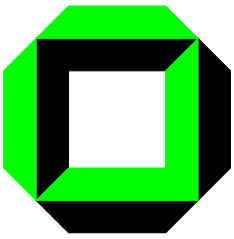

Universität Karlsruhe
Institut für Rechnerentwurf
und Fehlertoleranz
Prof. Dr.-Ing. D. Schmid

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

Diplomarbeit

Verifikation von Zeitbedingungen mittels arithmetischer Entscheidungsverfahren

von

Holger Weindel

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

Tag der Anmeldung : 01.09.1996
Tag der Abgabe: 28.02.1997

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 28.Februar 1997

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Integrierte Schaltungen, speziell digitale Schaltungen, spielen in der Industrie und im öffentlichen Leben eine immer wichtigere Rolle. Gerade in sicherheitskritischen Bereichen, wie z.B. in Kernkraftwerken, Flugleitsystemen und in der medizinischen Technik ist es zwingend erforderlich, daß die eingesetzten digitalen Schaltungen fehlerfrei arbeiten. Die Entwicklung solcher sicherheitskritischer Schaltungen muß folglich die Verifikation derselben einbeziehen, das heißt es muß überprüft werden, ob die Schaltungen ihre Spezifikation, also die formale Beschreibung ihrer Aufgabe erfüllt.

Oft werden zur Spezifikation Impulsdiagramme verwendet, welche den zeitlichen Verlauf der Eingangs- und Ausgangssignale der digitalen Schaltung definieren. Unter einer Verifikation dieser Impulsdiagramme versteht man nun die Überprüfung der Ausgangssignale unter Berücksichtigung der Eingangssignale und der Schaltungsstruktur. Diese Überprüfung kann sich auf funktionales oder auf temporales Verhalten der digitalen Schaltung beziehen.

Eine Möglichkeit, eine Schaltung zu überprüfen ist die Durchführung einer vollständigen Simulation, wobei für sämtliche Belegungen von Eingabevariablen in jedem erreichbaren Zustand die Ausgabe der Schaltung auf Korrektheit überprüft werden muß. Die hochgradige Komplexität heutiger Schaltungen hat jedoch zur Folge, daß die Überprüfung durch eine vollständige Simulation mit enormem Aufwand verbunden ist: die Anzahl der möglichen Belegungen der Variablen hängt exponentiell von ihrer Anzahl ab. Eine vollständige Simulation ist somit in den allermeisten Fällen unmöglich. Bei zwei 32-Bit breiten Eingaben wären 2^{64} Eingaben zu prüfen. Sind 32 Zustandsvariablen (1 Register) vorhanden, so entstehen 2^{96} Testmuster. Wenn man jedes in 1ns prüfen könnte, dauert die Simulation $2,5 * 10^{12}$ Jahre. Da selbst kleine digitale Schaltungen beispielsweise durch die Verwendung von Registern mehr als 100 Zustandsvariablen aufweisen, wird die Verifikation einer digitalen Schaltung durch Testen der einzelnen Pfade

zu einer zeitaufwendigen Komponente bei deren Entwicklung.

Auch marktwirtschaftlich betrachtet ist die in die Entwicklung einer Schaltung und damit auch die in deren Verifikation investierte Zeit von großer Bedeutung. Zu spät auf den Markt gebrachte Schaltungen decken häufig nicht einmal deren Entwicklungskosten und können nicht mehr gewinnbringend vermarktet werden. Folglich steigt mit wachsendem Konkurrenzkampf die Notwendigkeit, die Verifikation mit leistungsstarken Werkzeugen automatisch und schnell durchführen zu können.

In diesem Zusammenhang ist die sogenannte *formale Verifikation* von besonderem Interesse. Unter der formalen Verifikation einer digitalen Schaltung versteht man den mathematischen Nachweis, daß die Schaltung ihre Spezifikation erfüllt. Um eine formale Verifikation durchführen zu können, werden zunächst Formalismen benötigt, mit deren Hilfe digitale Schaltungen beschrieben werden können. Diese Formalismen müssen zum einen ausdrucksstark genug sein, um auch komplexe Schaltungen beschreiben zu können. Zum anderen müssen die Formalismen derart gestaltet sein, daß möglichst schnell entscheidbar ist, ob die Schaltung die geforderten Eigenschaften erfüllt.

