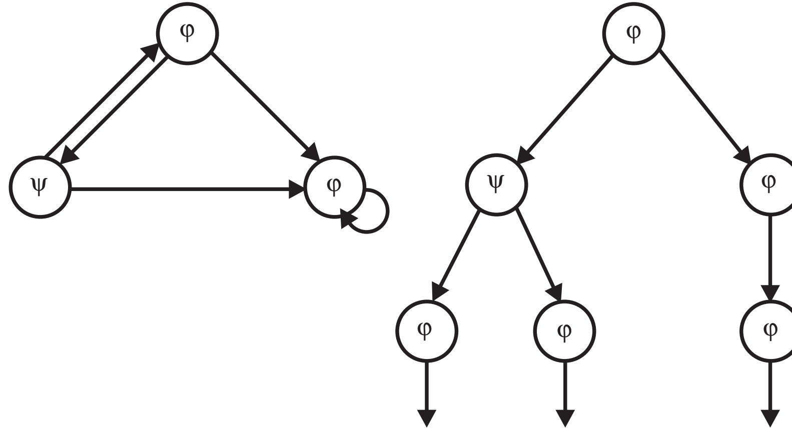

Abbildung 2.1: Kripke-Struktur und unendlicher Berechnungsbaum

Dabei wird eine Zustandsübergangsrelation als total bezeichnet, wenn für jeden Zustand $s_i \in \mathcal{S}$ ein Zustand $s_j \in \mathcal{S}$ mit $(s_i, s_j) \in \mathcal{R}$ existiert. Jeder Zustand hat also mindestens einen Folgezustand. Handelt es sich bei $\mathcal{R}$ um eine Funktion, gibt es also für jeden Zustand genau einen Folgezustand, so spricht man von einer linearen Struktur.
Eine Struktur gibt also einen Graphen an, bei dem die Knoten durch $\mathcal{S}$ und die Kanten durch $\mathcal{R}$ gegeben sind. Die Knoten sind dabei durch $\mathcal{L}$ mit Variablen markiert.
Ausgehend von einem Startzustand s_0 "*wickelt*" man die temporale Struktur entsprechend den durch $\mathcal{R}$ gegebenen Folgezuständen ab und erhält so einen unendlichen Baum. Da die Pfade des Baumes mit Wurzel s_0 im allgemeinen den möglichen Berechnungen eines Systems entsprechen, spricht man von einem Berechnungsbaum (Abbildung 2.1).

Sei $\mathcal{M} = \langle \mathcal{S}, \mathcal{R}, \mathcal{L} \rangle$ eine Kripke Struktur. Ein Pfad $\vec{\pi}$ in $\mathcal{M}$ ist eine unendliche Folge von Zuständen $s_0, s_1, s_2, s_3,...$ aus $\mathcal{S}$ derart, daß für alle i $\in \mathbb{N}$ gilt: $(s_i, s_{i+1}) \in \mathcal{R}$. Der Suffix $\vec{\pi}_i$ eines Pfades $\vec{\pi} := s_0, s_1, ..., s_i, s_{i+1}, ...$ sei definiert als $\vec{\pi}_i := s_i, s_{i+1},...$. $\vec{\pi}^i$ entspricht dem Zustand s_i.

2.3 Syntax und Semantik der Temporallogiken

In diesem Kapitel werden die am meisten in praktischen Anwendungen verwendeten Logiken vorgestellt, die verzweigende temporale Logik CTL und die lineare temporale Logik LTL. CTL beinhaltet Operatoren, die Aussagen nur über verzweigenden Zeitstrukturen machen können, während LTL auf rein linearen Zeitabläufen beruht. Um diese bei-

den Logiken besser vergleichen zu können, wird jedoch eine mächtigere Sprache vorgestellt, die sowohl CTL als auch LTL umfaßt.

2.3.1 Syntax und Semantik von CTL⋆

Die verzweigende temporale Logik CTL⋆ hat neben den üblichen aussagenlogischen Operatoren noch zwei Klassen von Operatoren: die temporalen Operatoren und die Pfadquantoren. Temporale Operatoren treffen Aussagen, die nur für einen einzelnen Berechnungspfad, ausgehend von dem momentanen Zustand, gelten. Dabei kann eine Formel gelten:

- für jeden Zustand des Pfades (G oder *always*)
- nur für einen möglichen Folgezeitpunkt des Pfades (F oder *sometimes*)
- man kann nur über den unmittelbaren Folgezustand Aussagen treffen (X oder *next*)
- man kann zwei Formeln miteinander verknüpfen, bei der die erste solange gelten muß, bis die zweite gilt (U oder *until*)
- oder die erste Formel muß gelten, wenn die zweite zum ersten Mal gilt (W oder *when*).

Bei der verzweigenden temporalen Logik muß durch Pfadquantoren bestimmt werden:

- ob eine Formel auf allen möglichen Pfaden (A oder "*for all paths*")
- oder nur auf einem Pfad (E oder "*there exists a path*")

gelten muß.

Die Syntax von CTL⋆ basiert auf 2 Klassen von Formeln: Zustandsformeln und Pfadformeln. Die Semantik einer Zustandsformeln hängt von einem bestimmten Zustand ab, während die Semantik einer Pfadformel von einem gegebenen Pfad abhängt.

Definition 2.3.1 (Syntax von CTL⋆)
Sei $\mathcal{V}$ die Menge der aussagenlogischen Variablen, dann definiert man die Syntax von Zustandsformeln bzw. Pfadformeln [Emer90] wie folgt:
Zustandsformeln $\mathcal{SF}_\mathcal{V}$:

- *Jede aussagenlogische Formel ist eine Zustandsformel.*
- *Sind φ und ψ Zustandsformeln, so sind auch $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi$ Zustandsformeln.*
- *Ist φ eine Pfadformel, so sind $\mathsf{E}\varphi$, $\mathsf{A}\varphi$ Zustandsformeln.*

Pfadformeln $\mathcal{P}_{\mathcal{SF}_\mathcal{V}}$:

- *Jede Zustandsformel ist eine Pfadformel.*
- *Sind φ und ψ Pfadformeln, so sind auch $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi$ Pfadformeln.*
- *Sind φ und ψ Pfadformeln, so sind auch $\mathsf{X}\varphi$, $\varphi \;\mathsf{U}\; \psi$,$\varphi \;\mathsf{W}\; \psi$, $\mathsf{F}\varphi$, $\mathsf{G}\varphi$ Pfadformeln.*

CTL⋆ ist die Menge der Zustandsformeln $\mathcal{SF}_\mathcal{V}$

Sei $\mathcal{M} = \langle \mathcal{S}, \mathcal{R}, \mathcal{L} \rangle$ eine Kripke-Struktur, φ eine CTL$\star$-Formel, sowie s_0 ein Zustand aus $\mathcal{S}$. Die semantische Folgerbarkeit '$\models$' ist eine Relation mit der folgenden Bedeutung:

- Ist φ eine Zustandsformel, so bedeutet die Notation $\mathcal{M}, s_0 \models \varphi$, daß die Formel φ im Zustand s_0 der Kripke-Struktur $\mathcal{M}$ erfüllt ist.
- Ist φ eine Pfadformel, dann bedeutet die Notation $\mathcal{M}, \vec{\pi} \models \varphi$, daß die Formel φ entlang des Pfades $\vec{\pi}$ der Kripke-Struktur $\mathcal{M}$ erfüllt ist.

Basierend auf einer temporalen Struktur gemäß Definition 2.2.1 kann nun die Semantik von CTL$\star$ angegeben werden.

Definition 2.3.2 (Semantik von CTL$\star$)
Die Semantik von CTL$\star$ ist rekursiv definiert.
Sei v eine aussagenlogische Formel, φ, φ_1 und φ_2 Zustandsformeln, ψ, ψ_1 und ψ_2 Pfadformeln und $\mathcal{M} = \langle \mathcal{S}, \mathcal{R}, \mathcal{L} \rangle$ eine Kripke-Struktur, dann gilt:

- ($\mathcal{S}$1) $\mathcal{M}, s_0 \models v$ *gdw.* $v \in \mathcal{L}(s_0)$.
- ($\mathcal{S}$2) $\mathcal{M}, s_0 \models (\varphi_1 \wedge \varphi_2)$ *gdw.* ($\mathcal{M}, s_0 \models \varphi_1$ *und* $\mathcal{M}, s_0 \models \varphi_2$)
  $\mathcal{M}, s_0 \models (\varphi_1 \vee \varphi_2)$ *gdw.* ($\mathcal{M}, s_0 \models \varphi_1$ *oder* $\mathcal{M}, s_0 \models \varphi_2$)
  $\mathcal{M}, s_0 \models \neg\varphi$ *gdw.* $\mathcal{M}, s_0 \not\models \varphi$
- ($\mathcal{S}$3) $\mathcal{M}, s_0 \models \mathsf{E}\psi$ *gdw. ein Pfad $\vec{\pi}$ in $\mathcal{M}$ existiert, der mit s_0 anfängt, und $\mathcal{M}, \vec{\pi} \models \psi$ gilt.*
  $\mathcal{M}, s_0 \models \mathsf{A}\psi$ *gdw. für alle Pfade $\vec{\pi}$ in $\mathcal{M}$, die mit s_0 anfangen, $\mathcal{M}, \vec{\pi} \models \psi$ gilt.*
- ($\mathcal{P}$1) $\mathcal{M}, \vec{\pi} \models \varphi$ *gdw. $\mathcal{M}, s_0 \models \varphi$, wobei s_0 der erste Zustand von $\vec{\pi}$ ist.*
- ($\mathcal{P}$2) $\mathcal{M}, \vec{\pi} \models (\psi_1 \wedge \psi_2)$ *gdw.* ($\mathcal{M}, \vec{\pi} \models \psi_1$ *und* $\mathcal{M}, \vec{\pi} \models \psi_2$)
  $\mathcal{M}, \vec{\pi} \models (\psi_1 \vee \psi_2)$ *gdw.* ($\mathcal{M}, \vec{\pi} \models \psi_1$ *oder* $\mathcal{M}, \vec{\pi} \models \psi_2$)
  $\mathcal{M}, \vec{\pi} \models \neg\psi$ *gdw.* $\mathcal{M}, \vec{\pi} \not\models \psi$
- ($\mathcal{P}$3) $\mathcal{M}, \vec{\pi} \models (\psi_1 \mathsf{U} \psi_2)$ *gdw. ein Index i existiert, so daß $\mathcal{M}, \vec{\pi}_i \models \psi_2$ und für alle j, $0 \leq j < i$, $\mathcal{M}, \vec{\pi}_j \models (\psi_1 \wedge \neg\psi_2)$ gilt.*
  $\mathcal{M}, \vec{\pi} \models (\psi_1 \mathsf{W} \psi_2)$ *gdw. ein Index i existiert, so daß $\mathcal{M}, \vec{\pi}_i \models (\psi_1 \wedge \psi_2)$ und für alle j, $0 \leq j < i$, $\mathcal{M}, \vec{\pi}_j \not\models \psi_2$ gilt.*
  $\mathcal{M}, \vec{\pi} \models \mathsf{G}\psi$ *gdw. für alle j, $\mathcal{M}, \vec{\pi}_j \models \psi$ gilt*
  $\mathcal{M}, \vec{\pi} \models \mathsf{F}\psi_1$ *gdw. ein Index i existiert, so daß $\mathcal{M}, \vec{\pi}_i \models \psi$ gilt*
  $\mathcal{M}, \vec{\pi} \models \mathsf{X}\psi$ *gdw.* $\mathcal{M}, \vec{\pi}_1 \models \psi$

Definition 2.3.3 (Gültigkeit und Erfüllbarkeit in CTL$\star$)
Eine Zustandsformel φ (bzw. Pfadformel ψ) heißt gültig, wenn für alle Strukturen $\mathcal{M}$ und alle Zustände s (bzw. für alle Pfade $\vec{\pi}$) von $\mathcal{M}$ gilt $\mathcal{M}, s_0 \models \varphi$ (bzw. $\mathcal{M}, \vec{\pi} \models \psi$).
Eine Zustandsformel φ (bzw. Pfadformel ψ) heißt erfüllbar, wenn es eine Struktur $\mathcal{M}$ und einen Zustand s (bzw. einen Pfad $\vec{\pi}$) von $\mathcal{M}$ gibt, so daß $\mathcal{M}, s_0 \models \varphi$ (bzw. $\mathcal{M}, \vec{\pi} \models \psi$) gilt.

Die Logiken LTL und CTL lassen sich als echte Teilmengen von CTL$\star$ angeben, in dem die Syntax-Regeln von Definition 2.3.1 beschränkt werden (Abbildung 2.2).

Abbildung 2.2: Die Teilmengen der temporalen Aussagenlogik CTL$\star$

2.3.2 Syntax von LTL

In der linearen temporalen Aussagenlogik LTL sind die Strukturen linear und total geordnet, d.h. es gibt für jeden Zustand genau einen Folgezustand. Sei also eine lineare Kripke-Struktur $\mathcal{M} = \langle \mathcal{S}, \vec{\pi}, \mathcal{L} \rangle$ gegeben, wobei $\vec{\pi}$ ein Pfad ist. Die Abbildung 2.3 zeigt die Wirkungsweise der temporalen Operatoren $\mathsf{F}, \mathsf{G}, \mathsf{X}, \mathsf{W}$ und U.

Definition 2.3.4 (Syntax von LTL)
Sei $\mathcal{V}$ die Menge der aussagenlogischen Variablen. Die LTL-Formeln sind Pfadformeln φ, wobei φ mit den folgenden Regeln generiert ist.

- *Jede aussagenlogische Variable ist eine Pfadformel.*
- *Sind φ und ψ Pfadformeln, so sind auch $\varphi \wedge \psi$, $\varphi \vee \psi$ und $\neg\varphi$ Pfadformel.*
- *Sind φ und ψ Pfadformeln, so sind auch $\mathsf{F}\varphi$, $\mathsf{G}\varphi$, $\mathsf{X}\varphi$, $\varphi\mathsf{W}\psi$ und $\varphi\mathsf{U}\psi$ Pfadformeln.*

Es kommen also außer aussagenlogischen Formeln keine Zustandsformeln als Teilformeln vor.
Eine LTL-Formel φ heißt erfüllbar, wenn eine lineare Struktur $\mathcal{M} = \langle \mathcal{S}, \vec{\pi}, \mathcal{L} \rangle$ existiert, so daß $\mathcal{M}, \vec{\pi} \models \varphi$ gilt. Solche Strukturen sind dann Modelle von φ. φ ist dann gültig, wenn $\neg\varphi$ nicht erfüllbar ist.
Bei der linearen temporalen Logik variieren die Variablenbelegungen der Struktur mit der Zeit, so daß sich in jedem Zeitpunkt die Wahrheitswerte der Formeln ändern können. Z.B. bei der gegebenen Belegung in der Struktur in der Abbildung 2.4 ist die Formel $\mathsf{G}\varphi$ falsch, während die Formel $\mathsf{G}(\varphi \vee \psi)$ wahr ist. Bei der linearen temporalen Aussagenlogik wird nur eine Eingabenfolge betrachtet, während bei der verzweigenden Temporallogik mehrere Eingabenfolgen betrachtet werden können.

2.3.3 Syntax von CTL

Auch die verzweigende temporale Aussagenlogik CTL ist syntaktisch und semantisch eine Untermenge von CTL$\star$. Bei CTL müssen Pfadquantoren und temporale Operatoren paarweise vorkommen.

Abbildung 2.3: Wirkungsweise der LTL-Operatoren

Definition 2.3.5 (Syntax von CTL)
Sei $\mathcal{V}$ die Menge der aussagenlogischen Variablen, dann definiert man die Syntax von CTL wie folgt:

- *($\mathcal{S}1$) Jede aussagenlogische Variable $v \in \mathcal{V}$ ist eine CTL-Formel*
- *($\mathcal{S}2$) Sind φ und ψ CTL-Formeln, so sind auch $\neg\varphi, \varphi \lor \psi, \varphi \land \psi$ CTL-Formeln*
- *($\mathcal{S}3$) Sind φ und ψ CTL-Formeln, so sind auch $\mathsf{EX}\varphi$, $\mathsf{EF}\varphi$, $\mathsf{EG}\varphi$, $\mathsf{E}[\varphi\mathsf{U}\psi]$, $\mathsf{E}[\varphi\mathsf{W}\psi]$, $\mathsf{AX}\varphi$, $\mathsf{AF}\varphi$, $\mathsf{AG}\varphi$, $\mathsf{A}[\varphi\mathsf{W}\psi]$und $\mathsf{A}[\varphi\mathsf{U}\psi]$ CTL-Formeln.*

CTL hat nur Zustandsformeln. Alle Teilformeln sind auch Zustandsformeln.

Abbildung 2.5 verdeutlicht die Wirkungsweise der temporalen Operatoren anhand einfacher Beispiele.

Abbildung 2.4: Eine lineare Struktur

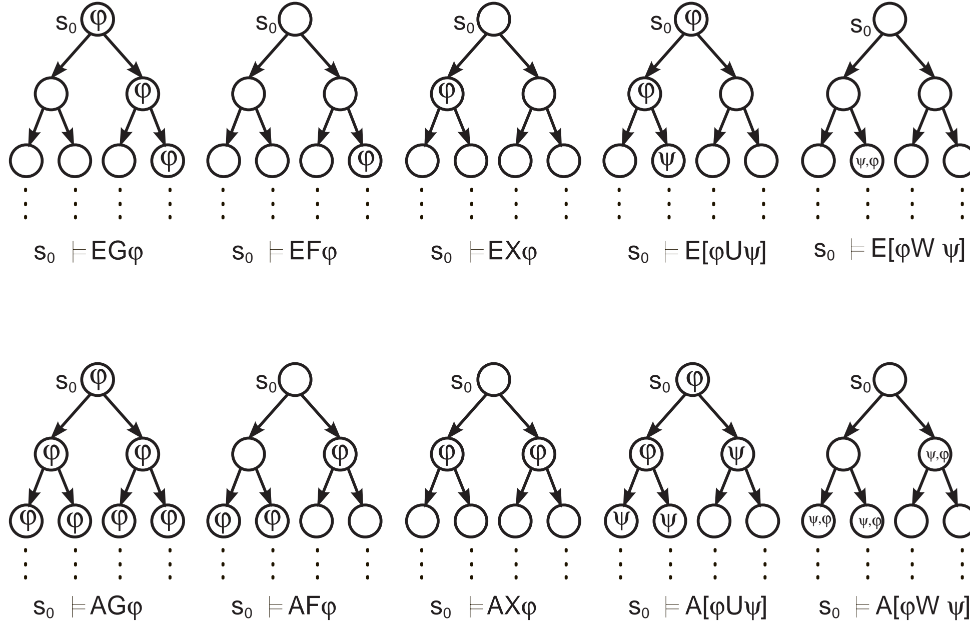

Abbildung 2.5: Wirkungsweise der CTL-Operatoren

In der Literatur findet man für die Definition der Syntax von CTL häufig andere Formulierungen, welche sich auf ein kleineres Basissystem von Operatoren beschränken und die verbleibenden Operatoren mit Umformungen auf diese Basisoperatoren zurückführen. So werden in [BCLM93, BCMD90, CaCl94] für die Definition der Syntax von CTL die temporalen Operatoren EX, EU, EG, in [AlCD90, EMSS92a] EX, EU, AU und in [Brow86, ClES86, CoMB91, McMi93a, VeLe93] EX, AX, EU, AU verwendet.

Zwischen den CTL-Operatoren bestehen Zusammenhänge, welche die Definition der Logik wesentlich vereinfachen können. Im folgenden sollen einige derartige Basissysteme von temporalen Operatoren angegeben werden.

Verwendet man für die Definition der Syntax von CTL die temporalen Operatoren EX, EU und EG, so erhält man die verbleibenden Operatoren durch:

$$\begin{array}{lcl}
\mathsf{EF}\varphi & = & \mathsf{E}[true\ \mathsf{U}\ \varphi] \\
\mathsf{E}[\varphi \mathsf{W} \psi] & = & \mathsf{E}[\neg\psi\ \mathsf{U}\ (\varphi\ \wedge\ \psi)] \\
\mathsf{AX}\varphi & = & \neg\mathsf{EX}\neg\varphi \\
\mathsf{A}[\varphi \mathsf{U} \psi] & = & \neg\mathsf{E}[\neg\psi\ \mathsf{U}\ (\neg\varphi\ \wedge\ \neg\psi)]\ \wedge\ \neg\mathsf{EG}\neg\psi \\
\mathsf{A}[\varphi \mathsf{W} \psi] & = & \neg\mathsf{E}[\neg\psi\ \mathsf{U}\ (\neg\varphi\ \wedge\ \psi)]\ \wedge\ \neg\mathsf{EG}\neg\psi \\
\mathsf{AG}\varphi & = & \neg\mathsf{E}[true\ \mathsf{U}\ \neg\varphi]
\end{array}$$

$$\mathsf{AF}\varphi = \neg\mathsf{EG}\neg\varphi$$

Bei der Verwendung des Basissystems EX, EU und AU können die restlichen Operatoren wie folgt definiert werden:

$$\begin{aligned}
\mathsf{EG}\varphi &= \neg\mathsf{A}[true\ \mathsf{U}\ \neg\varphi] \\
\mathsf{EF}\varphi &= \mathsf{E}[true\ \mathsf{U}\ \varphi] \\
\mathsf{AX}\varphi &= \neg\mathsf{EX}\neg\varphi \\
\mathsf{AG}\varphi &= \neg\mathsf{E}[true\ \mathsf{U}\ \neg\varphi] \\
\mathsf{AF}\varphi &= \neg\mathsf{EG}\neg\varphi \\
\mathsf{E}[\varphi\mathsf{W}\psi] &= \mathsf{E}[\neg\psi\ \mathsf{U}\ (\varphi\ \wedge\ \psi)] \\
\mathsf{A}[\varphi\mathsf{W}\psi] &= \mathsf{A}[\neg\psi\ \mathsf{U}\ (\varphi\ \wedge\ \psi)]
\end{aligned}$$

Erfolgt eine Definition von CTL mittels der temporalen Basisoperatoren EX, AX, EU und AU, so erhält man die verbleibenden Operatoren gemäß den Identitäten:

$$\begin{aligned}
\mathsf{EG}\varphi &= \neg\mathsf{A}[true\ \mathsf{U}\ \neg\varphi] \\
\mathsf{EF}\varphi &= \mathsf{E}[true\ \mathsf{U}\ \varphi] \\
\mathsf{AG}\varphi &= \neg\mathsf{E}[true\ \mathsf{U}\ \neg\varphi] \\
\mathsf{AF}\varphi &= \mathsf{A}[true\ \mathsf{U}\ \varphi] \\
\mathsf{E}[\varphi\mathsf{W}\psi] &= \mathsf{E}[\neg\psi\ \mathsf{U}\ (\varphi\ \wedge\ \psi)] \\
\mathsf{A}[\varphi\mathsf{W}\psi] &= \mathsf{A}[\neg\psi\ \mathsf{U}\ (\varphi\ \wedge\ \psi)]
\end{aligned}$$

Einen semantischen Unterschied gibt es bei den sog. starken und schwachen Operatoren. Bei den starken Operatoren ist für $(\varphi\ \mathsf{U}\ \psi)$ und $(\varphi\ \mathsf{W}\ \psi)$ gefordert, daß ψ irgendwann gelten muß, während bei dem schwachen UNTIL möglich ist, daß ψ niemals gilt. Bei den vorherigen Definitionen wurden nur die starken Operatoren verwendet. Auch hier kann die eine Variante durch die andere dargestellt werden. Um die beiden Varianten voneinander zu unterscheiden, bezeichnet UNTIL bzw. WHEN die schwache und U bzw. W die starke Version:

Ersetzung von starken durch schwache Operatoren:

$$\begin{aligned}
\mathsf{E}[\varphi\ \mathsf{U}\ \psi] &= \neg\mathsf{A}[\neg\psi\ \mathsf{UNTIL}\ (\neg\varphi\ \wedge\ \neg\psi)] \\
\mathsf{A}[\varphi\ \mathsf{U}\ \psi] &= \neg\mathsf{E}[\neg\psi\ \mathsf{UNTIL}\ (\neg\varphi\ \wedge\ \neg\psi)] \\
\mathsf{E}[\varphi\ \mathsf{W}\ \psi] &= \neg\mathsf{A}[\neg\varphi\ \mathsf{WHEN}\ \psi] \\
\mathsf{A}[\varphi\ \mathsf{W}\ \psi] &= \neg\mathsf{E}[\neg\varphi\ \mathsf{WHEN}\ \psi]
\end{aligned}$$

Ersetzung von schwachen durch starke Operatoren:

$$\begin{aligned}
\mathsf{E}[\varphi\ \mathsf{UNTIL}\ \psi] &= \neg\mathsf{A}[\neg\psi\ \ \mathsf{U}\ \ (\neg\varphi\ \wedge\ \neg\psi)] \\
\mathsf{A}[\varphi\ \mathsf{UNTIL}\ \psi] &= \neg\mathsf{E}[\neg\psi\ \ \mathsf{U}\ \ (\neg\varphi\ \wedge\ \neg\psi)] \\
\mathsf{E}[\varphi\ \mathsf{WHEN}\ \psi] &= \neg\mathsf{A}[\neg\varphi\ \ \mathsf{W}\ \ \psi] \\
\mathsf{A}[\varphi\ \mathsf{WHEN}\ \psi] &= \neg\mathsf{E}[\neg\varphi\ \ \mathsf{W}\ \ \psi]
\end{aligned}$$

Diese Äquivalenzen erhält man, indem man die linke Seite zweimal negiert und ein Negationszeichen nach innen durchzieht. Dabei wird die Negation von einer Variante des U- bzw. W-Operators durch die andere Variante dargestellt [Schn96b].

2.4 Fixpunktgleichungen

Eine im Hinblick auf eine effiziente Modellprüfung wichtige Eigenschaft von CTL liegt in der von Emerson und Clarke [ClEm81a, ClEm81b, ClGL93] bewiesenen Fixpunktcharakterisierung der temporalen Operatoren.
Gegeben sei eine Kripke-Struktur $\mathcal{M} = \langle \mathcal{S}, \mathcal{R}, \mathcal{L} \rangle$ gemäß Definition 2.2.1. Im folgenden soll jede CTL-Formel φ mit der Menge aller Zustände von $\mathcal{S}$, in denen φ erfüllt ist, identifiziert werden. Mengen von Zuständen werden durch aussagenlogische Formeln dargestellt. Dabei ist ein Zustand genau dann in dieser Menge, wenn die Formel mit den Variablenbelegungen aus diesem Zustand wahr ist. Diese aussagenlogischen Formeln werden - wie in nächsten Kapitel gezeigt wird - mittels binärer Entscheidungsdiagramme dargestellt. Hier wird die kleinste Menge - die leere Menge - mit $false$ und die größte Menge - $\mathcal{S} selbst$ - mit $true$ identifiziert.

Gegeben sei eine Abbildung $\tau : \mathcal{P}(\mathcal{S}) \longrightarrow \mathcal{P}(\mathcal{S})$, wobei $\mathcal{P}(\mathcal{S})$ die Potenzmenge von $\mathcal{S}$ ist. Dann gilt:

- $\mathcal{T} \in \mathcal{P}(\mathcal{S})$ heißt *Fixpunkt* von τ, wenn $\tau(\mathcal{T}) = \mathcal{T}$ gilt
- τ heißt *monoton*, wenn für alle Elemente $\mathcal{T}, \mathcal{Q} \in \mathcal{P}(\mathcal{S})$ gilt:
  aus $\mathcal{T} \subseteq \mathcal{Q}$ folgt $\tau(\mathcal{T}) \subseteq \tau(\mathcal{Q})$
- τ heißt $\cup - stetig$, wenn für alle Elemente $\mathcal{T}_0, \mathcal{T}_1, \mathcal{T}_2 ...$ von $\mathcal{P}(\mathcal{S})$ gilt:
  aus $\mathcal{T}_0 \subseteq \mathcal{T}_1 \subseteq ...$ folgt $\tau(\bigcup_i \mathcal{T}_i) = \bigcup_i \tau(\mathcal{T}_i)$
- τ heißt $\cap - stetig$, wenn für alle Elemente $\mathcal{T}_0, \mathcal{T}_1, \mathcal{T}_2 ...$ von $\mathcal{P}(\mathcal{S})$ gilt:
  aus $\mathcal{T}_0 \supseteq \mathcal{T}_1 \supseteq ...$ folgt $\tau(\bigcap_i \mathcal{T}_i) = \bigcap_i \tau(\mathcal{T}_i)$

Nach Tarski [Tars55] besitzt die Abbildung τ die folgenden Eigenschaften:

- Falls τ monoton ist, so ist τ auch $\cup$-stetig und $\cap$-stetig
- Falls τ monoton ist, so besitzt τ einen kleinsten Fixpunkt $klFp\ Z.[\tau(Z)]$ und einen größten Fixpunkt $grFp\ Z.[\tau(Z)]$. Dabei ist der kleinste bzw. der größte Fixpunkt von τ definiert als der Durchschnitt bzw. die Vereinigung aller Fixpunkte von τ
- Falls τ monoton ist, so gilt:
  für alle $i \in \mathbb{N}$, $\tau^i(false) \subseteq \tau^{i+1}(false)$ und $\tau^i(true) \supseteq \tau^{i+1}(true)$. Dabei bedeutet τ^i die i-malige Iteration von τ
- Falls τ monoton und $\mathcal{M}$ endlich ist, so gilt:
  es existiert $i_0, j_0 \in \mathbb{N}$, so daß für alle $j \geq i_0$, $\tau^j(false) = \tau^{i_0}(false)$ und für alle $j \geq j_0$, $\tau^j(true) = \tau^{j_0}(true)$
- Falls τ monoton und $\mathcal{M}$ endlich ist, so gilt:
  es existiert $i_0, j_0 \in \mathbb{N}$, so daß $klFp\ Z.[\tau(Z)] = \tau^{i_0}(false)$ und $grFp\ Z.[\tau(Z)] = \tau^{j_0}(true)$

Als Folgerung dieser Eigenschaften können, falls τ monoton ist, die kleinsten bzw. die größten Fixpunkte von τ mit den folgenden Funktionen berechnet werden:

function $klFp(\tau)$	function $grFp(\tau)$
$\{\mathcal{Q} = false;$	$\{\mathcal{Q} = true;$
$\mathcal{T} = \tau(\mathcal{Q});$	$\mathcal{T} = \tau(\mathcal{Q});$
while$(\mathcal{Q} \neq \mathcal{T})$	while$(\mathcal{Q} \neq \mathcal{T})$
$\{\mathcal{Q} = \mathcal{T};$	$\{\mathcal{Q} = \mathcal{T};$
$\mathcal{T} = \tau(\mathcal{T});$	$\mathcal{T} = \tau(\mathcal{T});\}$
return$(\mathcal{Q});$	return$(\mathcal{Q});$
$\}$	$\}$

Wenn jede CTL-Formel φ mit der Menge $\{s|\mathcal{M}, s_0 \models \varphi\}$ aller Zustände, in denen φ erfüllt ist, identifiziert wird, so kann jeder CTL-Operator als kleinster oder größter Fixpunkt charakterisiert werden:

- $\mathsf{E}[\varphi\ \mathsf{U}\ \psi] = klFp\ Z.(\psi \lor (\varphi \land \mathsf{EX}\ Z))$
- $\mathsf{A}[\varphi\ \mathsf{U}\ \psi] = klFp\ Z.(\psi \lor (\varphi \land \mathsf{AX}\ Z))$
- $\mathsf{EG}\ \varphi = grFp\ Z.(\varphi \land \mathsf{EX}\ Z)$
- $\mathsf{AG}\ \varphi = grFp\ Z.(\varphi \land \mathsf{AX}\ Z)$
- $\mathsf{EF}\ \varphi = klFp\ Z.(\varphi \lor \mathsf{EX}\ Z)$
- $\mathsf{AF}\ \varphi = klFp\ Z.(\varphi \lor \mathsf{AX}\ Z)$

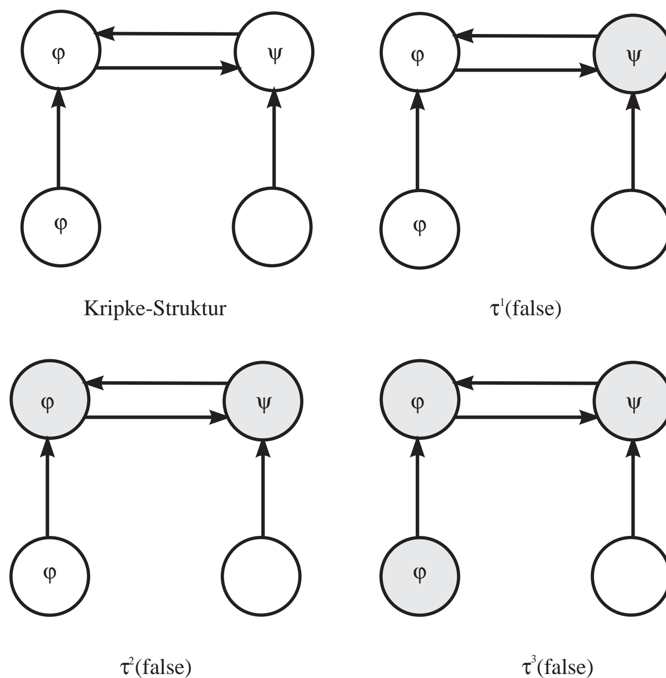

Abbildung 2.6: Berechnung von $\mathsf{E}[\varphi\ \mathsf{U}\ \psi]$

Abbildung 2.6 zeigt, wie die Funktion $klFp$ die Menge der Zustände berechnet, in denen $\mathsf{E}[\varphi \ \mathsf{U} \ \psi]$ erfüllt ist. In diesem Falle ist $\tau(Z) = \psi \ \vee \ (\varphi \wedge \mathsf{EX} \ Z)$.
Die Abbildung zeigt - bei einer gegebenen Kripke-Struktur - wie $\tau^i(false)$ nach $\mathsf{E}[\varphi \ \mathsf{U} \ \psi]$ konvergiert. Die schattierten Zustände bezeichnen die berechnete Menge nach der i-ten Iteration von τ.
Es ist hier leicht erkennbar, daß $\tau^3(false) = \tau^4(false)$ und daher $\mathsf{E}[\varphi \ \mathsf{U} \ \psi] = \tau^3(false)$. Wegen $s_0 \in \tau^3(false)$ gilt $\mathcal{M}, s_0 \models \mathsf{E}[\varphi \ \mathsf{U} \ \psi]$.

Kapitel 3

Stand der Technik

3.1 Motivation

Automatische Beweiser basieren auf Aussagenlogik, temporaler Aussagenlogik und auf Verfahren zum Vergleich von Automaten. Aussagenlogik ist vollständig und entscheidbar [BlBu87, Scho87]. Die Verfahren, die auf Aussagenlogik basieren, sind daher gut automatisierbar, jedoch sind ihre Ausdrucksmittel für die Verifikation von Schaltungen sehr begrenzt. Für Spezielle Anwendungen, wie die Verifikation von Schaltnetzen auf Logikebene, reicht aber die Aussagenlogik aus.
In den letzten Jahren wurden insbesondere bei der Entwicklung von automatischen Beweisern der Aussagelogik, große Fortschritte mit Hilfe von Entscheidungsgraphen (engl. „BDD's" für „binary decision diagrams") erzielt. diese Graphen wurden zuerst in [Aker78] zur Repräsentation von booleschen Funktionen vorgeschlagen und von [Brya86] derart ergänzt, daß daraus eine eindeutige Normalform entstand. Diese Darstellung wird erfolgreich zur Verifikation komplexer Schaltnetze eingesetzt [MWBS88, MaBi88] und ständig in ihrer Effizienz verbessert [MiIY90a]. Allgemein steigt jedoch die Komplexität der Verifikation mit Tautologieprüfern exponentiell an, so daß der Anwendung dieses Verfahrens Grenzen gesetzt sind.
Für die Verifikation von Schaltwerken reicht die klassische Aussagenlogik nicht mehr aus. Hierzu kann man auf die Automatentheorie zurückgreifen, indem sequentielle Schaltungen durch endliche Automaten [Koha70] dargestellt werden, so daß sich die Schaltungsäquivalenz einfach auf Automatenäquivalenz zurückführen läßt. Der Vergleich zweier endlicher Automaten geschieht durch Konstruktion des Produktautomaten, dessen Ausgang bei Gleichheit den Wahrheitswert "wahr" annimmt [SuFr86]. Obwohl es effiziente Verfahren zur Bildung des Produktautomaten gibt [CoMa91], ist ein expliziter Durchlauf aller erreichbaren Zustände im allgemeinen wegen des großen Speicherplatz- und Rechenzeitbedarfs nicht möglich [Evek91b].
Automaten kann man auch als Kripke-Strukturen interpretieren. Die Modellprüfung besteht aus dem Nachweis, daß eine Spezifikation, gegeben als temporallogische Formel, bezüglich des Modells der Implementierung, gegeben als endlicher Automat, wahr ist [Gupt92]. Dank der Entscheidbarkeit des zugrundeliegenden Kalküls ist eine komplette

automatische Beweisausführung möglich. Die Effizienz der Verfahren erlaubt die Verifikation mittelgroßer Schaltungen.

3.2 Binäre Entscheidungsdiagramme

Der Übergang von der Modellprüfung zur symbolischen Modellprüfung basiert auf einer bezüglich Speicherbedarf und Manipulierbarkeit effizienteren Darstellung von Zustandsmengen und Zustandsübergangsrelationen. Ein wichtiges Hilfsmittel hierfür stellen die *binäre Entscheidungsdiagramme* dar, die zuerst in [Aker78] zur Repräsentation von booleschen Funktionen vorgeschlagen wurden und von Bryant [Brya86] derart geändert, daß daraus eine Normalform entstand.