Es wurden bereits viele Formalismen entwickelt, die nach unterschiedlichen Merkmalen unterteilt werden können. Zum einen gibt es entscheidbare und nicht entscheidbare Formalismen. Ein zweites Unterscheidungsmerkmal ist das Art der Verifikation. Einige Formalismen eignen sich besser zur Verifikation temporaler Eigenschaften, andere wiederum zur Verifikation funktionaler Eigenschaften. In dieser Ausarbeitung soll vornehmlich ein Formalismus betrachtet werden, mit dessen Hilfe temporale Eigenschaften verifizieren werden können. Bei diesem Formalismus handelt es sich um die monadische Arithmetik zweiter Ordnung mit einem Sukzessor (S1S).

Die Tabelle 1.1 zeigt bekannte Formalismen, die sich zur Verifikation temporaler und funktionaler Eigenschaften eignen.

Formale Definitionen und detailierte Beschreibungen der in obiger Tabelle genannten Formalismen findet man u.a. in [1, 2, 25].

Einige Formalismen eignen sich sowohl zur Verifikation temporaler Eigenschaften, als auch zur Verifikation funktionaler Eigenschaften. Zu diesen Formalismen gehört auch die Arithmetik S1S. So wurde S1S bereits für die automatische Verifikation von rekursiv definierten Strukturen benutzt [8]. Aber auch zur Überprüfung von Datenpfaden kann S1S hinzugezogen werden, wie das folgende Beispiel zeigt:

Beispiel:

Bild 1.1 zeigt einen seriellen Addierer. Die Addition erfolgt durch Abarbeitung einer 'WHILE'-Schleife. Die erste Zahl, die sich im Register R_1 befindet, wird in jedem Schleifendurchgang um eins erhöht und die zweite Zahl, die sich im Register R_2 befindet, wird in jedem Schleifendurchgang um eins verkleinert. Wenn die Zahl aus Register R_2 Null erreicht hat, ist

Formalismus	entscheidbar	Spezif. temporaler Eigenschaften	Spezif. funktionaler Eigenschaften
Temporale Logiken: - LTL, CTL, CTL* - ITL	 ja nein	 gut gut	 ungeeignet ungeeignet
Sprachen (ω-regulär)	ja	geeignet	ungeeignet
Arithmetiken: - Peano - Skolem, Pressburger - S1S	 nein ja ja	 gut geeignet gut	 geeignet geeignet geeignet
aussagenlog. μ-Kalkül	ja	gut	ungeeignet
Boyer-Moore Logik	nein	ungeeignet	gut
Prädikatenlogik höherer Ordnung	nein	gut	gut
Formale Synthese	nein	ungeeignet	geeignet

Tabelle 1.1: Gebräuchliche Formalismen

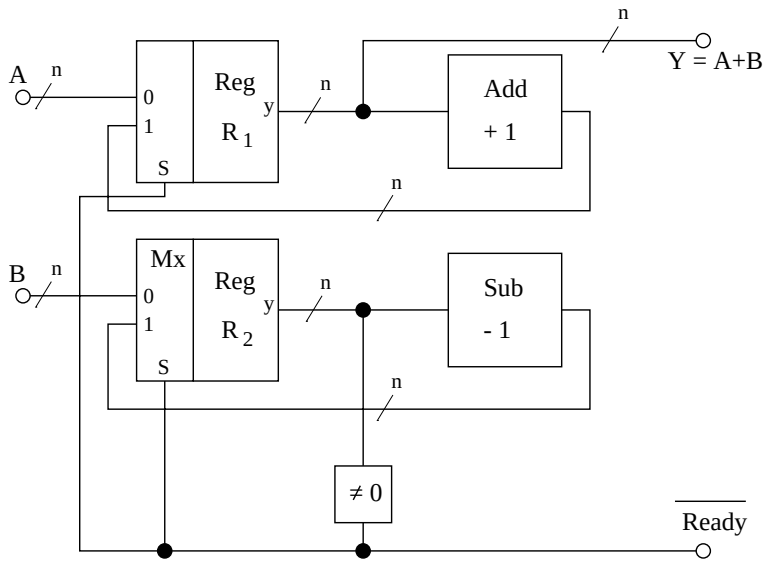

Abbildung 1.1: Beispiel: Serieller Addierer

die Operation abgeschlossen. Registerüberläufe werden in diesem Beispiel nicht beachtet.