3.2.1 Binäre Entscheidungsbäume

Bei der Berechnung des Funktionswerts einer booleschen Funktion kann man eine Wertetabelle bilden, indem bei einer konkreten Variablenbelegung den Funktionswert sofort ablesen kann. Die Darstellung der gesamten Funktion erfordert jedoch einen großen Speicherbedarf ($O(2^{|\varphi|})$). Die Darstellung der booleschen Funktion durch eine aussagenlogische Formel ist hingegen sehr kompakt. Dafür ist der Test, ob eine Formel gilt oder nicht sehr aufwendig.

Theorem 3.2.1 (Kofaktoren [BHMS86])
Sei f eine boolesche Funktion mit n Variablen $x_1, ..., x_n$ und $i \in \{1, ...n\}$. Der Ausdruck $f(x_1, ..., x_i := 1, ..., x_n)$ ist der Kofaktor von f nach x_i bzw. die Entwicklung von f nach x_i (geschrieben f_{x_i}), der Ausdruck $f(x_1, ..., x_i := 0, ..., x_n)$ ist der Kofaktor von f nach $\neg x_i$ (geschrieben $f_{\neg x_i}$).

Theorem 3.2.2 (Shannonscher Entwicklungssatz [Shan48])
Sei f eine boolesche Funktion mit n Variablen $x_1, ..., x_n$ und $i \in \{1, ...n\}$. Dann gilt:
$f(x_1, ..., x_i, ..., x_n) = (x_i \ \wedge \ f_{x_i}) \ \vee \ (\neg x_i \ \wedge \ f_{\neg x_i})$.

Entwickelt man eine Funktion nach allen ihren Variablen gemäß einer vorgegebenen Variablenbelegung, so resultiert daraus ein konstanter Wert, der gerade dem Funktionswert der zugehörigen Belegung entspricht.

Theorem 3.2.3 (Homomorphie von Kofaktoren)
Gegeben seien zwei boolesche Funktionen f und g sowie ein beliebiger zweistelliger boolescher Operator $\star$. Dann gilt $(f \star g)_{x_i} = f_{x_i} \star g_{x_i}$ und $(\neg f)_{x_i} = \neg(f_{x_i})$.

Die Kofaktor-Bildung ist also distributiv bezüglich boolescher Operatoren. Der Shannonsche Entwicklungssatz läßt sich auch graphisch darstellen (Abbildung 3.1). Entwickelt man eine boolesche Funktion nach allen Variablen und verwendet die graphische Darstellung von Abbildung 3.1 so erhält man einen *binären Entscheidungsbaum* [More82].

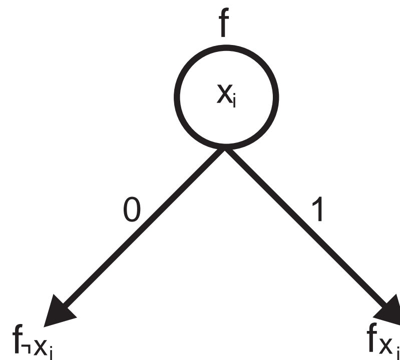

Abbildung 3.1: Entwicklung der Funktion f nach der Variablen x_i

Mittels Durchlaufen eines binären Entscheidungsbaumes kann der zu einer gegebenen Variablenbelegung gehörende Wahrheitswert der zugrunde liegenden booleschen Funktion ermittelt werden. Dazu wird beginnend beim Wurzelknoten im Falle eines inneren Knoten v entsprechend der Belegung der zugehörigen Variablen $var(v)$ nach $f_{var(v)}$ oder $f_{\neg var(v)}$ verzweigt. Ist ein Blattknoten erreicht, so entspricht der Wert dieses Blattes dem zu der Variablenbelegung gehörenden Wahrheitswert. Abbildung 3.2 zeigt den binären Entscheidungsbaum für die boolesche Funktion $f = (x_1 \wedge x_2) \vee x_3$.

3.2.2 Binäre Entscheidungsdiagramme

Binäre Entscheidungsbäume bieten gegenüber den Wertetabellen kaum nennenswerte Vorteile. Man erkennt, daß diese Darstellung redundante Teile enthält:

- identische Teilbäume (z.B. B1, B2 und B3 in Abbildung 3.2)
- Entscheidungsknoten, die nicht benötigt werden, da beide Unterbäume identisch sind (z.B. B4 in Abbildung 3.2)

Eliminiert man diese redundanten Teile so erhält man einen Baum nach Abbildung 3.3. Diese ursprünglich von Akers vorgestellte Struktur [Aker78] wurde von Bryant [Brya86] durch die Einführung von Variablenordnung zu einer Normalformdarstellung weiterentwickelt. Hierbei wurde zusätzlich gefordert, daß beim Durchlaufen des Baumes von der Wurzel zu einem Blatt die auf dem jeweiligen Pfad vorhandenen Variablen immer in der gleichen Reihenfolge auftreten. In Abbildung 3.2 und Abbildung 3.3 ist diese Ordnung durch die Folge x_1, x_2, x_3 gegeben. Diese Eigenschaft führte auch zu ihrem Namen *geordnete binäre Entscheidungsdiagramme* (*Reduced Ordered Binary Decision Diagrams*). Solche Bäume werden meist als *ROBDD* oder vereinfachend als *BDD* bezeichnet. Er ist wie folgt definiert:

Definition 3.2.1 (Geordnetes binäres Entscheidungsdiagramm)
Ein ROBDD mit n Variablen ist ein gerichteter zyklenfreier Graph $G := (V, E)$. *Ein*

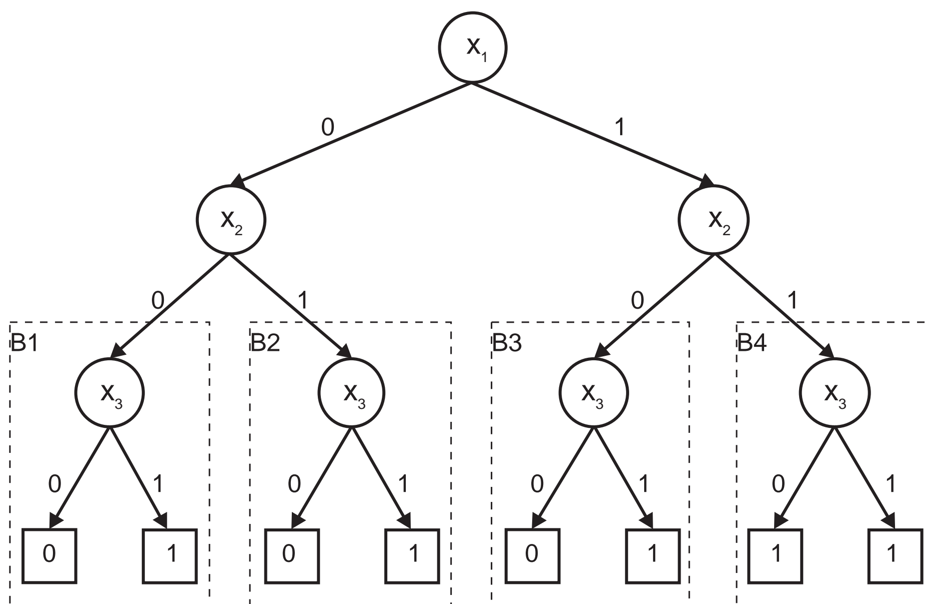

Abbildung 3.2: Binärer Entscheidungsbaum

Knoten $v \in V$ ist entweder innerer Knoten oder Blatt. Ein Blatt hat keine Söhne, jedoch einen zugeordneten Wert $val(v) \in \{0, 1\}$. Ein innerer Knoten hat zwei Söhne $links : V \to V$, $rechts : V \to V$ sowie einen Index $var(v) \in \{1, ..., n\}$. Der Graph ist geordnet, d.h. es gilt $var(v) < var(links(v))$ sowie $var(v) < var(rechts(v))$. Die Kanten $e \in E$ sind alle Paare $(v, links(v))$ und $(v, rechts(v))$. Die Größe eines ROBDDs ist die Zahl der Knoten, geschrieben als $|G|$.

Für die Reduktion eines BDDs, d.h. die Beseitigung redundanter Teile benötigt man den Begriff der Isomorphie von ROBDDs, der wie folgt definiert ist:

Definition 3.2.2 (Isomorphie)
Zwei ROBDDs $G := (V, E)$ und $G' := (V', E')$ heißen isomorph genau dann, wenn es eine bijektive Funktion $h : V \to V'$ gibt, so daß für alle Blätter gilt $val(v) = val(h(v))$ und für alle innere Knoten $h(links(v)) = links(h(v))$ sowie $h(rechts(v)) = rechts(h(v))$ gilt.

Definition 3.2.3 (Reduzierter ROBDD)
Ein ROBDD heißt reduziert genau dann, wenn kein Knoten existiert, für den $links(v) = rechts(v)$ gilt und wenn keine zwei Knoten v und v' existieren, so daß die Teilbäume mit den Wurzeln v und v' isomorph sind.

Die Normalform-Eigenschaft von ROBDD ergibt sich aus dem folgenden Satz [Brya86]:

Theorem 3.2.4
Für jede boolesche Funktion f und eine gegebene Variablenordnung gibt es einen - bis auf Isomorphie - eindeutigen, reduzierten ROBDD.

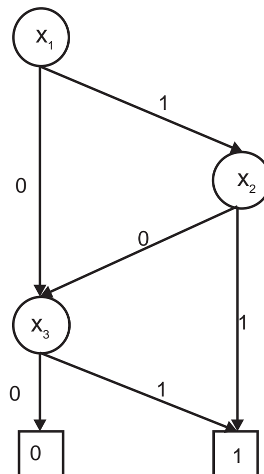

Abbildung 3.3: Reduzierter Baum von Abbildung 3.2

Die Größe eines ROBDDs wird entscheidend von seiner zugehörigen Variablenordnung beeinflußt. In Abbildung 3.4 ist der ROBDD der booleschen Funktion $f = (x_1 \wedge x_2) \vee x_3$ für die Variablenordnung $x_1 < x_3 < x_2$ gezeigt. Offensichtlich ist der ROBDD in der Abbildung 3.3 mit der Variablenordnung $x_1 < x_2 < x_3$ kleiner als der in Abbildung 3.4. Die Bestimmung der optimalen Variablenordnung für eine bestimmte Funktion ist ein NP-vollständiges Problem, läßt sich also nicht effizient berechnen [FrSu90].

3.2.3 Verknüpfung von BDDs

Möchte man den ROBDD einer Funktionsverknüpfung $f = f_1 \star f_2$ bestimmen, so gilt mit Satz 3.2.3:

$$\begin{aligned} f &= x_i f_{x_i} \ \vee \ \neg x_i f_{\neg x_i} \\ &= x_i (f_1 \star f_2)_{x_i} \ \vee \ \neg x_i (f_1 \star f_2)_{\neg x_i} \\ &= x_i (f_{1x_i} \star f_{2x_i}) \ \vee \ \neg x_i (f_{1\neg x_i} \star f_{2\neg x_i}) \end{aligned}$$

In den ROBDDs für f_1 und f_2 entsprechen die Kofaktoren nach x_i den rechten Teilbäumen sowie die Kofaktoren nach $\neg x_i$ den linken Teilbäumen des Knoten x_i. Berücksichtigt man noch, daß gilt $f_{x_i} = f$ und $f_{\neg x_i} = f$ falls x_i in f nicht vorkommt, so kann die Verknüpfung $f := f_1 \star f_2$ rekursiv wie in Abbildung 3.5 durchgeführt werden. Die Negation eines ROBDDs erfolgt durch Komplementierung der Blätter. Durch Anwendung der rekursiven Verknüpfungsprozedur von Abbildung 3.5 kann ein ROBDD entstehen, der nicht mehr reduziert ist. Aus diesem Grund muß nach der Verknüpfung wieder eine Reduktion durchgeführt werden.
Die Verknüpfung von zwei ROBDDs kann im schlimmsten Fall exponentiellen Aufwand erfordern, da nach Abbildung 3.5 jeder Verknüpfungsaufruf wiederum zwei rekursive Aufrufe

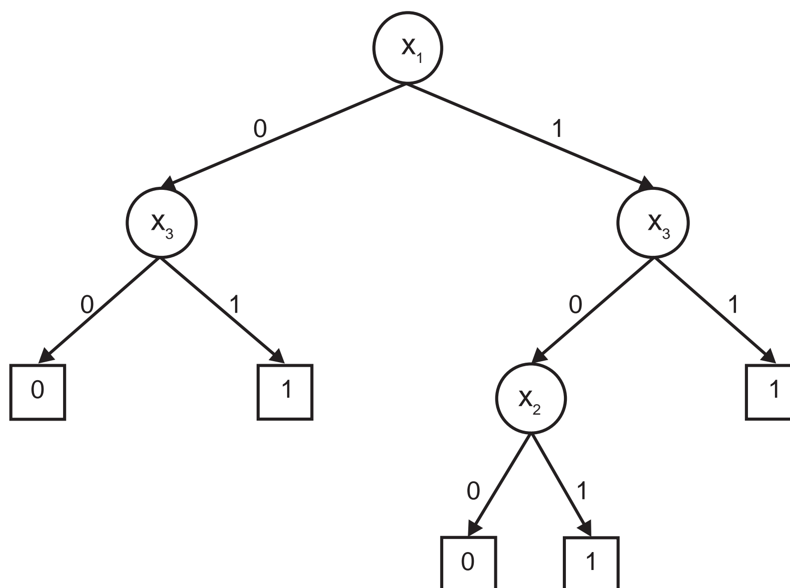

Abbildung 3.4: Einfluß der Variablenordnung auf die ROBDD-Größe

zur Folge haben kann. Der Aufwand kann jedoch durch Anwendung spezieller Techniken linear zu der Größe der ROBDDs sein [Brya86].

3.2.4 Symbolische Zustandstraversierung mittels BDDs

Gegeben sei eine Kripke-Struktur $\mathcal{M} = \langle \mathcal{S}, \mathcal{R}, \mathcal{L} \rangle$, mit einer Menge von atomaren Formeln $\mathcal{V} = \{v_1, ..., v_n\}$ und eine Zustandsmenge $\mathcal{S} = \{s_1, ..., s_m\}$. Aufgrund der Injektivität der Markierungsfunktion gilt $m \leq 2^n$. jeder Zustand $s_i \in \mathcal{S}$ entspricht in eindeutiger Weise einer Teilmenge $\mathcal{V}_i$ der Variablenmenge $\mathcal{V}$.

- jeder Zustand s_i entspricht einer Konjunktion $\bigwedge_{\sigma_i \in \mathcal{V}} \sigma_i$ mit $\begin{cases} \sigma_i = v_i \; falls \; v_i \in \mathcal{L}(s) \\ \sigma_i = \neg v_i \; falls \; v_i \notin \mathcal{L}(s) \end{cases}$
- die Vereinigung bzw. Schnitt von entsprechen Disjunktion bzw. Konjunktion
- das Komplement entspricht der Negation

Diese Sichtweise gestattet es also, Zustandsmengen als boolesche Funktionen aufzufassen. Die so erhaltenen booleschen Funktionen werden nun mittels BDD dargestellt.
Zur Verdeutlichung der Abhängigkeiten zwischen den unterschiedlichen Repräsentationen von Zustandsmengen soll in der in Abbildung 3.6 dargestellten Kripke-Struktur $\mathcal{M} = \langle \mathcal{S}, \mathcal{R}, \mathcal{L} \rangle$ die Zustandsmenge $\mathcal{S}' := \{s_0, s_2\} \subseteq \mathcal{S}$ betrachtet werden. Dabei wird jeder Zustand als Tupel (q_0, q_1) repräsentiert. $\mathcal{S}'$ entspricht der Menge $\{(0,0),(1,1)\}$ von booleschen Vektoren, der booleschen Funktion $(\neg q_0 \wedge \neg q_1) \vee (q_0 \wedge q_1)$ sowie dem in

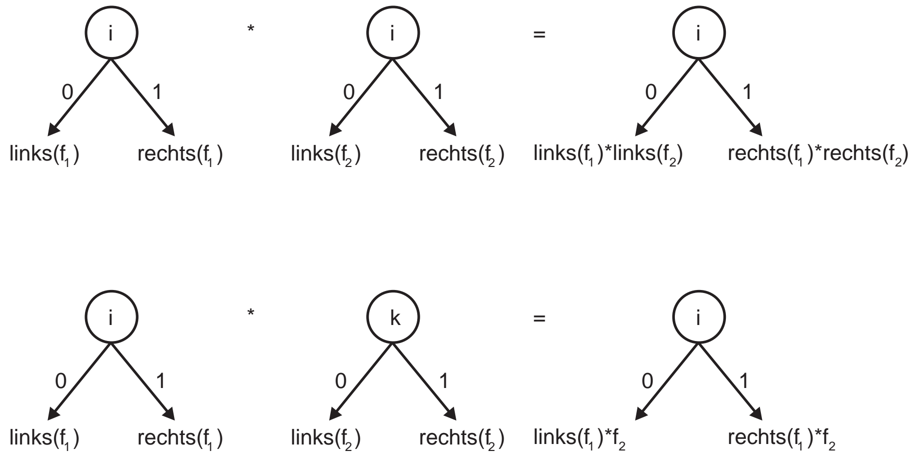

Abbildung 3.5: Verknüpfung von zwei ROBDD ($i < k$)

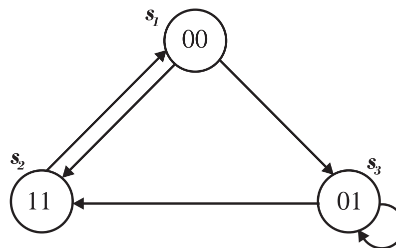

Abbildung 3.6: Beispiel für eine Kripke-Struktur

Abbildung 3.7 dargestellten BDD (bei variablenordnung $q_0 < q_1$).

Diese Betrachtung bezüglich Zustandsmengen kann auf die Darstellung der Zustandsübergangsrelationen übertragen werden.
Zur Bestimmung der Zustandsübergangsrelation $\mathcal{R}$ verwendet man neben der Variablenmenge $\mathcal{V}$ eine weitere Kopie $\mathcal{V}' = \{v_1', ..., v_n'\}$ von $\mathcal{V}$. Die erste Variablenmenge charakterisiert den aktuellen Zustand, die zweite Variablenmenge charakterisiert den Folgezustand. Somit entspricht der Zustand $s_i \in \mathcal{S}$ einer Teilmenge $\mathcal{V}_i$ von $\mathcal{V}$ und jeder Folgezustand $s_j \in \mathcal{S}$ einer Teilmenge $\mathcal{V}_i'$ von $\mathcal{V}'$. Aufgrund der Disjunktheit der Mengen $\mathcal{V}$ und $\mathcal{V}'$ ist ein Zustandsübergang $(s_i, s_j) \in \mathcal{R}$ eindeutig durch eine Teilmenge von $\mathcal{V}_i \cup \mathcal{V}_i'$ charakterisiert.

Mit der so erhaltenen Mengendarstellung kann eine Zustandsübergangsrelation als boolesche Funktion dargestellt werden. Insbesondere ist somit auch die gesamte Zustandsübergangsrelation (als Vereinigung sämtlicher Zustandsübergänge) als eine boolesche Funktion in den Variablen $v_1, ..., v_n, v_1', ..., v_n'$ darstellbar.

Abbildung 3.7: Darstellung von Zustandsmengen

Die der Zustandsübergangsrelation entsprechende boolesche Funktion wird ebenfalls mittels BDDs dargestellt.

Als Beispiel soll die Zustandsübergangsrelation der in der Abbildung 3.6 dargestellten Kripke-Struktur betrachtet werden. Abbildung 3.8 zeigt das der Zustandsübergangsrelation $\mathcal{R}$ entsprechende BDD bei Variablenordnung $q_0 < q_1 < q_0' < q_1'$. Die zugehörige boolesche Funktion lautet:

$$(q_0 \ \wedge \ \neg q_1') \ \vee \ (q_0 \ \wedge \ q_1 \ \wedge \ \neg q_0' \ \wedge \ \neg q_1')$$

3.3 CTL Modellprüfung

Um eine Verifikation durchzuführen, benötigt man - wie in Kapitel 2 erläutert - eine Implementierungsbeschreibung und eine Spezifikation der zu verifizierenden Schaltung. Eine geeignete Darstellung der Implementierung ist die Kripke-Struktur gemäß Definition 2.2.1. Dabei heißt die Kripke-Struktur $\mathcal{M} = \langle \mathcal{S}, \mathcal{R}, \mathcal{L} \rangle$ *Modell* für die als CTL-Formel gegebene Spezifikation genau dann, wenn diese Formel in $\mathcal{M}$ erfüllbar ist.

Definition 3.3.1 (Modellprüfung)
Unter dem Begriff Modellprüfung bezeichnet man die Lösung des Problems, für eine gegebene Kripke-Struktur $\mathcal{M} = \langle \mathcal{S}, \mathcal{R}, \mathcal{L} \rangle$ und eine gegebene CTL-Formel φ die Menge $Sat(\varphi) = \{s \in \mathcal{S} |\ \mathcal{M}, s \models \varphi\}$, d.i. die Menge aller Zustände $s \in \mathcal{S}$ in denen φ erfüllt ist, zu bestimmen.

Abbildung 3.8: Darstellung der Zustandsübergangrelation

Benutzt man ein Basissystem bestehend aus $\mathsf{EX}, \mathsf{EU}, \mathsf{EG}$ und den logischen Operatoren $\neg$ und $\wedge$ sowie der Konstanten $true$, so kann man $Sat(\varphi)$ wie folgt berechnen:
Seien φ und ψ CTL-Formeln und sei v eine atomare Formel, so gilt:

- $Sat(true) = \mathcal{S}$
- $Sat(v) = \{s \in \mathcal{S} \mid v \in \mathcal{L}(s)\ \}$
- $Sat(\neg\varphi) = \mathcal{S} \setminus Sat(\varphi)$
- $Sat(\varphi \ \wedge \ \psi) = Sat(\varphi) \cap Sat(\psi)$
- $Sat(\mathsf{EX}\varphi) = \{s_0 \in \mathcal{S} \mid \exists s_1 \in Sat(\varphi),\ (s_0, s_1) \in \mathcal{R}\}$
- $Sat(\mathsf{E}[\varphi\ \mathsf{U}\ \psi]) = \{s \in \mathcal{S} \mid \exists \vec{\pi} \in \mathcal{M}, n \in \mathbb{N} : \vec{\pi}^0 := s$ und $\vec{\pi}^0, ..., \vec{\pi}^{(n-1)} \in Sat(\varphi), \vec{\pi}^n \in Sat(\psi)\ \}$
- $Sat(\mathsf{EG}\varphi) = \{s_0 \in \mathcal{S} \mid \exists \vec{\pi}^0 := s, \forall n \in \mathbb{N}\ \vec{\pi}^n \in Sat(\varphi)\ \}$

Die Ermittlung der Zustandsmenge $Sat(v)$ für eine gegebene aussagenlogische Variable v erfolgt durch die Markierungsfunktion $\mathcal{L}$ der verwendeten Kripke-Struktur. Die Bestimmung der Zustandsmengen $Sat(\neg\varphi)$ bzw. $Sat(\varphi \ \wedge \ \psi)$ geschieht durch Bildung des Mengenkomplements von $Sat(\varphi)$ bzw. durch Berechnung der Schnittmenge von $Sat(\varphi)$ und $Sat(\psi)$.
Die Bestimmung der Zustandsmenge $Sat(\mathsf{EX}\varphi)$ erfordert die Betrachtung der Zustandsübergangsgraphen $\mathcal{R}$. Dabei werden alle Zustände zurückgeliefert, die einen Folgezustand haben, in welchen φ erfüllt ist. Die Berechnung dieser Mengen besteht in der Lösung von

Fixpunktgleichungen.

Als Beispiel betrachte man die CTL-Formel $\mathsf{EF}(\varphi \wedge \mathsf{EX}\psi)$. Bei der Durchführung

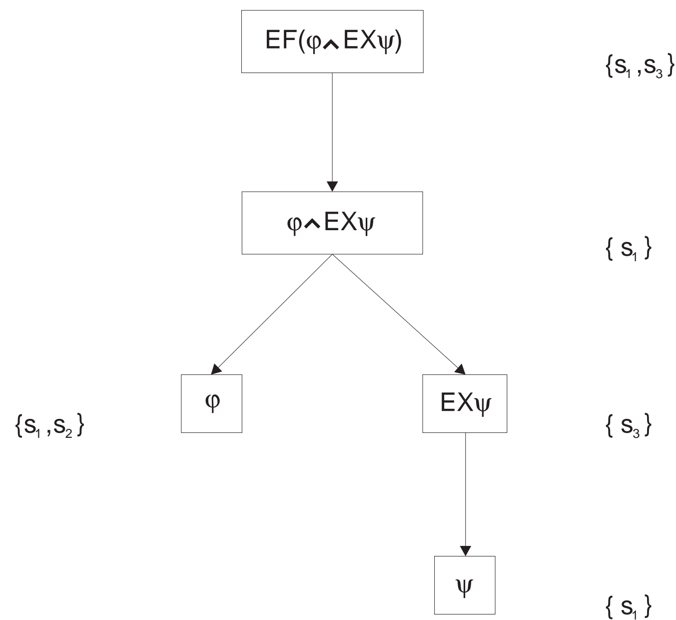

Abbildung 3.9: Syntaxbaum einer CTL-Formel und Ergebnisse der Modellprüfung

der Modellprüfung werden zunächst die atomaren Formeln betrachtet und aufbauend auf diesen Ergebnissen werden die Zustandsmengen berechnet, in denen weitere Formeln gelten. In Abbildung 3.9 wird der zugehörige Syntaxbaum von $\mathsf{EF}(\varphi \wedge \mathsf{EX}\psi)$ dargestellt. Hier werden zunächst die Zustandsmengen $Sat(\varphi)$ und $Sat(\psi)$ bestimmt, in denen die Variablen φ bzw. ψ erfüllt sind. Danach wird mittels $Sat(\psi)$ die Zustandsmenge

Abbildung 3.10: Eine Kripke-Struktur

$Sat(\mathsf{EX}\psi)$ bestimmt. Anschließend wird die Schnittmenge von $Sat(\mathsf{EX}\psi)$ und $Sat(\varphi)$ berechnet. Abschließend wird die Menge aller Zustände ermittelt, die die Zustände in $Sat(\varphi \wedge \mathsf{EX}\psi)$ erreichen. McMillan entwickelte an der Carnegie-Mellon Universität in Pittsburgh einen CTL-Modellprüfung namens SMV. Der Aufwand dieses Verfahrens ist linear zu der Modellgröße und der Länge der Formel [ClEm81b, ClGB86]. Aber bei der Modellierung größerer Schaltungen ergibt sich das Problem, daß der Zustandsraum der Modelle sehr groß werden kann. (*Explosion des Zustandsraums*, *engl. state explosion problem*).
In *EMC* erfolgte die Darstellung der Zustandsmengen durch explizite Aufzählung der in der Menge enthaltenen Zustände z.B. in Form einer Adjazenzmatrix. Dies hatte zur Folge, daß sowohl die bei der Extraktion der Kripke-Struktur aus der zu modellierenden Schaltung als auch die innerhalb des Modellprüfungsverfahren gebildeten Zustandsmengen einen sehr hohen Speicherbedarf aufwiesen.

Ein Ausweg für dieses Problem liegt in einer effizienten Darstellung von Zustandsmengen, welche eine explizite Darstellung von Zustandsmengen vermeidet. Stattdessen werden die Zustandsmengen mittels spezieller Graphen, sog. *binäre Entscheidungsgraphen* oder *BDD*'s (für *binary decision diagram*), welche eine effiziente Speicherung und Manipulation der Zustandsmengen gestatten, dargestellt. Diese Verfahren werden als *symbolische Modellprüfung* (*engl. symbolic model checking*) bezeichnet. Unter der Bezeichnung *symbolische Modellprüfung* versteht man eine Variante der Modellprüfung, bei der sowohl die innerhalb der Berechnung auftretenden Zustandsmengen als auch die Zustandsübergangsrelation gemäß den in dem vorigen Abschnitt vorgestellten Techniken symbolisch in Form von BDDs repräsentiert werden [BCLM94, BCMD90, CaCl94, ClGL93, McMi92a, McMi92b].

Im folgenden soll ein Algorithmus zur symbolischen Modellprüfung für CTL vorgestellt werden, wobei die Übergangsrelation $\mathcal{R}(\vec{v}, \vec{v'})$ als BDD der booleschen Funktion in den Variablen $\vec{v} = (v_1, ..., v_n)$, die den aktuellen Zustand beschreiben, und eine Kopie $\vec{v'} = (v'_1, ..., v'_n)$ von $\vec{v}$, die den Folgezustand beschreiben, dargestellt ist.
Der symbolische Modellprüfungsalgorithmus ist durch eine Funktion *Check* gegeben, die eine CTL-Formel als Eingabe erwartet und ein BDD liefert, das die Zustände beschreibt, in denen die CTL-Formel erfüllt ist. Bei einer atomaren Formel v liefert $Check(v_i)$ das BDD für v_i.
Hier wird das Basissystem $\{\mathsf{EX}, \mathsf{EU}, \mathsf{EG}\}$ betrachtet. Diese drei CTL-Operatoren werden mittels drei Funktionen bearbeitet, die BDDs als Eingabe erwarten und ein BDD als Ergebnis liefern:

$$\begin{aligned}
Check(\mathsf{EX}f) &= CheckEX(Check(f)) \\
Check(\mathsf{E}[f \;\mathsf{U}\; g]f) &= CheckEU(Check(f), Check(g)) \\
Check(\mathsf{EG}f) &= CheckEG(Check(f))
\end{aligned}$$

Für die Ermittlung des BDDs, welches der CTL-Formel $\neg f$ entspricht, verwendet man den Standard-BDD-Algorithmus für die Negation. Zu beachten ist dabei, daß das so erhaltene BDD einer Menge von Zustandsvariablen entspricht, die neben den tatsächlich gesuchten

auch eventuell vorhandene “nicht-existente” Vektoren enthält, denen überhaupt kein Zustand zugeordnet ist. Zur Beseitigung dieser “nicht-existierenden” Zustände muß daher das durch die Negation erhaltene Ergebnis-BDD konjunktiv mit einem BDD, welches der Menge aller existierenden Zustände entspricht, verknüpft werden. Auch die Berechnung des BDDs für die CTL-Formel $f \wedge g$ erfolgt durch Anwendung des Standard-BDD-Algorithmus für die Konjunktion.
Zur Bestimmung des BDDs für EX muß die Menge aller Zustände ermittelt werden, für welche ein Folgezustand existiert, in dem f erfüllt ist:

$$CheckEX(f(\vec{v})) = \exists \vec{v'}(f(\vec{v'}) \ \wedge \ \mathcal{R}(\vec{v}, \vec{v'}))$$

Die Berechnung des BDDs für die CTL-Formel $\mathsf{E}[f \ \mathsf{U} \ g]$ erfolgt unter Verwendung der Fixpunktcharakterisierung

$$CheckEU(f(\vec{v}), g(\vec{v})) = klFpZ(\vec{v}).(g(\vec{v}) \ \vee \ [f(\vec{v}) \ \wedge \ CheckEX(Z(\vec{v}))])$$

des Operators EU. Das BDD für die CTL-Formel $\mathsf{EG}\ f$ erhält man in ähnlicher Weise durch Ausnutzung der Fixpunktcharakterisierung des Operators EG.

$$CheckEG(f(\vec{v})) = glFp\ Z(\vec{v}).(f(\vec{v}) \ \wedge \ CheckEX(Z(\vec{v})))$$

Die größten und kleinsten Fixpunkte können mit den Funktionen $grFp$ und $klFp$ von Abschnitt 2.4 berechnet werden.

Bei der Verifikation mancher Systeme, ist man nur an bestimmten Eingabefolge interessiert, welche eine Eigenschaft unendlich oft erfüllen. Solche Bedingungen werden als *Fairneßrestriktionen* bezeichnet.
Eine Fairneß-Restriktion kann als eine Menge von Zuständen dargestellt werden, die eine logische Formel erfüllen. Ein Pfad heißt dann $fair$ unter einer Menge von Fairneß-Restriktionen, wenn jede Restriktion unendlich oft entlang dieses Pfades erfüllt ist. Die Pfadquantoren werden nur auf faire Pfade beschränkt.

Im folgenden wird die Prozedur $Check$ derart geändert, daß sie Fairneß-Restriktionen behandelt. Sei $H = \{h_1, ..., h_n\}$ eine Menge von Fairneß-Restriktionen, wobei die h_i's CTL-Formeln sind. man definiert eine neue Prozedur $CheckFair$, die eine CTL-Formel unter Fairneß-Restriktionen H überprüft. Dabei werden wie bei $Check$ drei Prozeduren $CheckFairEX$, $CheckFairEG$ und $CheckFairEU$ definiert.

Betrachtet man die Formel $\mathsf{EG}\ f$ unter einer Menge H von Fairneß-Restriktionen, so besagt die Formel, daß ein Pfad existiert, der mit dem aktuellen Zustand beginnt, wobei f in jedem Zustand dieses Pfades erfüllt ist und jede Fairneß-Restriktion aus H unendlich oft entlang dieses Pfades erfüllt ist. Sei $\mathcal{P}$ die Menge der Zustände, die diesen Pfad bilden. Dabei ist jeder Zustand s aus $\mathcal{P}$ der Anfang eines unendlichen Pfades, in dem f immer gilt und jede Fairneß-Restriktion erfüllbar ist. Somit berechnet die Prozedur $CheckFairEG$ $(f(\vec{v}))$ den größten Fixpunkt

$$grFp\ Z(\vec{v})[f(\vec{v}) \ \wedge \ \bigwedge_{i=0}^{n} CheckEX(CheckEU(f(\vec{v}), Z(\vec{v}) \ \wedge \ Check(h_i)))].$$

Die Berechnung von $\mathsf{EX}\ f$ und $\mathsf{E}[f\ \mathsf{U}\ g]$ unter Fairneß-Restriktionen ist einfacher. Die Menge aller Zustände, die den Anfang eines fairen Pfades darstellen, ist:

$$fair(\vec{v}) = CheckFair(\mathsf{EG}\ true).$$

Die Formel $\mathsf{EX}\ f$ ist in einem Zustand s unter Fairneß-Restriktionen wahr, genau dann wenn ein Folgezustand s' aus $\mathcal{S}$ existiert, so daß s' f erfüllt und s' der Anfang eines fairen Pfades ist. Dabei ist $\mathsf{EX}\ f$ unter Fairneß-Restriktionen äquivalent zu $\mathsf{EX}(f\ \wedge\ fair)$ ohne Fairneß-Restriktionen. Dann gilt:

$$CheckFairEX(f(\vec{v})) = CheckEX(f(\vec{v})\ \wedge\ fair(\vec{v})).$$

Analog ist die Formel $\mathsf{E}[f\ \mathsf{U}\ g]$ unter Fairneß-Restriktionen äquivalent zu $\mathsf{E}[f\ \mathsf{U}\ (g \wedge fair]$ ohne Fairneß-Restriktionen. Dann gilt:

$$CheckFairEX(f(\vec{v}), g(\vec{v})) = CheckEU(f(\vec{v}), g(\vec{v})\ \wedge\ fair(\vec{v})).$$

3.4 LTL-Modellprüfung

Im folgenden werden zwei Verfahren zur LTL-Modellprüfung vorgestellt. Das erste Verfahren basiert auf die Darstellung der Formel in einer speziellen Art von Kripke-Struktur sog. *Tableau* und der zweite Verfahren übersetzt ein gegebenes LTL-Modellprüfungsproblem in ein äquivalentes CTL-Modellprüfungsproblem mit Fairneß.

3.4.1 Tableau-Verfahren

Lichtenstein und Pnueli [LiPn85] haben ein Verfahren entwickelt, das linear zu der Modell-Größe und exponentiell zu der Formel-Länge ist. Obwohl dieses Verfahren linear zu der Modellgröße ist, war es bei größeren Modellen aufgrund der Explosion des Zustandsraumes nicht einsetzbar. In [ClGH94] wurde dieses Verfahren daher derart erweitert, daß symbolische Traversierungsverfahren mittels BDDs eingesetzt werden konnten. Damit war es möglich, größere Schaltungen zu verifizieren.