Es ergibt sich folgende Schleifeninvariante: $R_1 + R_2 = A + B$. Nach dem ersten Durchgang durch die Schleife muß demnach:

$$(R_1 + 1) + (R_2 - 1) = A + B$$

gelten. Die Aufgabe der Arithmetik ist nun zu zeigen, daß

$$(R_1 + 1) + (R_2 - 1) = R_1 + R_2$$

ist. Mit S1S ist man in der Lage diese Gleichung zu spezifizieren.

Thema dieser Arbeit ist es demnach mit Hilfe einer formalen Verifikation digitale Schaltungen zu untersuchen und zu entscheiden, ob diese ihre Spezifikation erfüllen. Die Arbeit kann folgendermaßen gegliedert werden:

1. Büchis theoretische Arbeit effizient zu implementieren
2. Einbettung von LTL (lineare temporale Logik)

1.2 Formalismen zur Verifikation zeitlicher Eigenschaften

Die bereits bestehenden Ansätze zur formalen Verifikation zeitlicher Eigenschaften bedienen sich u.a. der folgenden Formalismen:

Schon die **Aussagenlogik** ist ein erster Formalismus zur Verifikation. Da in der Aussagenlogik kein Zeitverhalten modelliert werden kann, eignet sie sich nur zur Beschreibung von logischen Funktionen und ist demnach für die Verifikation von Schaltwerken ungeeignet. Trotzdem findet sie häufige Anwendung, da sie Grundlage anderer Formalismen ist. Desweiteren kann sie relativ effizient mit Hilfe von BDDs entschieden werden. Der theoretische Aufwand zur Entscheidung der Aussagenlogik ist NP-vollständig und das 'Modell-Checking' hat einen linearen Aufwand.

Die **temporale Aussagenlogik** wurde zusätzlich ausgehend von der Aussagenlogik um ein Zeitmodell ergänzt. Der Wahrheitswert der einzelnen Aussagevariablen ist hier im Gegensatz zur Aussagelogik nicht global gültig, sondern jeweils einem festen Zeitpunkt zugeordnet. Ein Zeitmodell kann durch eine Kripkestruktur definiert werden. Das ist ein endliches Zustandsübergangssystem, dessen Zustände verkettet sind. Alle Zustände sind an eine Menge von Aussagevariablen gebunden, die in dem jeweiligen Zuständen gelten. Ein Pfad durch dieses Zustandsübergangssystem ist demnach eine Folge von Zustandsbelegungen der Aussagevariablen.

Beispiel:

Gegeben sei eine Kripke-Struktur mit zwei Zuständen a und b und das in Bild 1.2 angegebene Zustandsübergangssystem:

Abbildung 1.2: Beispiel: Kripkestruktur mit zwei Zuständen

Dann gilt entweder $a \wedge \neg b$ oder $\neg a \wedge b$. Weiter gilt:

$$\begin{aligned} &\models a \\ &\models \mathsf{X}b \\ &\models \mathsf{F}b \\ &\models \mathsf{GX}b \end{aligned}$$

Man unterscheidet generell drei Klassen von Operatoren:

1. boolesche Operatoren: $\neg,\ \wedge,\ \vee,\ \rightarrow,\ \ldots$
2. temporale Operatoren: $\mathsf{G}, \mathsf{F}, \mathsf{X}, \mathsf{U}, \mathsf{W}, \mathsf{B}$
3. Pfadquantoren: $\forall\ \exists$

Die Bedeutung der booleschen Operatoren entspricht denen der Aussagenlogik. Die temporalen Operatoren haben folgende Bedeutung [12]:

[$\mathsf{X}\varphi$]: Der '*NEXT*'–Operator X gibt an, daß φ zum nächsten Zeitpunkt betrachtet wird.