Gegeben sei eine Formel f und eine Kripke-Struktur $\mathcal{M}$. Es wird eine spezielle Struktur ein sog. *Tableau* $\mathcal{T}$ für f gebildet. Diese Struktur enthält alle Pfade, die f erfüllen. Bei dem Produkt $\mathcal{P}$ des Tableau's $\mathcal{T}$ mit der Kripke-Struktur $\mathcal{M}$ erhält man alle Pfade, die sowohl in $\mathcal{M}$ als auch in $\mathcal{T}$ auftreten.
Das Tableau für die Formel f ist eine Struktur $\mathcal{T} = \langle \mathcal{S}_\mathcal{T}, \mathcal{R}_\mathcal{T}, \mathcal{L}_\mathcal{T} \rangle$ über die Menge $\mathcal{V}_f$ der atomaren Formeln von f. Jeder Zustand in $\mathcal{S}_\mathcal{T}$ enthält eine Menge von atomaren Formeln von f. Die Menge der elementaren Teilformeln $el(f)$ von f erhält man mit den folgenden Regeln:

- $el(p) = \{p\}$ wenn $p \in \mathcal{V}_f$
- $el(\neg p) = el(p)$
- $el(g\ \vee\ h) = el(g) \cup el(h)$

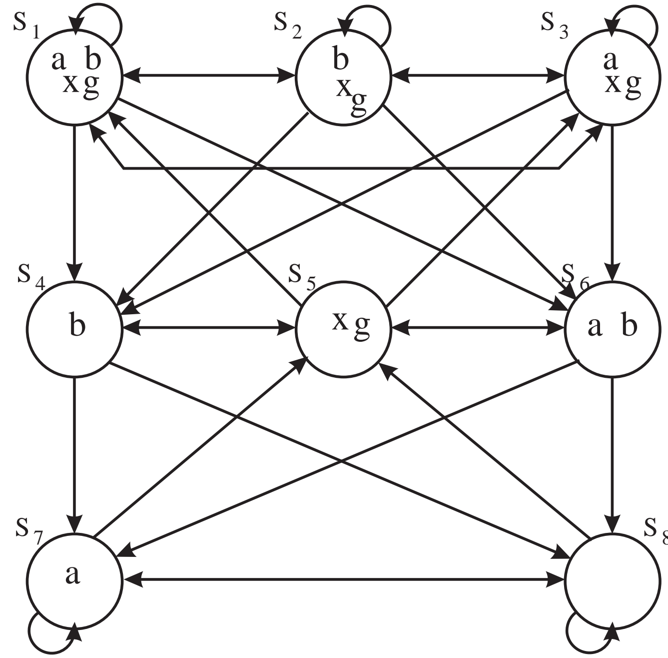

Abbildung 3.11: Tableau für $g = a \mathsf{U} h$

- $el(\mathsf{X}g) = \{\mathsf{X}g\} \cup el(g)$
- $el(gUh) = \{\mathsf{X}(g \mathsf{U} h)\} \cup el(g) \cup el(h)$

Jeder Teilformel g von f entspricht eine Menge $Sat(g)$ von Zuständen, die g erfüllen. Sat ist wie folgt definiert:

- $Sat(g) = \{s|g \in s\}$ wobei $g \in el(f)$
- $Sat(\neg g) = \{s|s \notin Sat(g)\}$
- $Sat(g \ \vee \ h) = Sat(g) \cup Sat(h)$
- $Sat(g \mathsf{U} h) = Sat(h) \cup (Sat(g) \cap Sat(\mathsf{X}(g \mathsf{U} h)))$

Jeder Untermenge von $el(g)$ entspricht ein Zustand aus $\mathcal{T}$. Das Übergangssystem $\mathcal{R}_\mathcal{T}$ eines Tableaus erhält man mit der folgenden Regel:

$$\mathcal{R}_\mathcal{T}(s, s') = \bigwedge_{\mathsf{X}g \in el(f)} s \in Sat(\mathsf{X}g) \Longleftrightarrow s' \in Sat(g)$$

Abbildung 3.11 zeigt ein Tableau für die Formel $g = a \mathsf{U} b$. Das Produkt $\mathcal{P}$ der Kripke-Struktur $\mathcal{M} = \langle \mathcal{S}, \mathcal{R}, \mathcal{L} \rangle$ mit dem Tableau $\mathcal{T} = \langle \mathcal{S}_\mathcal{T}, \mathcal{R}_\mathcal{T}, \mathcal{L}_\mathcal{T} \rangle$ ist eine Struktur $\mathcal{P} = \langle \mathcal{S}_\mathcal{P}, \mathcal{R}_\mathcal{P}, \mathcal{L}_\mathcal{P} \rangle$ mit :

- $\mathcal{S}_\mathcal{P} = \{(s, s')|s \in \mathcal{S}_\mathcal{T}, s' \in \mathcal{S}$ und $\mathcal{L}(s') \cap \mathcal{V}_f = \mathcal{L}_{\mathcal{T}(s)}\}$
- $\mathcal{R}_\mathcal{P}((s, s'), (t, t'))$ *gdw.* $\mathcal{R}_\mathcal{T}(s, t)$ und $\mathcal{R}(s', t')$

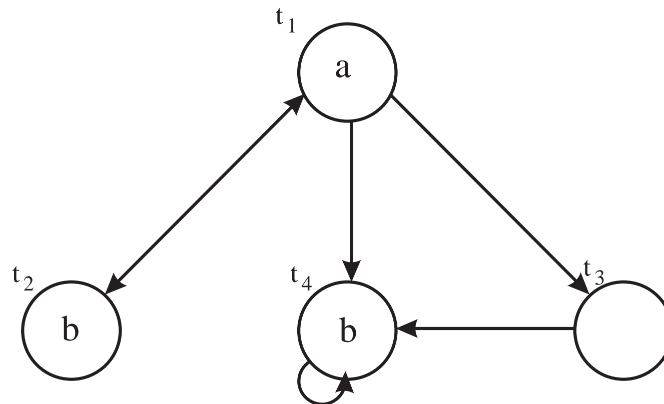

Abbildung 3.12: Eine Kripke-Struktur $\mathcal{M}$

- $\mathcal{L}_{\mathcal{P}}(s, s') = \mathcal{L}_{\mathcal{T}}(s)$

Die Menge $Sat(f)$ wird um die Zustände $(s, s') \in \mathcal{P}$ erweitert, wobei $(s, s') \in Sat(g)$ gdw. $s \in Sat(g)$. Danach wird CTL-Modellprüfung verwendet, um die Menge aller Zustände V aus $\mathcal{P}$, $V \subseteq Sat(f)$ zu ermitteln, die die Formel $\mathsf{EG}true$ unter der Fairneß-Restriktionen

$$\{Sat(\neg(g \mathsf{~U~} h) \ \vee \ h) | g \mathsf{~U~} h \text{ Teilformel von } f\}$$

erfüllen.

Theorem 3.4.1 *$\mathcal{M}, s_0 \models \mathsf{E}f$ gdw. ein Zustand t aus $\mathcal{T}$ existiert, so daß $(s, t) \in Sat(f)$ und $\mathcal{P}, (s, t) \models \mathsf{EG}true$ unter der Fairneß-Restriktionen $\{Sat(\neg(g \mathsf{~U~} h) \ \vee \ h) | g \mathsf{~U~} h$ Teilformel von $f\}$.*

Abbildung 3.13: Produkt der Kripke-Struktur $\mathcal{M}$ mit dem Tableau $\mathcal{T}$

Die Abbildung 3.13 zeigt den Produkt des Tableaus aus Abbildung 3.11 mit der Kripke-Struktur in Abbildung 3.12. In Abbildung 3.13 sind nur die Zustände (s_2, t_4), (s_3, t_1) und (s_3, t_2) aus $Sat(g)$, so daß nur die Zustände t_1, t_2 und t_4 aus $\mathcal{M}$ erfüllen die Formel $\mathsf{E}g = \mathsf{E}[a \mathrel{\mathsf{U}} b]$.

3.4.2 Verfahren von Schneider

Bei diesem Verfahren [Schn96b] wird im ersten Schritt eine Formel in eine sog. *Hardwareformel* $\mathcal{HW}$ übersetzt, abschließend wird $\mathcal{HW}$ in ein CTL-Modellprüfungsproblem mit Fairneß überführt.

Definition 3.4.1 (Hardwareformeln)
Seien $\Omega(\vec{a}, \vec{p})$, $\Phi_j(\vec{a}, \vec{p})$ und $\Psi_j(\vec{a}, \vec{p})$ für $j = 0, \ldots, f$ aussagenlogische Formeln mit den Aussagenvariablen $\vec{a}$ und $\vec{p}$ und sei $\vec{\omega}$ ein boolesches Tupel, dann stellt die folgende Formel eine Hardwareformel dar:

$$\mathcal{HW}(t_0, \vec{i}) := \left(\begin{array}{l} \exists \vec{q}. \\ \quad \left[\forall t. \left(\vec{q}^{(t_0)} \leftrightarrow \vec{\omega}\right) \wedge \left(\vec{q}^{(t+t_0+1)} \leftrightarrow \vec{\Omega}(\vec{i}^{(t+t_0)}, \vec{q}^{(t+t_0)})\right)\right] \wedge \\ \quad \bigwedge_{j=0}^{f} \left[\forall t. \Phi_j(\vec{i}^{(t+t_0)}, \vec{q}^{(t+t_0)})\right] \vee \left[\exists t. \Psi_j(\vec{i}^{(t+t_0)}, \vec{q}^{(t+t_0)})\right] \end{array} \right)$$

Die Teilformeln $\forall t. \Phi_j(\vec{i}^{(t+t_0)}, \vec{q}^{(t+t_0)})$ bzw. $\exists t. \Psi_j(\vec{i}^{(t+t_0)}, \vec{q}^{(t+t_0)})$ nennt man die Sicherheitseigenschaften bzw. die Lebendigkeitseigenschaften von $\mathcal{HW}(t_0, \vec{i})$.

Einen Vorteil der Hardwareformeln gegenüber andere Automatenformeln ist die boolesche Abgeschlossenheit. Damit kann man für jede boolesche Verknüpfung von Hardwareformeln eine äquivalente Hardwareformel finden. Daher gilt folgender Satz:

Theorem 3.4.2 (Boolesche Abgeschlossenheit der Hardwareformeln)
Sind $\mathcal{H}_1(\vec{i}, t_0), \mathcal{H}_2(\vec{i}, t_0)$ Hardwareformeln, so kann man Hardwareformeln $\mathcal{H}_\neg(\vec{i}, t_0)$, $\mathcal{H}_\wedge(\vec{i}, t_0)$, $\mathcal{H}_\vee(\vec{i}, t_0)$ finden, für die gilt:

- $\mathcal{H}_\neg(\vec{i}, t_0) = \neg\mathcal{H}_1(\vec{i}, t_0)$
- $\mathcal{H}_\wedge(\vec{i}, t_0) = \mathcal{H}_1(\vec{i}, t_0) \wedge \mathcal{H}_2(\vec{i}, t_0)$
- $\mathcal{H}_\vee(\vec{i}, t_0) = \mathcal{H}_1(\vec{i}, t_0) \vee \mathcal{H}_2(\vec{i}, t_0)$

Die Übersetzung von LTL-Modellprüfungsproblemen in FCTL-Modellprüfungsprobleme erfordert mehrere Transformationen. Die NEXT-Operatoren in einer LTL-Formel kann man stets nach außen ziehen. Dabei erhält man eine Formel in pränexe Normalform. Außerdem sind die Hardwareformeln abgeschlossen bezüglich NEXT-Operatoren, d.h. man kann für jede Hardwareformel $\mathcal{HW}$ eine Hardwareformel für $\mathsf{NEXT}\ \mathcal{HW}$ konstruieren.

Die Hardwareformeln sind bezüglich des WHEN-Operators linksabgeschlossen, d.h. für jede Hardwareformel $\mathcal{HW}$ kann eine zu $\mathcal{HW}$ WHEN b äquivalente Hardwareformel konstruiert werden, falls in b keine weiteren temporalen Operatoren vorkommen. Mit dieser Eigenschaft kann die NEXT-freie Teilformel "Bottom-up" bearbeitet und eine Hardwareformel konstruiert werden.

Enthält b wiederum temporale Operatoren, so wird die Formel b durch eine aussagenlogische Formel mit neuen variablen l ersetzt. Für die resultierende Formel kann man dann eine Hardwareformel konstruieren. Die neu eingeführten Variablen l müssen noch der Einschränkungen gehorchen, daß sie gerade den Werteverlauf von b annehmen, d.h. es muß generell $l = b$ gelten.
Nach diesem Schritt kann man eine Hardwareformel für die gesamte LTL-Formel konstruieren, indem die NEXT- und WHEN-Teilformeln sowie die booleschen Verknüpfungen von Hardwareformeln in äquivalente Hardwareformeln übersetzt werden.
Aus den generierten Gleichungen bei der Übersetzung von WHEN-Ausdrücke werden Fairneßrestriktionen abgeleitet. Dazu werden die Gleichungen weiter vereinfacht, so daß nur noch Gleichungen $\mathsf{ALWAYS}(l_j = \varphi_j)$ auftreten, bei denen φ_j genau einen temporalen Operator enthält. Aus diesen Gleichungen werden dann Büchiformeln generiert, wobei die Zustandsübergangsformeln in das Zustandsübergangssystem eingefügt werden und die Akzeptanzbedingung als Fairneßrestriktion betrachtet werden.

Aus der Übersetzung der Formel (NEXT (EVENTUAL ack)) WHEN req ergibt sich die folgende Hardwareformel :

$$\begin{array}{l}
\exists q_1.?q_0.((\\
\quad ((q_1 = 0) \ \wedge \ \mathsf{ALWAYS}\ (\mathsf{NEXT} q_1 = q_1 \ \vee \ q_0)) \ \wedge \\
\quad ((q_0 = 0) \ \wedge \ \mathsf{ALWAYS}\ (\mathsf{NEXT} q_0 = req))) \ \wedge \\
\qquad ((\mathsf{ALWAYS}\ (q_0 \ \vee \ q_1) \ \vee \ \mathsf{EVENTUAL}\ ((q_0 \ \vee \ q_1) \ \wedge \ ack))))
\end{array}$$

Im letzten Schritt wird die gewonnene Hardwareformel in ein CTL-Modellprüfungsproblem übersetzt. Dabei bildet das Zustandsübergangssystem die Kripke-Struktur und die Sicherheits- und Lebendigkeits-Eigenschaften die zu verifizierende Formel. Weil die temporalen Operatoren ohne Pfadquantoren vorkommen, werden die Teilformeln $\mathsf{A}(\mathsf{G}x \vee \mathsf{F}y)$ durch die Formel $\neg\mathsf{E}[\neg y \ \mathsf{U}\ (\neg x \wedge \mathsf{EG}\neg y)]$ ersetzt. Somit kann man die symbolischen Traversierungsverfahren auch für LTL nutzbar machen.

3.5 CTL⋆-Modellprüfung

In [EmLe85] wurde ein Algorithmus vorgestellt, der ausgehend von einem CTL-Modellprüfer und einem LTL-Modellprüfer einen CTL⋆-Modellprüfer entwickelt.

```
function CTL⋆(M, Θ)
    case Θ of
1       is_prop(Θ)     : return CTL(M, Θ);
2       φ ∧ ψ          : return (CTL⋆(M, φ) ∩ CTL⋆(M, ψ));
3       φ ∨ ψ          : return (CTL⋆(M, φ) ∪ CTL⋆(M, ψ));
4       ¬φ             : return M\ CTL⋆(M, φ);
5       is_LTL(Θ)      : return LTL(M, Θ);
6       is_CTL(Θ)      : return CTL(M, Θ);
        default        : {
7                          Θ = Φ(Eψ_1, ..., Eψ_n);
8                          s_1 = CTL⋆(M, ψ_1); insert(Q_1, s_1, M)
9                          ...
10                         s_n = CTL⋆(M, ψ_n); insert(Q_n, s_n, M)
11                         return LTL(M, Φ(Q_1, ..., Q_n));
12                         };
end CTL⋆.
```

Bei der Bearbeitung einer CTL$\star$-Formel wird die Operatorenmenge auf $\{\wedge, \vee, \neg, \mathsf{E}\}$ und die temporalen Operatoren reduziert.
In den Zeilen 7 bis 11 wird die Formel Θ in einer Menge von Zustandsformeln $\mathsf{E}\psi_i$ zerlegt. Für jede Teilmenge wird die Funktion CTL$\star$ aufgerufen und entsprechend dem Ergebnis wird die Kripke-Struktur mit einer neuen Variablen Q_i in den Zuständen aus der Zustandsmenge s_i ergänzt.
Nach der Bearbeitung aller Teilformeln werden diese Teilformeln durch die neue Variablen Q_i substituiert. Da die neu gewonnene Formel $\Phi(Q_1, ..., Q_n)$ eine LTL-Formel ist, wird die Funktion LTL$\star$ mit dieser Formel aufgerufen. So liefert dann die Funktion die Menge der Zustände in denen eine gegebene Formel erfüllt ist.

Bei diesem Verfahren kann man CTL- und LTL-Modellprüfer benutzen. Jedoch um eine effiziente Implementierung zu haben, muß auch der LTL-Modellprüfer die symbolischen Traversierungsverfahren nutzen. Das vorgestellte LTL-Modellprüfer aus Abschnitt 3.4.2 könnte hier verwendet werden um die LTL-Teilformeln zu bearbeiten.

Kapitel 4

Übersetzung von CTL$\star$ in CTL mit Fairneß

In diesem Abschnitt wird ein neues Verfahren vorgestellt, das die Vorteile der CTL-Modellprüfung nutzt, um sie auf CTL$\star$ zu übertragen. CTL-Modellprüfung wird wegen der symbolischen Darstellung von Zustandsmengen mittels BDDs und wegen des linearen Zeitaufwandes in der Praxis viel eingesetzt. Weil CTL$\star$ auch Pfadformeln enthalten kann, ist es nicht möglich, die Menge der Zustände, die ein Pfad darstellt, mittels BDD's darzustellen, und somit ist die symbolische Modellprüfung auf CTL$\star$ direkt nicht anwendbar.
Der Hauptunterschied zwischen CTL$\star$ und CTL ist, daß Pfadquantoren und temporale Operatoren bei CTL nur paarweise vorkommen dürfen. Daher gibt es bei der Übersetzung von CTL$\star$ in CTL zwei Hauptprobleme:

- (i) temporale Operatoren ohne vorausgehende Pfadquantoren
- (ii) Pfadquantoren ohne folgende temporale Operatoren

In diesem Kapitel wird gezeigt, daß zu einem gegebenen CTL$\star$-Modellprüfungsproblem ein äquivalentes CTL-Modellprüfungsproblem existiert.
Die Übersetzung verläuft in zwei Schritten. Nach dem ersten Schritt erhält man eine Formel, in der nach den Pfadquantoren nur temporale Operatoren vorkommen, d.h. Problem (i) wurde gelöst. Diese Menge von Formeln wird im folgenden als CTL+ bezeichnet. Im zweiten Schritt werden die temporalen Operatoren, die ohne Pfadquantoren auftreten, durch Variablen ersetzt und Fairneßrestriktionen gebildet, d.h. Problem (ii) wird gelöst.

4.1 Übersetzung von CTL$\star$ in CTL+

CTL+ ist eine Untermenge von CTL$\star$. Dabei wird die Syntax folgendermaßen beschränkt:

Definition 4.1.1 (Syntax von CTL+)
Sei $\mathcal{V}$ die Menge der aussagenlogischen Variablen, dann definiert man die Syntax von

Zustandsformeln bzw. Pfadformeln für CTL+ wie folgt:
Zustandsformeln $\mathcal{SF}_{\mathcal{V}}^{+}$:

- *Jede aussagenlogische Formel ist eine Zustandsformel.*
- *Sind* φ *und* ψ *Zustandsformeln, so sind auch* $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi$ *Zustandsformeln.*
- *Sind* φ *und* ψ *Pfadformeln, so sind* $\mathsf{EF}\varphi$, $\mathsf{EG}\varphi$, $\mathsf{EX}\varphi$, $\mathsf{E}[\varphi\ \mathsf{W}\ \psi]$, $\mathsf{E}[\varphi\ \mathsf{U}\ \psi]$, $\mathsf{AF}\varphi$, $\mathsf{AG}\varphi$, $\mathsf{AX}\varphi$, $\mathsf{A}[\varphi\ \mathsf{W}\ \psi]$, $\mathsf{A}[\varphi\ \mathsf{U}\ \psi]$ *Zustandsformeln.*

Pfadformeln $\mathcal{P}_{\mathcal{SF}_{\mathcal{V}}^{+}}$:

- *Jede Zustandsformel ist eine Pfadformel.*
- *Sind* φ *und* ψ *Pfadformeln, so sind auch* $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi$ *Pfadformeln.*
- *Sind* φ *und* ψ *Pfadformeln, so sind auch* $\mathsf{X}\varphi$, $\varphi\ \mathsf{U}\ \psi$,$\varphi\ \mathsf{W}\ \psi$, $\mathsf{F}\varphi$, $\mathsf{G}\varphi$ *Pfadformeln.*

CTL+ ist die Menge der Zustandsformeln $\mathcal{SF}_{\mathcal{V}}^{+}$

Beim Problem (*ii*) kommt nach einem Pfadquantor ein temporaler Operator oder ein boolescher Operator (Negation, Disjunktion oder Konjunktion). In erstem Fall liegt die Teilformel in CTL. Im zweiten Fall muß der Pfadquantor nach innen geschoben werden. Hierbei treten drei Fälle auf:

- Die Pfadquantoren werden ohne weiteres nach innen geschoben (Satz 4.1.1 ($\mathcal{T}_1$) und ($\mathcal{T}_2$))
- Die Pfadquantoren werden eliminiert (($\mathcal{T}_3$),($\mathcal{T}_4$))
- Auf E folgt eine Konjunktion oder auf A eine Disjunktion. Hier muß die Konjunktion in eine Disjunktion oder umgekehrt umgewandelt werden (Satz 4.1.2).