$\mathsf{G}\varphi$: Der '*ALWAYS*'–Operator G gibt an, daß die Formel φ von nun an immer gilt.

[$\mathsf{F}\varphi$]: Der '*EVENTUAL*'–Operator F gibt an, daß die Formel φ in einem zukünftigen Zeitpunkt mindestens einmal gültig wird.

[$\varphi\ \mathsf{U}\ \psi$]: Die Formel φ gilt bis die Formel ψ gilt.

[$\varphi\ \mathsf{W}\ \psi$]: Die Formel φ gilt nachdem die Formel ψ zum ersten mal wahr wurde.

[$\varphi\ \mathsf{B}\ \psi$]: Die Formel φ wird vor der Formel ψ zum Nächsten mal wahr.

Die drei Operatoren [U], [W] und [B] sind ineinander überführbar [20].

Es gibt die beiden Pfadquantoren **A** und **E**. Pfadquantoren wandeln Pfadformeln, also Formeln, die sich auf einen Pfad beziehen, in Zustandsformeln um. Zustandsformeln beziehen sich folglich nicht mehr auf eine Pfad sondern auf einen bestimmten Zustand. Demnach gilt:

- **A**. φ gilt in einem Zustand S, gdw. auf allen von S ausgehenden Pfaden φ gilt.
- **E**. φ gilt in einem Zustand S, gdw. auf einem von S ausgehenden Pfaden φ gilt.

Alle oben aufgezählten Operatoren können beliebig ineinander geschachtelt werden.

Beispiel:

Seien E_1, E_2 und A Variablen, dann ist

$$\exists A.\mathsf{G}[\mathsf{X}\,[(\neg A)\ \mathsf{U}\ E_1 \vee E_2]\ \mathsf{W}\ E_1]$$

eine Formel der temporalen Aussagenlogik.

Allgemein unterscheiden sich die verschiedenen temporalen Aussagenlogiken in der Syntax. So wird zwischen linearer, verzweigender und intervallbasierter temporaler Aussagenlogik unterschieden.

- **Lineare temporale Logik (LTL):** Bei dieser temporalen Logik gibt es keine Pfadquantoren. Diese Logik besteht aus reinen Pfadformeln.
- **verzweigende temporale Logik: (CTL, CTL*)** Bei dieser Logik geht man davon aus, daß jeder Zeitpunkt mehrere Folgezeitpunkte besitzen kann, und der Folgezeitpunkt aus der Menge aller Zeitpunkte indeterministisch ermittelt wird. Es entsteht demnach eine Baumstruktur, bei der jeder Pfad wiederum eine LTL ist. Beispiele dieser Logiken sind CTL (Computation Tree Logic) oder CTL*. CTL wurde von Clarke und Emerson [14] entwickelt und hat einen Aufwand von $O(\|M\|, \|\varphi\|)$, wobei $\|M\|$ die Anzahl der Zustände + die Anzahl der Zustandsübergänge angibt und $\|\varphi\|$ die Länge der Formel ist. Nachteilig ist jedoch die geringe Ausdrucksstärke von CTL.

  CTL* ist eine Obermenge von CTL und LTL, hat jedoch einen schlechteren Aufwand als CTL.
- **Temporale Intervall-Logik(ITL):** Im Gegensatz zu den linearen und verzweigenden temporalen Aussagelogiken wird in der temporalen Intervall-Logik nicht mehr den einzelnen Zuständen jeweils eine aussagenlogische Belegung zugewiesen, stattdessen erhalten Intervalle von Zuständen eine aussagenlogische Belegung. Ein großer Nachteil von ITL ist die Tatsache, daß ITL nicht mehr entscheidbar ist [17]. Formalismen, die nicht entscheidbar sind, können in der automatischen Verifikation nicht eingesetzt werden

In Tabelle 1.2 ist der Aufwand für den Erfüllbarkeitstest und für das 'Model-Checking' der temporalen Logiken LTL CTL und CTL* aufgeführt. Eine ausführlichere Beschreibung besagter Logiken findet sich beispielsweise in [20].