Mit dem folgenden Satz kann man die ersten beiden Fälle beseitigen:

Theorem 4.1.1
Seien φ *und* ψ *beliebige Formeln. Dann gilt:*

- ($\mathcal{T}_1$) $\mathcal{M}, s_0 \models \mathsf{A}(\varphi \wedge \psi) \Longleftrightarrow \mathcal{M}, s_0 \models (\mathsf{A}\varphi \wedge \mathsf{A}\psi)$
- ($\mathcal{T}_2$) $\mathcal{M}, s_0 \models \mathsf{E}(\varphi \vee \psi) \Longleftrightarrow \mathcal{M}, s_0 \models (\mathsf{E}\varphi \vee \mathsf{E}\psi)$
- ($\mathcal{T}_3$) $\mathcal{M}, s_0 \models \mathsf{A}\neg\varphi \Longleftrightarrow \mathcal{M}, s_0 \models \neg\mathsf{E}\varphi$
- ($\mathcal{T}_4$) $\mathcal{M}, s_0 \models \mathsf{E}\neg\varphi \Longleftrightarrow \mathcal{M}, s_0 \models \neg\mathsf{A}\varphi$

Ist φ *eine Zustandsformel, so gilt:*

- ($\mathcal{T}_5$) $\mathcal{M}, s_0 \models \mathsf{E}\varphi \Longleftrightarrow \mathcal{M}, s_0 \models \varphi$
- ($\mathcal{T}_6$) $\mathcal{M}, s_0 \models \mathsf{A}\varphi \Longleftrightarrow \mathcal{M}, s_0 \models \varphi$
- ($\mathcal{T}_7$) $\mathcal{M}, s_0 \models \mathsf{E}(\varphi \wedge \psi) \Longleftrightarrow \mathcal{M}, s_0 \models (\varphi \wedge \mathsf{E}\psi)$
- ($\mathcal{T}_8$) $\mathcal{M}, s_0 \models \mathsf{A}(\varphi \vee \psi) \Longleftrightarrow \mathcal{M}, s_0 \models (\varphi \vee \mathsf{A}\psi)$

Beweis:

$(\mathcal{T}_1)$:

$$
\begin{array}{lll}
& \mathcal{M}, s_0 \models \mathsf{A}(\varphi \ \wedge \ \psi) & \\
\Longleftrightarrow & \forall \vec{\pi} \in \mathcal{M}, \vec{\pi} = (s_0, \ldots) \ : \mathcal{M}, \vec{\pi} \models (\varphi \ \wedge \ \psi) & (\mathcal{S}3) \\
\Longleftrightarrow & \forall \vec{\pi} \in \mathcal{M}, \vec{\pi} = (s_0, \ldots) \ : (\mathcal{M}, \vec{\pi} \models \varphi \text{ und } \mathcal{M}, \vec{\pi} \models \psi) & (\mathcal{P}2) \\
\Longleftrightarrow & (\forall \vec{\pi} \in \mathcal{M}, \vec{\pi} = (s_0, \ldots) \ : \mathcal{M}, \vec{\pi} \models \varphi) \text{ und} & \\
& (\forall \vec{\pi} \in \mathcal{M}, \vec{\pi} = (s_0, \ldots) \ : \mathcal{M}, \vec{\pi} \models \psi) & \\
\Longleftrightarrow & \mathcal{M}, s_0 \models \mathsf{A}\varphi \text{ und } \mathcal{M}, s_0 \models \mathsf{A}\psi & (\mathcal{S}3) \\
\Longleftrightarrow & \mathcal{M}, s_0 \models (\mathsf{A}\varphi \ \wedge \ \mathsf{A}\psi) & (\mathcal{S}2)
\end{array}
$$

$(\mathcal{T}_2)$: analog zu $(\mathcal{T}_1)$

$(\mathcal{T}_5)$:

$$
\begin{array}{lll}
& \mathcal{M}, s_0 \models \mathsf{E}\varphi \text{ und } \varphi \text{ ist eine Zustandsformel} & \\
\Longleftrightarrow & \exists \vec{\pi} \in \mathcal{M}, \ \vec{\pi} = (s_0, \ldots) \ : \mathcal{M}, \vec{\pi} \models \varphi & (\mathcal{S}3) \\
\Longleftrightarrow & \exists \vec{\pi} \in \mathcal{M}, \ \vec{\pi} = (s_0, \ldots) \ : \mathcal{M}, s_0 \models \varphi & (\mathcal{P}1) \\
\Longleftrightarrow & \mathcal{M}, s_0 \models \varphi &
\end{array}
$$

$(\mathcal{T}_6)$: analog zu $(\mathcal{T}_5)$

$(\mathcal{T}_7)$:

$$
\begin{array}{lll}
& \mathcal{M}, s_0 \models \mathsf{E}(\varphi \ \wedge \ \psi) \text{ und } \varphi \text{ ist eine Zustandsformel} & \\
\Longleftrightarrow & \exists \vec{\pi} \in \mathcal{M}, \ \vec{\pi} = (s_0, \ldots) \ : \mathcal{M}, \vec{\pi} \models (\varphi \ \wedge \ \psi) & (\mathcal{S}3) \\
\Longleftrightarrow & \exists \vec{\pi} \in \mathcal{M}, \ \vec{\pi} = (s_0, \ldots) \ : (\mathcal{M}, \vec{\pi} \models \varphi \text{ und } \mathcal{M}, \vec{\pi} \models \psi) & (\mathcal{P}2) \\
\Longleftrightarrow & \exists \vec{\pi} \in \mathcal{M}, \ \vec{\pi} = (s_0, \ldots) \ : (\mathcal{M}, s_0 \models \varphi \text{ und } \mathcal{M}, \vec{\pi} \models \psi) & (\mathcal{P}1) \\
\Longleftrightarrow & (\mathcal{M}, s_0 \models \varphi) \text{ und } (\exists \vec{\pi} \in \mathcal{M}, \ \vec{\pi} = (s_0, \ldots) : \mathcal{M}, \vec{\pi} \models \psi) & \\
\Longleftrightarrow & (\mathcal{M}, s_0 \models \varphi) \text{ und } (\mathcal{M}, s_0 \models \mathsf{E}\psi) & (\mathcal{S}3) \\
\Longleftrightarrow & \mathcal{M}, s_0 \models (\varphi \ \wedge \ \mathsf{E}\psi) & (\mathcal{S}2)
\end{array}
$$

$(\mathcal{T}_8)$: analog zu $(\mathcal{T}_7)$

Der letzte Fall für die Übersetzung von CTL⋆ in CTL+ ist das Auftreten von Pfadquantoren E bzw. A auf eine Konjunktion bzw. Disjunktion. Die beiden Fälle können auf einen Fall reduziert werden, indem man die Äquivalenz $\mathsf{A}\varphi = \neg\ \mathsf{E}\neg\varphi$ benutzt. Um die Übersetzung übersichtlicher zu machen, werden die temporalen Operatoren F, G und U durch W ersetzt.
Die Idee der Lösung beruht auf der Betrachtung der möglichen Reihenfolgen des Auftretens der Ereignisse. Bei der Betrachtung der Formel $[(x_1 \ \mathsf{W} \ b_1) \ \wedge \ (x_2 \ \mathsf{W} \ b_2)]$ können zwei Fälle auftreten:

- das Ereignis b_1 tritt vor oder gleichzeitig mit b_2 auf
- das Ereignis b_2 tritt vor oder gleichzeitig mit b_1 auf

Bei der Betrachtung des ersten Falles kann man folgern, daß, wenn b_1 erfüllt ist, auch x_1 und $(x_2 \ \mathsf{W} \ b_2)$ erfüllt sein sollen. Der zweite Fall ist analog zum ersten Fall. Durch Kombination der beiden Fälle erhält man eine Disjunktion und kann den Pfadquantor zunächst nach innen schieben. Ist die Anzahl der temporalen Operatoren bei der Konjunktion mehr als zwei, so kann man zunächst die Konjunktion der ersten beiden temporalen Operatoren durch eine Disjunktion ersetzen, danach wird die Formel ausmultipliziert und so erhält

man eine Disjunktion von Konjunktionen. Die Anzahl der temporalen Operatoren bei den Konjunktionen ist um eins reduziert. So wird das Verfahren rekursiv angewendet, bis man eine Disjunktion von temporalen Operatoren erhält.
Die verschiedenen Fälle, die auftreten können, werden im folgenden Satz behandelt:

Theorem 4.1.2 (Disjunktive Form einer Konjunktion temp. Operatoren)
Seien x_1, x_2, b, b_1 und b_2 CTL⋆-Formel, dann gelten folgende Gleichungen:

- $[x_1 \mathsf{W}\, b] \wedge [x_2 \mathsf{W}\, b] = [(x_1 \wedge x_2) \mathsf{W}\, b]$
- $[x_1 \mathsf{W}\, b] \wedge \neg[x_2 \mathsf{W}\, b] = \neg[(\neg x_1 \vee x_2) \mathsf{W}\, b]$
- $\neg[x_1 \mathsf{W}\, b] \wedge \neg[x_2 \mathsf{W}\, b] = \neg[(x_1 \vee x_2) \mathsf{W}\, b]$
- $[x_1 \mathsf{W}\, b_1] \wedge [x_2 \mathsf{W}\, b_2] = \begin{pmatrix} [(x_1 \wedge b_1 \wedge [x_2 \mathsf{W}\, b_2]) \mathsf{W}\, (b_1 \vee b_2)] \vee \\ [(x_2 \wedge b_2 \wedge [x_1 \mathsf{W}\, b_1]) \mathsf{W}\, (b_1 \vee b_2)] \end{pmatrix}$
- $[x_1 \mathsf{W}\, b_1] \wedge \neg[x_2 \mathsf{W}\, b_2] = \begin{pmatrix} \neg[(\neg x_1 \vee \neg b_1 \vee [x_2 \mathsf{W}\, b_2]) \mathsf{W}\, (b_1 \vee b_2)] \vee \\ \neg[(x_2 \vee \neg b_2 \vee \neg[x_1 \mathsf{W}\, b_1]) \mathsf{W}\, (b_1 \vee b_2)] \end{pmatrix}$
- $\neg[x_1 \mathsf{W}\, b_1] \wedge \neg[x_2 \mathsf{W}\, b_2] = \begin{pmatrix} \neg[(x_1 \vee \neg b_1 \vee [x_2 \mathsf{W}\, b_2]) \mathsf{W}\, (b_1 \vee b_2)] \vee \\ \neg[(x_2 \vee \neg b_2 \vee [x_1 \mathsf{W}\, b_1]) \mathsf{W}\, (b_1 \vee b_2)] \end{pmatrix}$
- $\mathsf{X} x_1 \wedge [x_2 \mathsf{W}\, b_2] = [x_2 \wedge b_2) \wedge \mathsf{X} x_1] \vee \mathsf{X}[x_1 \wedge [x_2 \mathsf{W}\, b_2]]$
- $\mathsf{X} x_1 \wedge \neg[x_2 \mathsf{W}\, b_2] = [\neg x_2 \wedge b_2) \wedge \mathsf{X} x_1] \vee \mathsf{X}[x_1 \wedge \neg[x_2 \mathsf{W}\, b_2]]$

Mit dem Satz 4.1.2 hat man alle Probleme, die bei der Übersetzung von CTL⋆ in CTL+ auftreten, gelöst. Daher gilt folgender Satz:

```
function Stern2Plus(Θ)
    case Θ of
1       v ∈ 𝒱            : return v;
2       ¬φ               : return ¬Stern2Plus(φ);
3       [E,A] φ          : return Stern2Plus_Pfad(φ,[E,A]);
4       [GG,FF] φ        : return Stern2Plus([G,F]φ);
5       [GX,FX] φ        : return X(Stern2Plus([G,F]φ));
6       [G,F,X] φ        : return [G,F,X] (Stern2Plus(φ));
7       φ [U,W,∧,∨] ψ    : return (Stern2Plus(φ) [U,W,∧,∨] Stern2Plus(ψ));
end Stern2Plus.
```

Theorem 4.1.3 (Übersetzung von CTL⋆ in CTL+)
Die Funktion Stern2Plus *übersetzt eine gegebene CTL⋆-Formel in eine CTL+-Formel*

```
function Stern2Plus_Pfad(Θ, P)
    case Θ of
1       Θ ∈ SF_V  : return Stern2Plus(Θ);
2       ¬φ        :
3               if (P = E) return ¬Stern2Plus_Pfad(φ,A);
4               else  return ¬Stern2Plus_Pfad(φ,E);
5       [GG,FF] φ : return Stern2Plus_Pfad([G,F] φ,P);
6       [GX,FX]φ  : return PX(Stern2Plus_Pfad([G,F] φ,P));
7       [G,F,X] φ : return P[G,F,X](Stern2Plus_Pfad(φ,P));
8       φ U ψ     : return P(Stern2Plus(φ) U Stern2Plus(ψ));
9       φ W ψ     : return P(Stern2Plus_Pfad(φ,P) W Stern2Plus(ψ));
10      φ [∧,∨] ψ :
11              if (φ ∈ SF_V)
12                  return (Stern2Plus(φ) [∧,∨] Stern2Plus_Pfad(ψ,P));
13              else if (ψ ∈ SF_V)
14                  return (Stern2Plus_Pfad(φ,P) [∧,∨] Stern2Plus(ψ));
15              else if (P = E) and (Θ = φ ∨ ψ)
16                  return (Stern2Plus_Pfad(φ,P) ∨ Stern2Plus_Pfad(ψ,P));
17              else if (P = A) and (Θ = φ ∧ ψ)
18                  return (Stern2Plus_Pfad(φ,P) ∧ Stern2Plus_Pfad(ψ,P));
19              else  return Konj_Stern2Plus(Θ);
end Stern2Plus_Pfad.
```

Beweis: Gegeben sei eine CTL⋆-Formel φ. Die Funktion Stern2Plus bearbeitet temporale aussagenlogische Formel rekursiv "top-down". Damit das Ergebnis dieser Funktion in CTL+ liegt, soll sie eine Zustandsformel als Eingabe erhalten. Dabei werden während der Verarbeitung auch Pfadformeln bearbeitet. In den Zeilen 1, 2, 6 und 7 liegt die Formel bereits in CTL+. Die Zeilen 4 und 5 eliminieren redundante Operatoren. Die eigentliche Aufgabe dieser Funktion ist, Pfadquantoren derart zu setzen, daß sie nur von temporalen Operatoren gefolgt sind. Diesen Schritt erledigt die Funktion Stern2Plus$_{Pfad}$.
Die Funktion Stern2Plus$_{Pfad}$ arbeitet ebenfalls "top-down". In der Zeile 1 werden Pfadquantoren eliminiert, weil sie von Zustandsformeln gefolgt sind (Satz 4.1.1 ($\mathcal{T}_5$) und ($\mathcal{T}_6$)). In den Zeilen 3 und 4 werden die Gleichungen ($\mathcal{T}_3$) und ($\mathcal{T}_4$) verwendet. Bei der Zeile 5 werden auch redundante Teilformeln eliminiert. Nach dem Vertauschen von G oder F mit X (Zeile 6) folgt ein Pfadquantor einem temporalen Operator X (CTL+). Danach wird die Teilformel mit einem Pfadquantor ergänzt und die Funktion Stern2Plus$_{Pfad}$ aufgerufen (Satz 4.1.2). In den Zeilen 7 und 9 werden die Teilformeln mit Pfadquantoren ergänzt und weiter bearbeitet. Hier sieht man, daß die Gleichungen ($\mathcal{G}_{1.1}$) und ($\mathcal{G}_{1.2}$) von Satz 4.1.4 verletzt sind, weil man keine Erkenntnisse hat, ob ψ eine Zustandsformel ist oder nicht. ist ψ eine Zustandsformel, dann gilt die Ergänzung. Ist ψ aber eine Pfadformel, dann liegt die Formel nicht in CTL und daher muß sie durch eine Variable ersetzt werden, die eine Zustandsformel ist. So ist dann die Ersetzung richtig. Dieser Schritt wird im nächsten Abschnitt erläutert. Bei dem UNTIL (Zeile 8) werden die rechte und linke Seite

mit der Funktion Stern2Plus bearbeitet, weil hier keine Ergänzung stattgefunden hat.
In den Zeilen 12, 14, 16 und 18 werden entweder Pfadquantoren nach innen geschoben (Satz 4.1.1 ($\mathcal{T}_1$) und ($\mathcal{T}_2$)) oder Pfadquantoren eliminiert (($\mathcal{T}_5$) und ($\mathcal{T}_6$)).
Die Funktion Konj_Stern2Plus bearbeitet ein E, das auf eine Konjunktion folgt, gemäß Satz 4.1.2. Folgt ein A auf eine Disjunktion, so wird die Äquivalenz $\mathsf{A}\varphi = \neg\mathsf{E}\neg\varphi$ verwendet. So gibt die Funktion Stern2Plus eine Formel zurück, die in CTL+ liegt. In vielen Fällen kann man aus einer CTL⋆-Formel durch Ergänzen von Pfadquantoren eine äquivalente CTL-Formel gewinnen. Die Eigenschaft gilt nur bei F, G und X und ist bezüglich WHEN-Operatoren linksrekursiv nur dann, wenn die rechte Seite eine Zustandsformel ist.

Theorem 4.1.4 (Ergänzung von Pfadquantoren)
Ist ψ eine Zustandsformel und φ eine Pfadformel, so gelten folgende Gleichungen:

$(\mathcal{G}_{1.1}) \models \mathsf{E}[\varphi\ \mathsf{W}\ \psi] = \mathsf{E}[(\mathsf{E}\varphi)\ \mathsf{W}\ \psi]$	$(\mathcal{G}_{1.2}) \models \mathsf{A}[\varphi\ \mathsf{W}\ \psi] = \mathsf{A}[(\mathsf{A}\varphi)\ \mathsf{W}\ \psi]$
$(\mathcal{G}_{2.1}) \models \mathsf{EF}\varphi = \mathsf{EF}(\mathsf{E}\varphi)$	$(\mathcal{G}_{2.2}) \models \mathsf{AF}\varphi = \mathsf{AF}(\mathsf{A}\varphi)$
$(\mathcal{G}_{3.1}) \models \mathsf{EG}\varphi = \mathsf{EG}(\mathsf{E}\varphi)$	$(\mathcal{G}_{3.2}) \models \mathsf{AG}\varphi = \mathsf{AG}(\mathsf{A}\varphi)$
$(\mathcal{G}_{4.1}) \models \mathsf{EX}\varphi = \mathsf{EX}(\mathsf{E}\varphi)$	$(\mathcal{G}_{4.2}) \models \mathsf{AX}\varphi = \mathsf{AX}(\mathsf{A}\varphi)$

Beweis:

$(\mathcal{G}_{1.1})$:

$$
\begin{aligned}
& \mathcal{M}, s_0 \models \mathsf{E}[(\mathsf{E}\varphi)\ \mathsf{W}\ \psi] \\
\Longleftrightarrow\ & \exists\vec{\pi} \in \mathcal{M}, \vec{\pi} := (s_0, ...) \quad [\mathcal{M}, \vec{\pi} \models (\mathsf{E}\varphi)\ \mathsf{W}\ \psi] \ \vee\ [\mathcal{M}, \vec{\pi} \models \mathsf{G}\neg\psi] \\
\Longleftrightarrow\ & \exists\vec{\pi} \in \mathcal{M}, \vec{\pi} := (s_0, ...), \exists i \in \mathbb{N}. [(\mathcal{M}, \vec{\pi}_i \models (\mathsf{E}\varphi) \ \wedge\ \psi) \wedge \\
& \quad (\forall j < i. \mathcal{M}, s_j \models \neg\psi)] \vee \\
& \quad [\mathcal{M}, \vec{\pi} \models \mathsf{G}\neg\psi] \\
\Longleftrightarrow\ & \exists\vec{\pi} \in \mathcal{M}, \vec{\pi} := (s_0, ...), \exists i \in \mathbb{N}. [(\mathcal{M}, \vec{\pi}_i \models (\mathsf{E}\varphi) \ \wedge\ \mathcal{M}, \vec{\pi}_i \models \psi) \wedge \\
& \quad (\forall j < i. \mathcal{M}, s_j \models \neg\psi)] \vee \\
& \quad [\mathcal{M}, \vec{\pi} \models \mathsf{G}\neg\psi] \\
\Longleftrightarrow\ & \exists\vec{\pi} \in \mathcal{M}, \vec{\pi} := (s_0, ...), \exists i \in \mathbb{N}. [(\mathcal{M}, s_i \models (\mathsf{E}\varphi) \ \wedge\ \mathcal{M}, s_i \models \psi) \wedge \\
& \quad (\forall j < i. \mathcal{M}, s_j \models \neg\psi)] \vee \\
& \quad [\mathcal{M}, \vec{\pi} \models \mathsf{G}\neg\psi] \\
\Longleftrightarrow\ & \exists\vec{\pi} \in \mathcal{M}, \vec{\pi} := (s_0, ...), \exists i \in \mathbb{N}. [[\exists\vec{\eta} \in \mathcal{M}, \vec{\eta} := (s_0, ..., s_i, ...) \ \wedge\ \mathcal{M}, \vec{\eta}_i \models \varphi] \wedge \\
& \quad [\mathcal{M}, s_i \models \psi] \wedge \\
& \quad (\forall j < i. \mathcal{M}, s_j \models \neg\psi)] \vee \\
& \quad [\mathcal{M}, \vec{\pi} \models \mathsf{G}\neg\psi] \\
\Longleftrightarrow\ & \exists\vec{\pi} \in \mathcal{M}, \vec{\pi} := (s_0, ...), \exists i \in \mathbb{N}. \exists\vec{\eta} \in \mathcal{M}. \\
& \quad [[\vec{\eta} := (s_0, ..., s_i, ...) \ \wedge\ \mathcal{M}, \vec{\eta}_i \models \varphi] \wedge \\
& \quad [\mathcal{M}, s_i \models \psi] \wedge \\
& \quad (\forall j < i. \mathcal{M}, s_j \models \neg\psi)] \vee \\
& \quad [\mathcal{M}, \vec{\pi} \models \mathsf{G}\neg\psi] \\
\Longleftrightarrow\ & \exists\vec{\pi} \in \mathcal{M}, \vec{\pi} := (s_0, ...), \exists i \in \mathbb{N}. \exists\vec{\eta} \in \mathcal{M}. \\
& \quad [[\vec{\eta} := (s_0, ..., s_i, ...) \ \wedge\ \mathcal{M}, \vec{\eta}_i \models \varphi] \wedge \\
& \quad [\mathcal{M}, \vec{\eta}_i \models \psi] \wedge
\end{aligned}
$$

$$
\begin{array}{lll}
 & & (\forall j < i.\mathcal{M}, s_j \models \neg\psi)] \vee \\
 & & [\mathcal{M}, \vec{\pi} \models \mathsf{G}\neg\psi] \\
\iff & \exists\vec{\pi} \in \mathcal{M}, \vec{\pi} := (s_0, ...), \exists i \in \mathbb{N}. & \exists\vec{\eta} \in \mathcal{M}. \\
 & & [[\vec{\eta} := (s_0, ..., s_i, ...) \ \wedge \ \mathcal{M}, \vec{\eta}_i \models (\varphi \ \wedge \ \psi)] \ \wedge \\
 & & (\forall j < i.\mathcal{M}, s_j \models \neg\psi)] \vee \\
 & & [\mathcal{M}, \vec{\pi} \models \mathsf{G}\neg\psi] \\
\iff & \exists\vec{\eta} \in \mathcal{M}.\vec{\eta} := (s_0, ..., s_i, ...), & [\mathcal{M}, \vec{\eta} \models (\varphi \ \mathsf{W} \ \psi)] \\
\iff & \mathcal{M}, s_0 \models \mathsf{E}[\varphi \ \mathsf{W} \ \psi] &
\end{array}
$$

Der Beweis für ($\mathcal{G}_{1.2}$) ist analog zu ($\mathcal{G}_{1.1}$). Die Beweise der Gleichungen ($\mathcal{G}_{2.*}$), ($\mathcal{G}_{3.*}$) und ($\mathcal{G}_{4.*}$) sind ähnlich. Hier wird beispielsweise der Beweis für die Gleichung ($\mathcal{G}_{2.1}$) durchgeführt.

($\mathcal{G}_{2.1}$):

$$
\begin{array}{llll}
 & \mathcal{M}, s_0 \models \mathsf{EFE}\varphi & & \\
\iff & \exists\vec{\pi} \in \mathcal{M}.\vec{\pi} := (s_0, ...) & & : \mathcal{M}, \vec{\pi} \models \mathsf{FE}\varphi \\
\iff & \exists\vec{\pi} \in \mathcal{M}.\vec{\pi} := (s_0, ...), & \exists i \in \mathbb{N}. & : \mathcal{M}, \vec{\pi}_i \models \mathsf{E}\varphi \\
\iff & \exists\vec{\pi} \in \mathcal{M}.\vec{\pi} := (s_0, ...), & \exists i \in \mathbb{N}. & : s_i \in \vec{\pi} \ \wedge \ \mathcal{M}, s_i \models \mathsf{E}\varphi \\
\iff & \exists\vec{\eta} \in \mathcal{M}.\vec{\eta} := (s_0, ..., s_i, ...) & & : \mathcal{M}, \vec{\eta}_i \models \varphi \\
\iff & \exists\vec{\eta} \in \mathcal{M}.\vec{\eta} := (s_0, ..., s_i, ...) & & : \mathcal{M}, \vec{\eta} \models \mathsf{F}\varphi \\
\iff & \mathcal{M}, s_0 \models \mathsf{EF}\varphi & &
\end{array}
$$

4.2 Übersetzung von CTL+ in CTL mit Fairneß

Nachdem man eine CTL⋆-Formel in CTL+ übersetzt hat, liegt das Problem darin, daß temporale Operatoren ohne Pfadquantoren vorkommen. Diese Teilformeln werden dann durch neue Variablen ersetzt, die immer äquivalent zu diesen Formeln sein müssen. Dabei werden neue Definitionen der Form $\mathsf{G}(\ell_i = \psi_i)$ eingefügt, wobei ψ zur Vereinfachung nur einen temporalen Operator enthalten darf, und jedes Vorkommen von ψ_i durch ℓ_i ersetzt wird. Somit erhält man dann eine Formel, bei der keine temporale Operatoren ohne Pfadquantoren vorkommen. Weil die Formel bereits in CTL+ war, ist sie dann in CTL, weil die Pfadquantoren und temporalen Operatoren paarweise vorkommen.

Theorem 4.2.1

Gegeben sei eine Zustandsformel $\Phi \in \mathcal{SF}_{\mathcal{V}}$. Es existiert eine Menge Υ:=$\{\mathsf{G}(\ell_1 = \psi_1), ..., \mathsf{G}(\ell_n = \psi_n)\}$ mit neuen Variablen ℓ_i und eine CTL-Formel Ψ, so daß gilt:

$$\Phi = \forall \ell_1, ..., \ell_n. \bigwedge_{i=0}^{n} [\mathsf{G}(\ell_i = \psi_i)] \rightarrow \Psi$$

```
function Plus2CTL(Θ)
    case Θ of
1       v ∈ 𝒱                 : return v
2       ¬φ                    : return ¬Plus2CTL(φ)
3       [X,G,F] φ             : return Auslagern(Θ)
4       [A,E] [X,G,F] φ       : return [A,E] [X,G,F] (Plus2CTL(φ))
5       [A,E](φ [W,U] ψ)      : return [A,E] ((Plus2CTL(φ)) [W,U] (Plus2CTL(ψ)))
6       φ [∧,∨] ψ             : return (Plus2CTL(φ) [∧,∨] Plus2CTL(ψ))
7       φ [W,U] ψ             : return Auslagern(Θ)
end Plus2CTL.
```

Die Teilformel Ψ erhält man, wenn man die Funktion Plus2CTL auf die Formel Φ anwendet. Dabei wird geprüft, ob die bearbeitete Teilformel bereits in CTL liegt. Kommt ein temporaler Operator ohne Pfadquantor vor (Zeile 3 und 7), so wird die Funktion Auslagern aufgerufen, die eine aussagenlogische Formel zurückliefert und eine Menge von Definitionen der Form $\mathsf{G}(\ell_i = \psi_i)$ bildet. Weil die bearbeitete Formel bereits in CTL+ liegt, kommt es nicht vor, daß nach einem Pfadquantor eine Konjunktion oder eine Disjunktion kommt.

```
ϒ:= {};

function Auslagern (Θ)
  case Θ of
1    v ∈ 𝒱          :  return v;
2    ¬φ             :  return ¬Auslagern(φ);
3    [X,G,F] φ      :  ℓ:= new_var();
                       ϒ:= ϒ ∪{G(ℓ = [X, G,F] (Auslagern(φ))};
                       return ℓ;
4    [A,E] φ        :  return Auslagern(φ);
5    φ [∧,∨] ψ      :  return (Auslagern(φ) [∧,∨] Auslagern(ψ));
6    φ [U,W] ψ      :  ℓ:= new_var();
                       ϒ:= ϒ ∪{G(ℓ = (Auslagern(φ)) [U,W] (Auslagern(ψ))};
                       return ℓ;
end Auslagern.
```

Somit hat man aus einer CTL⋆-Formel eine CTL-Formel und eine Menge von Gleichungen erhalten. Aus diesen Gleichungen bildet man Fairneßrestriktionen, damit bei der Verifikation nur die Pfade betrachtet werden, in der diese Fairneßrestriktionen unendlich oft gelten (sog. *faire Pfade*).

4.3 Bildung von Fairneßrestriktionen

Die Menge Υ enthält Gleichungen der Form $\mathsf{G}(\ell_i = \psi_i)$ wobei ψ_i nur einen temporalen Operator enthält. Mit dem folgenden Satz kann man diese in zusätzliche Zustandsübergangs-

gleichungen und eine Fairneßrestriktion umwandeln. Das Zustandsübergangssystem wird durch diese Zustandsübergangsgleichungen ergänzt und die Fairneßrestriktionen werden entsprechend behandelt, daß die gebildete Kripke-Struktur entweder nur die fairen Pfaden enthält oder die fairen Pfaden markiert werden, so daß bei der Modellprüfung nur diese Pfade betrachtet werden.

Theorem 4.3.1 (Bildung von Fairneßrestriktionen)

- $\mathsf{G}(\ell = x\ \mathsf{W}\ b) := \left(\begin{array}{l} \exists pq. \\ [(p=1) \wedge (\mathsf{X}p = \ell \wedge p \wedge (x \vee \neg b) \vee \neg\ell \wedge q \wedge b \wedge \neg x)] \wedge \\ [(q=1) \wedge (\mathsf{X}q = \neg\ell \wedge q \wedge \neg(x \wedge b) \vee \ell \wedge p \wedge b \wedge x)] \wedge \\ [\mathsf{GF}p] \end{array}\right)$
- $\mathsf{G}(\ell = x\ \mathsf{U}\ b) := \left(\begin{array}{l} \exists pq. \\ [(p=1) \wedge (\mathsf{X}p = \ell \wedge p \wedge (x \vee b) \vee \neg\ell \wedge q \wedge \neg x \wedge \neg b)] \wedge \\ [(q=1) \wedge (\mathsf{X}q = \neg\ell \wedge q \wedge \neg b \vee \ell \wedge p \wedge b)] \wedge \\ [\mathsf{GF}p] \end{array}\right)$
- $\mathsf{G}(\ell = \mathsf{G}x) := \left(\begin{array}{l} \exists pq. \\ [(p=1) \wedge (\mathsf{X}p = \ell \wedge p \wedge x \vee \neg\ell \wedge q \wedge \neg x)] \wedge \\ [(q=1) \wedge (\mathsf{X}q = q = \neg\ell \wedge q)] \wedge \\ [\mathsf{GF}p] \end{array}\right)$
- $\mathsf{G}(\ell = \mathsf{F}x) := \left(\begin{array}{l} \exists pq. \\ [(p=1) \wedge (\mathsf{X}p = \neg\ell \wedge p \wedge \neg x \vee \ell \wedge q \wedge x)] \wedge \\ [(q=1) \wedge (\mathsf{X}q = \ell \wedge q)] \wedge \\ [\mathsf{GF}p] \end{array}\right)$
- $\mathsf{G}(\ell = \mathsf{X}x) := \left(\begin{array}{l} \exists pq. \\ [(p=0) \wedge (\mathsf{X}p = \ell \vee x \wedge q \vee \neg x \wedge p)] \wedge \\ [(q=0) \wedge (\mathsf{X}q = \neg\ell \vee x \wedge q \vee \neg x \wedge p)] \wedge \\ [\mathsf{GF}p = q] \end{array}\right)$

Kapitel 5

Experimentelle Ergebnisse

5.1 Der CTL-Modellprüfer SMV

Nach der Reduktion auf CTL muß ein CTL-Modellprüfer verwendet werden. Dazu wird in dieser Arbeit das bereits existierende Modellprüfungssystem SMV [McMi92b] benutzt. SMV wurde an der Carnegie-Mellon Universität in Pittsburgh implementiert und dient zur Modellprüfung von CTL-Formeln unter mehreren Fairneßrestriktionen der Form $\mathsf{GF}\varphi$ mit CTL-Formeln φ. SMV arbeitet durch die symbolische Modellprüfung sehr effizient.
Weitere Eigenschaften von SMV sind die dynamische Variablenordnung und modulare Beschreibung des Modells. Man kann bei der Verifikation komplexer Systeme die Beschreibung in kleineren Modulen zerlegen und sie nachher entsprechend instantiieren und mehrfach verwenden. Bei der dynamischen Variablenordnung wird zur Reduktion der BDD-Größe eine neue Ordnung berechnet.
Als Beispiel für eine SMV-Datei betrachte man das folgende:

```
MODULE main

VAR
      q0  : boolean;
      r   : boolean;
      e   : boolean;
      q1  : boolean;
      out : boolean;

ASSIGN
      init(q0)  := 0;
      next(q0)  := !r & ((e & q1) | ((!e) & q0));
      init(q1)  := 0;
      next(q1)  := !r & ((e & (!q0 & !q1)) | (!e & q1));

SPEC
      (out = (e & q0 & !q1)) →
          AG (r & AX AG !r →
                 (AX AF (e & AX AF (e & AX AF (e & out)))))
```

Dieses Beispiel ergibt sich aus der Verifikation der Schaltung REPEAT3 vom nächsten Abschnitt mit einer CTL-Spezifikation.

5.2 Verifikationsbeispiele

In diesem Abschnitt soll die Implementierung anhand kleinerer Schaltungen getestet werden. Dabei wird im ersten Beispiel zwei Spezifikationen angegeben, eine in CTL⋆ und die andere in CTL. Das zweite Beispiel veranschaulicht die einzelnen Schritte bei der Berechnung.

5.2.1 Identifikationsbaustein

Die zu verifizierende Schaltung stellt hier eine Aktivierungseinheit dar, welche die Aufgabe hat, eine andere, hier nicht erläuterte Schaltung nach dreimalige Erkennung der entsprechenden Adresse zu aktivieren. Die Adresse wird dabei seriell empfangen und zwischen den einzelnen Identifikationen der Adresse darf eine beliebig lange Zeitspanne vergehen. Das Ausgangssignal der Adreßerkennungseinheit wird in die Komponente REPEAT3 eingespeist. REPEAT3 hat die Aufgabe, die erfolgreichen Identifikationen zu zählen und damit das Aktivierungssignal zu erzeugen. Die Implementierung von REPEAT3 erfolgt wie in Abbildung 5.1 gezeigt.
Eine mögliche Spezifikation in CTL⋆ ist die folgende:

$$\text{Spec}_{CTL\star} := \mathsf{A}[(\mathsf{X}[(\mathsf{X}[(\mathsf{X}[out\ \mathsf{W}\ e])\ \mathsf{W}\ e])\mathsf{W}\ e])\mathsf{W}\ (\mathsf{A}[(\mathsf{XG}\neg r)\ \mathsf{W}\ r])]$$

Diese Formel ist folgendermaßen zu lesen: erfolgt zu einem beliebigen Zeitpunkt ein Rücksetzen der Schaltung und erfolgt danach kein Rücksetzen mehr, so gilt *out*, falls der e-Eingang dreimal auf hohem Potential war.
Im Vergleich zu $\text{Spec}_{CTL\star}$ ist die CTL-Spezifikation komplizierter und unübersichtlicher. Dabei gilt auch allgemein, daß die CTL⋆-Spezifikationen kürzer sind. Eine mögliche CTL-Spezifikation sieht folgendermaßen aus:

$$\text{Spec}_{CTL} := (\mathsf{AG}\ ((r \wedge \mathsf{AX\ AG}\neg r) \rightarrow (\mathsf{AX\ AF}(e \wedge \mathsf{AX\ AF}\ (e \wedge \mathsf{AX\ AF}(e\ \wedge\ out))))))$$

Abbildung 5.1: REPEAT3

Zur Verifikation dieser Schaltung benötigt man eine Kripke-Struktur oder ein Zustandsübergangssystem, das diese Struktur darstellt. Dabei sieht das Übergangssystem für REPEAT3 wie folgt aus:

$$((q_0 = 0) \wedge \mathsf{G}\ (\mathsf{X}\ q_0 = \neg r\ \wedge\ ((e\ \wedge\ q_1)\ \vee\ (\neg e\ \wedge\ q_0)))) \wedge$$
$$((q_1 = 0) \wedge\ \mathsf{G}\ (\mathsf{X}\ q_1 = \neg r\ \wedge\ ((e\ \wedge\ \neg q_0\ \wedge\ \neg q_1)\ \vee\ (\neg e\ \wedge\ q_1))))$$

Nach der Beweisdurchführung mit dem SMV-system erhält man folgende Ergebnisse:

	Operatoren	BDD's	Übergangssystem	Zustände	erreichbar
CTL⋆	12	138	19 BDD	32	24
CTL	18	130	19 BDD	32	24

Von den 32 theoretisch erreichbaren Zuständen sind nur 24 erreichbar. Das kommt zustande, weil der Zustand $q_1 q_0 = 11$ nie erreicht wird. Das SMV-System benötigte 138 BDD-Knoten bei CTL⋆- und 130 bei CTL-Spezifikationen für die Beweisdurchführung. Für die Darstellung der Kripke-Struktur wurden 19 BDD-Knoten benötigt.