	Erfüllbarkeitstest	'Model-Checking'
LTL	'PSPACE'-vollständig [3]	'PSPACE'-vollständig [3]
CTL	'EXPTIME'-vollständig [23]	linear [15]
CTL*	2×'EXPTIME'-vollständig [28]	'PSPACE'-vollständig [13]

Tabelle 1.2: Aufwand temporaler Logiken

Ein weiterer Ansatz zur formalen Verifikation liefern die **formalen Sprachen**. Jede digitale Schaltung hat eine Anzahl n von Eingaben und eine Anzahl m von Ausgaben. Betrachtet man nun diese Ein- und Ausgaben zu gewissen Zeitpunkten als zwei Tupel der Breite n und m, dann kann man den Eingabestrom $\vec{i}^0, \vec{i}^1, \ldots$ und den Ausgabestrom $\vec{o}^0, \vec{o}^1, \ldots$ der Schaltung als unendlich lange Wörter über diesen Tupeln auffassen. Die Schaltung selbst fungiert dann als Übersetzer, der den Eingabestrom in den Ausgabestrom übersetzt. Faßt man jetzt den Eingabestrom und den Ausgabestrom zu einem Gesamtstrom $(\vec{i}, \vec{o})$ zusammen, dann entsteht eine formale Sprache, die aus allen Wörtern besteht, die auf diese Art gebildet werden können. Da die Eingabe- und Ausgabefolge nicht beschränkt ist, es sich also um unendlich lange Wörter handelt, entstehen auf diese Art reguläre Sprachen über unendlich langen Wörtern [35]. Diese Sprache ist die Spezifikation der Schaltung. Jede Implementierung mit gleichem Ein- und Ausgabeverhalten führt zu derselben Sprache.

Ein weiterer Formalismus sind Arithmetiken. Die nun folgenden Arithmetiken sind Beispiele gebräuchlicher Arithmetiken.

- **Peano Arithmetik:** Die Peano Arithmetik [24] ist definiert durch die Peano-Axiome:

  (A1) $\forall a\, b.\, [S(a) = S(b)] \rightarrow [a = b]$

  (A2) $\forall a\, b.\, S(a) \neq 0$

  (A3) $\forall A.\, A(0) \wedge [\forall t. A(t) \rightarrow A(S(t))] \rightarrow [\forall t. A(t)]$

  Dabei ist

  A eine prädikatenlogische Variable,

  a, b, t numerische Variablen

  und S die Sukzessorfunktion.

  (A3) ist das benannte Induktionsgesetz und erfordert eine Quantifizierung über Prädikatsvariablen.

  Die Peano-Arithmetik ist zwar sehr mächtig, sie ist aber nicht entscheidbar [19] und somit nicht von direktem Nutzen.

- **Skolem Arithmetik, Pressburger Arithmetik:** Sowohl die Skolem Arithmetik [24], als auch die Pressburger Arithmetik [24] sind entscheidbare Untermengen der Peano Arithmetik. Der Nachteil bei diesen Arithmetiken liegt jedoch darin, daß nur Quantoren über der Zeit, nicht jedoch Quantoren über Signalen erlaubt sind. So ist $\forall x.\exists y.0 < x + y$ eine Formel der Pressburger Arithmetik, wobei mit x, y Zeitpunkte bezeichnet werden. Die nächste Formel kann mit der Skolem Arithmetik oder der Pressburger Arithmetik jedoch nicht ausgedrückt werden:

  $$\exists P.\exists t_1\ t_2.P(t_1) \wedge P(t_2) \wedge (t_1 < t_2)$$

  Hier stellt P ein Signal dar, das über einen Existenzquantor quantifiziert in die Formel eingeht.

  Dennoch kommt die Pressburger Arithmetik beispielsweise in Beweisern wie **HOL** [26] oder **PVS** [26, 32] zum Einsatz.

- **Monadische Arithmetik zweiter Ordnung mit einem Sukzessor:** In der monadischen Arithmetik zweiter Ordnung mit einem Sukzessor (S1S) [18, 9, 34] sind im Gegensatz zu der Skolem Arithmetik und zur Pressburger Arithmetik sowohl Quantoren über der Zeit als auch Quantoren über Signalen erlaubt. Insbesondere ist S1S genau so mächtig wie ω-Automaten und somit entscheidbar. Bereits in den sechziger Jahren hat Büchi ein Entscheidungsverfahren für S1S angegeben, welches auf der Überführung dieser Sprache in Büchi-Automaten beruht [18, 10]. Büchiautomaten werden in Kapitel 2.2.1 definiert.