Die Zeit, die das System für die beiden Beweise benötigt hat, sind gleich. Daher hat man einen Vorteil bei CTL⋆-Spezifikationen gegenüber CTL-Spezifikationen, daß sie kürzer sind. So hat $\text{Spec}_{CTL\star}$ 12 Operatoren und Spec_{CTL} 18 Operatoren.

5.3 Verifikation eines Einzelimpulsbausteins

Die Verifikation und Spezifikation eines Einzelimpulsebausteines dient in der Literatur häufig als Benchmark für verschiedene Verifikationssysteme [JoMC94]. Die Schaltung hat einen Eingang φ und einen Ausgang ψ und soll die folgende informelle Spezifkation erfüllen: innerhalb jedes Zeitraums, indem die Eingabe durchweg auf hohem Potential liegt, soll der Ausgang für exakt eine Taktperode lang auf 1 liegen. Der Einzelimpuls am Ausgang muß dabei auf jeden Fall vor dem nächsten 1-0 Übergang ausgegeben werden, kann aber zu einem beliebigen Zeitpunkt erfolgen.
Eine mögliche formale Spezifikation mit temporalen Operatoren lautet folgendermaßen:

$$\mathsf{EG}\left(\begin{array}{l}[\neg\varphi \to \mathsf{X}(\varphi \to (\varphi\ \mathsf{U}\ (\varphi \wedge \psi)))] \wedge \\ [\psi \to \mathsf{X}(\neg\psi\ \mathsf{U}\ \neg\varphi)] \wedge \\ [\neg\varphi \to \neg\psi]\end{array}\right)$$

Im folgende werden die einzelnen Schritte bei der Übersetzung dieser Formel in CTL beschrieben.
Nach der Bildung der Negationsnormalform erhält man die folgende äquivalente Formel:

$$\mathsf{EG}\left(\begin{array}{l}[\varphi \vee \mathsf{X}((\neg\varphi) \vee (\varphi\ \mathsf{U}\ (\varphi \wedge \psi)))] \wedge \\ [(\neg\psi \vee \mathsf{X}(\neg\psi\ \mathsf{U}\ \neg\varphi)] \wedge \\ [\varphi \vee \neg\psi]\end{array}\right)$$

Nachdem man die erste Teilformel mit dem Pfadquantor E ergänzt hat, folgt dieses E auf eine Konjunktion. Daher muß diese Teilformel in eine Disjunktion umgewandelt werden.

$$\mathsf{E}\left(\begin{array}{l}[\varphi \wedge ((\varphi \vee \neg\psi) \wedge (\neg\psi \vee \mathsf{X}(\neg\psi\ \mathsf{U}\ \neg\varphi)))] \vee \\ \left[\begin{array}{l}(\varphi \vee \neg\psi) \wedge \\ (\neg\psi \wedge \mathsf{X}(\neg\varphi \vee (\varphi\ \mathsf{U}\ (\varphi \wedge \psi))) \vee \\ \mathsf{X}((\neg\varphi \vee (\varphi\ \mathsf{U}\ (\varphi \wedge \psi))) \wedge (\neg\psi\ \mathsf{U}\ \neg\varphi)))\end{array}\right]\end{array}\right)$$

Bei diesem Vorgang wurden nur boolesche Transformationen und die Gleichung $\mathsf{X}\phi \wedge \mathsf{X}\psi = \mathsf{X}(\phi \wedge \psi)$ verwendet.
Nach der Ergänzung von Pfadquantoren folgt diesmal ein E auf die folgende Formel:

$$\left(\begin{array}{l}(\neg\varphi \vee (\varphi\ \mathsf{U}\ (\varphi \wedge \psi))) \wedge \\ (\neg\psi\ \mathsf{U}\ \neg\varphi)\end{array}\right)$$

Dabei wird die Formel ausmultipliziert und den Satz 4.1.2 darauf angewendet, so daß man die folgende Formel erhält:

$$\mathsf{E}\left(\begin{array}{l}[\neg\varphi \wedge (\neg\psi\ \mathsf{U}\ \neg\varphi)] \vee \\ [(\varphi \wedge (\neg(\varphi \wedge \psi)\ \mathsf{W}\ (\neg\varphi \vee (\varphi \wedge \psi)))) \ \mathsf{W}\ \psi] \vee \\ [(\neg(\varphi \wedge \psi) \wedge (\varphi\ \mathsf{W}\ (\psi \vee \neg\varphi)))\ \mathsf{W}\ \psi]\end{array}\right)$$

Und zum Schluß wird die Formel *bottom-up* wieder rekonstruiert, und man erhält eine äquivalente CTL-Formel.

$$\mathsf{EG}\left(\begin{array}{l} (\varphi \wedge ((\varphi \vee \neg\psi) \wedge (\neg\psi \vee \mathsf{EX}\ \mathsf{E}[\neg\psi\ \mathsf{U}\ \neg\varphi])))\vee \\ \left(\begin{array}{l} (\varphi \vee \neg\psi)\wedge \\ \left(\begin{array}{l} \neg\psi \wedge \mathsf{EX}(\neg\varphi \vee \mathsf{E}[\varphi\ \mathsf{U}\ (\varphi \wedge \psi)])\vee \\ \mathsf{EX}\left(\begin{array}{l} (\neg\varphi \wedge \mathsf{E}[\neg\psi\ \mathsf{U}\ \neg\varphi])\vee \\ \mathsf{E}[(\varphi \wedge \mathsf{E}[(\neg(\varphi \wedge \psi))\ \mathsf{W}\ (\neg\varphi \vee (\varphi \wedge \psi))]))\ \mathsf{W}\ \psi]\vee \\ \mathsf{E}[(\neg(\varphi \wedge \psi) \wedge \mathsf{E}[\varphi\ \mathsf{W}\ (\psi \vee \neg\varphi)])\ \mathsf{W}\ \psi] \end{array}\right) \end{array}\right) \end{array}\right) \end{array}\right)$$

Man sieht auch anhand dieses Beispiels, daß die CTL$\star$-Spezifikation viel kürzer als eine mögliche CTL-Spezifikation sein kann.

Literaturverzeichnis

[Aker78] S.B. Akers. Binary Decision Diagrams. *IEEE Transactions on Computers*, C-27(6), June 1978.

[AlCD90] R. Alur, C. Courcoubetics, and D.L. Dill. Model Checking for Real-Time Systems. In *Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 414–425, Washington, D.C., June 1990. IEEE Computer Society Press.

[AlHe91] R. Alur and T.A. Henzinger. Logics and Models of Real-Time: A Survey. In *Real Time: Theory in Practice*, volume 600 of *Lecture Notes in Computer Science*, pages 74–106. Springer-Verlag, 1991.

[BaVe86] S. Bapat and G. Venkatesh. Reasoning about digitalsystems using temporal logic. In *23rd ACM/IEEE Design Automation Conference (DAC86)*, pages 215–219, Las Vegas, USA, June 1986.

[BCLM93] J.R. Burch, E.M. Clarke, D.E. Long, K.L. McMillan, and D. Dill. Symbolic Model Checking for Sequential Circuit Verification. Technical Report CMU-CS-93-211, Carnegy Mellon University, Pittsburg, PA 15213, July 1993.

[BCLM94] J.R. Burch, E.M. Clarke, D.E. Long, K.L. MacMillan, and D.L. Dill. Symbolic model checking for sequential circuit verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(4):401–424, April 1994.

[BCMD90] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. In *Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 1–33, Washington, D.C., June 1990. IEEE Computer Society Press.

[BHMS86] R.K. Brayton, G.D. Hachtel, C.T. McMullen, and A.L. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*. The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, 1986.

[BlBu87] K.H. Bläsius and H.-J. Bürckert. *Deduktionssysteme*. Oldenburg Verlag, 1987.

[BrCD85] M.C. Browne, E.M. Clarke, and D.L. Dill. Checking the correctness of sequential circuits. In *Proceedings of the IEEE's International Conference on Computer Design*, pages 445–448, 1985.

[Brow86] M.C. Browne. An Improved Algorithm for Automatic Verification of Finite-State Machines Using Temporal Logic. In *Proceedings of Conference on Logic in Computer Science*, pages 260–266, Boston, Massachusetts, June 1986. IEEE Computer Society Press.

[Brya86] R.E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[CaCl94] S.V. Campos and E. Clarke. Real-Time Symbolic Model Checking for Discrete Time Models. In T. Rus and C. Rattray, editors, *Theories and Experiences for Real-Time System Develpment*, AMAST Series in Computing. World Scientific Press, AMAST Series in Computing, May 1994.

[CaPr88] P. Camurati and P. Prinetto. Formal verification of hardware correctness: Introduction and survey of current research. *IEEE Computer*, 21(7):8–19, July 1988.

[Chur36] A. Church. A note on the Entscheidungsproblem. *Journal of Symbolic Computation*, 1, 1936.

[ClEm81a] E.M. Clarke and E.A. Emerson. Characterizing Properties of Parallel Programs as Fixpoints. In *7th International Colloquium on Automata, Languages and Programming*, volume 85 of *Lecture Notes in Computer Science*. Springer-Verlag, 1981.

[ClEm81b] E.M. Clarke and E.A. Emerson. Synthesis of Synchronization Skeletons for Branching Time Temporal Logic. In *Logics of Programs: Workshop*, volume 131 of *Lecture Notes in Computer Science*, Yorktown Heights, New York, May 1981. Springer-Verlag.

[ClES83] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic. In *Proceedings of the tenth Annual ACM Symposium on Principles of Programming Languages*, 1983.

[ClES86] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, April 1986.

[ClGB86] E.M. Clarke, O. Grumberg, and M.C. Browne. Reasonig about networks with many identical finite-state processes. In *Proceedings of the Fifth Annual ACM Symposium on Principles of Distributed Computing*, pages 240–248, New York, August 1986. ACM.

[ClGH94] E.M. Clarke, O. Grumberg, and K. Hamaguchi. Another look at LTL model checking. In D. Gabbay and H.J. Ohlbach, editors, *Temporal Logic*, Lecture Notes in Artificial Intelligence. Springer-Verlag, July 1994.

[ClGL93] E.Clarke, O. Grumberg, and D. Long. Verification Tools for Finite State Concurrent Systems. In J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *A Decade of Concurrency-Reflections and Perspectives*, volume 803 of *Lecture Notes in Computer Science*, pages 124–175, Noordwijkerhout, Netherlands, June 1993. REX School/Symposium, Springer-Verlag.

[ClLM89b] E.M. Clarke, D.E. Long, and K.L. McMillan. A Language for Compositional Specification and Verification of Finite State Hardware Controllers. In J.A. Darringer and F.J. Rammig, editors, *International Symposioum on Computer Hardware Description Languages and their Applications*, pages 281–295, Amsterdam, 1989. IFIP North-Holland.

[CoMa91] O. Coudert and J.C. Madre. Symbolic computation of the valid states of a sequential machine: algorithms and discussion. In *ACM Workshop on Formal Methods in VLSI Design*, 1991. Miami.

[CoMB91] O. Coudert, J.C. Madre, and C. Berthet. Verifying temporal properties of sequential machines without building their state diagrams. In E.M. Clarke and R.P. Kurshan, editors, *Proceedings of the Workshop on Computer-Aided Verification (CAV90)*, volume 3 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, New York, 1991. American Mathematical Society, Springer-Verlag.

[Emer90] E.A. Emerson. Temporal and Modal Logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 996–1072, Amsterdam, 1990. Elsevier Science Publishers.

[EmLe85] E.A. Emerson and C.-L. Lei. Modalities for model checking: Branching time strikes back. In *Proceedings of the Twelfth Annual ACM Symposium on Principles of Programming Languages*, pages 84–96, New York, January 1985. ACM.

[EMSS92a] E.A. Emerson, A.K. Mok, A.P. Sistla, and J. Srinivasan. Quantitative Temporal Reasoning. *Journal of Real Time Systems*, 4:331–352, 1992.

[Evek91b] H. Eveking. *Verifikation digitaler Systeme.* Leitfäden und Mongraphien der Informatik. B.G. Teubner Verlag, Stuttgart, 1991.

[FrSu90] S.J. Friedman and K.J. Supowit. Finding the optimal variable ordering for binary decision diagrams. In *IEEE Transaction on Computers*, volume 39, pages 710–713, 1990.

[FuFu89] M. Fujita and H. Fujisawa. Specification, verification and synthesis of control circuits with propositional temporal logic. In J.A. Darringer and F.J. Rammig, editors, *Proceedings of the Ninth International Symposium on Computer Hardware Description Languages and their Applications*, pages 265–279, Washington, June 1989. IFIP WG 10.2, North-Holland.

[Gupt92] A. Gupta. Formal hardware verification methods: A survey. *Journal of Formal Methods in System Design*, 1:151–238, 1992.

[HuGr68] G.E. Hughes and M.J. Cresswell. *An Introduction to Modal Logics*. Methuen and Co. Ltd, 1968.

[JoMC94] S.D. Johnson, P.S. Miner, and A. Camilleri. Studies of the single pulser in various reasoning systems. In T. Kropf and R. Kumar, editors, *Proc. 2nd International Conference on Theorem Provers in Circuit Design (TPCD94)*, volume 901 of *Lecture Notes in Computer Science*, pages 126–145, Bad Herrenalb, Germany, September 1994. Springer-Verlag. published 1995.

[Keut91] K. Keutzer. The need for formal verification in hardware design and what formal verification has NOT done for me lately. In M. Archer, J.J. Joyce, K.N. Levitt, and P.J. Windley, editors, *International Workshop on Higher Order Logic Theorem Proving and its Applications*, pages 77–86, Davis, California, August 1991. IEEE Computer Society, ACM SIGDA, IEEE Computer Society Press.

[Koha70] Z. Kohavi. *Switching and Finite Automata Theory*. McGraw-Hill, New York, 1970.

[KuSK93a] R. Kumar, K. Schneider, and T. Kropf. Structuring and Automating Hardware Proofs in a Higher-Order Theorem-Proving Environment. *International Journal of Formal System Design*, pages 165–230, 1993.

[LiPn85] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proceedings of the Twelfth Annual ACM Symposium on Principles of Programming Languages*, pages 97–107, New York, January 1985. ACM.

[MaBi88] J.C. Madre and J.P. Billon. Proving circuit correctness using formal comparison between expected and extracted behavior. In *Proceedings of the 25th ACM/IEEE Design Automation Conference*, pages 205–210, Los Alamitos, CA, 1988. IEEE Computer Society Press.

[MaPn92] Z. Manna and A. Pnueli. *The temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, New York, Berlin, Heidelberg, 1992.

[McMi92a] K.L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem.* PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, May 1992. CMU-CS-92-131.

[McMi92b] K.L. McMillan. The SMV system, symbolic model checking - an approach. Technical Report CMU-CS-92-131, Carnegie Mellon University, 1992.

[McMi93a] K.L. McMillan. *Symbolic Model Checking.* Kluwer Academic Publishers, Norwell Massachusetts, 1993.

[MiIY90a] S. Minato, N. Ishiura, and S. Yajima. Shared Binary Decision Diagram with Attributed Edges for Efficient Boolean Function Manipulation. In *Proceedings of the 27th ACM/IEEE Design Automation Conference*, pages 52–57, Los Alamitos, CA, June 1990. ACM/IEEE, IEEE Society Press.

[More82] B.M.E. Moret. Decision Trees and Diagrams. *ACM Computing Surveys*, pages 593–623, 1982.

[MWBS88] S. Malik, A.R. Wang, R.K. Brayton, and A. Sangiovanni-Vincentelli. Logic Verification Using Binary Decision Diagrams in a Logic Synthesis Environment. In *International Conference on Computer-Aided Design*, pages 6–9. IEEE, November 1988.

[Pnue77] A. Pnueli. The temporal logic of programs. In *Proceedings of the Eighth Annual Symposium on Foundations of Computer Science*, volume 18, pages 46–57, New York, 1977. IEEE.

[Schn96b] K. Schneider. *Ein einheitlicher Ansatz zur Unterstützung von Abstraktionsmechanismen der Hardwareverifikation*, volume 116 of *DISKI (Dissertationen zur Künstlichen Intelligenz).* Infix Verlag, Sankt Augustin, 1996. ISBN 3-89601-116-2.

[Scho87] U. Schöning. *Logik für Informatiker*, volume 56 of *Reihe Informatik.* BI Wissenschaftsverlag, Universität Ulm, 3 edition, 1987.

[Shan48] C.E. Shannon. The synthesis of two-terminal switching circuits. *Bell Systems Technical Journal*, 1948.

[SuFr86] K.J. Supowit and S.J. Friedman. A New Method for Verifying Sequential Circuits. In *23rd ACM/IEEE Design Automation Conference*, pages 200–207. ACM/IEEE, IEEE, 1986.

[Tars55] A. Tarski. A Lattice-Theoretical Fixpoint Theorem and its Applications. *Pacific J. Math*, 5:285–309, 1955.

[VeLe93] B. Vergauwen and J. Lewi. A Linear Local Model Checking Algorithm for CTL. In E. Best, editor, *4th International Conference on Concurrency Theorie (CONCUR'93)*, volume 715 of *Lecture Notes in Computer Science*, pages 447–461, Hildesheim, Germany, August 1993. Springer-Verlag.

[WiPn89] A. Wilk and A. Pnueli. Specification and Verification of VLSI Systems. In *Proceedings of the International Conference on Computer Aided Design*, pages 460–463, Rehovot, Israel, November 1989. Department of Computer Science, The Weizmann Institute of Science.

[Wund91] H.-J. Wunderlich. *Hochintegrierte Schaltungen: Prüfgerechter Entwurf und Test*. Springer-Verlag, 1991.

[Yoel90] M. Yoeli. Formal verification of hardware design. Technical report, IEEE Computer Society Press, Los Alamitos, 1990.