  Vergleicht man S1S mit der Skolem Arithmetik oder der Pressburger Arithmetik, so stellt man fest, daß bis auf die Addition von zwei Zeitpunkten S1S eine Obermenge der beiden anderen Arithmetiken ist. Diese Addition von zwei Zeitpunkten, auf die in Kapitel 2.4 noch genauer eingegangen wird, spielt bei der Beschreibung von Impulsdiagrammen durch S1S jedoch nur eine untergeordnete Rolle. Deshalb ist S1S, beschränkt auf die Verifikation von Impulsdiagrammen, eine gemeinsame Obermenge der Skolem Arithmetik und der Pressburger Arithmetik.

  Büchi hat erstmals 1960 die Entscheidbarkeit von S1S gezeigt. Sein Entscheidungsverfahren, das in Form eines konstruktiven Beweises angegeben wird, ist weniger auf Effizienz ausgelegt, sondern widmet sich eher der Frage der Entscheidbarkeit von S1S. In Kapitel 3 wird das Büchische Entscheidungsverfahren im Detail behandelt.

  Nachteilig an Büchis Verfahren ist jedoch der nicht elementare Aufwand, nämlich $2^{\cdot^{\cdot^{2^n}}}$ [6]. Wie noch im weiteren Verlauf dieser Arbeit gezeigt wird, kann für eine Teilmenge von S1S ein exponentielles Verfahren angegeben werden, das einen großen Teil digitaler Schaltungsspezifikationen (u.a. ganz LTL) abdeckt.

1.3 Vergleich von temporalen Logiken mit arithmetischen Sprachen

Um entscheiden zu können, welcher Formalismus zur Verifikation zeitlichen Verhaltens für die jeweils gestellte Aufgabenstellung am besten geeignet ist, werden hier die Vor- und Nachteile der in Frage kommenden Formalismen gegenübergestellt. Viele Entscheidungsverfahren stützen sich bei der Verifikation auf die bereits genannten temporalen Logiken. So verwendet der Beweiser *'SMV'* beispielsweise CTL zur Verifikation. Besonderer Schwerpunkt sei hier auf den Ausgangspunkt der Impulsdiagramme gerichtet. In diesem Zusammenhang ist zunächst zu erwähnen, daß die gegebenen Impulsdiagramme in Gleichungen und Ungleichungen übersetzt werden können. Diese Gleichungen und Ungleichungen bilden dann Formeln der Arithmetik.

Da Impulsdiagramme keine verzweigende Zeit besitzen, sondern einer linearen Zeit folgen, genügt demnach auch ein lineares Zeitmodell zur Verifikation. Somit ist LTL als temporale Logik von besonderem Interesse.

Auf der anderen Seite lassen sich Impulsdiagramme sehr leicht in Arithmetiken ausdrücken, insbesondere in S1S können Impulsdiagramme kurz und prägnant übersetzt werden. Weiterhin hat S1S gegenüber LTL den Vorteil, daß S1S mächtiger ist. So kann in LTL beispielsweise nicht spezifiziert werden, daß ein Signal bei allen geraden Zeitpunkten einen definierten Wert haben soll, und bei allen ungeraden Zeitpunkten einen beliebigen Wert annehmen darf [29].

Dieser Umstand ergibt sich aus der Tatsache, daß bei temporalen Logiken ein Bezug auf bestimmte Zeitpunkte nicht direkt geschehen kann, sondern nur indirekt über bereits definierte Signale. In S1S hingegen ist ein direkter Bezug auf jeden Zeitpunkt möglich. Dieser direkte Bezug macht die Handhabung von S1S einfacher, wie das Beispiel in Abb. 1.3 verdeutlicht:

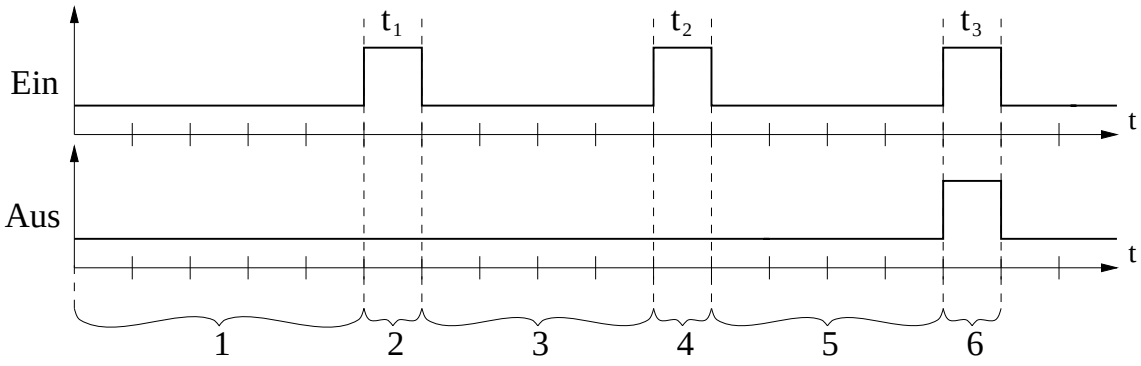

Abbildung 1.3: Beispiel: Modulo-3-Zähler

Bei diesem Beispiel handelt es sich um einen Modulo-3-Zähler, der ein Eingabesignal und ein Ausgabesignal besitzt. Die Schaltung hat die Aufgabe, das Ausgabesignal solange auf Null zu halten, bis das Eingabesignal zum dritten mal Eins wird. Abbildung 1.3 zeigt das hierzu

passende Impulsdiagramm, wobei das obere Signal 'Ein' das Eingabesignal und das untere Signal 'Aus' das Ausgabesignal der Schaltung ist. Zu den Zeitpunkten t_1, t_2, t_3 nimmt die Eingabe den Wert Eins an. Das Ausgabesignal hingegen soll dagegen nur zum Zeitpunkt t_3 den Wert Eins annehmen.

Bei der Spezifikation dieses Beispieles mit Hilfe von LTL muß man den Zeitstrahl in Intervalle einteilen, wobei sich innerhalb eines Intervalls kein Eingabesignal ändert. In diesem Beispiel erhält man sechs Intervalle:

$$\begin{array}{ll}
0 \leq t < t_1 & [(\neg Aus)\ \mathsf{U}\ Ein] \\
t = t_1 & [(\neg Aus)\ \mathsf{W}\ Ein] \\
t_1 < t < t_2 & [\mathsf{X}\,[(\neg Aus)\ \mathsf{U}\ Ein]\ \mathsf{W}\ Ein] \\
t = t_2 & [\mathsf{X}[(\neg Aus)\ \mathsf{W}\ Ein]\ \mathsf{W}\ Ein] \\
t_2 < t < t_3 & [\mathsf{X}[\mathsf{X}\,[Aus\ \mathsf{U}\ Ein]\ \mathsf{W}\ Ein]\ \mathsf{W}\ Ein] \\
t = t_3 & [\mathsf{X}[\mathsf{X}[Aus\ \mathsf{W}\ Ein]\ \mathsf{W}\ Ein]\ \mathsf{W}\ Ein]
\end{array}$$

An dieser Stelle soll die Beschreibung des dritten Intervalls beispielhaft erläutert werden. Das äußere [W] gibt an, daß zunächst gewartet wird, bis 'Ein' zum ersten Mal gilt $(t = t_1)$. Dann wird das Argument

$$\mathsf{X}\,[(\neg Aus)\ \mathsf{U}\ Ein]$$

ausgewertet, d.h. aufgrund der Operation $\mathsf{X}t$ um eine Zeiteinheit verschoben. Der Term

$$[(\neg Aus)\ \mathsf{U}\ Ein]$$

wird dann zum Zeitpunkt $t = t_1 + 1$ ausgewertet. Es wird wiederum gewartet, bis 'Ein' wieder wahr ist $(t = t_2)$. Von $t = t_1 + 1$ bis $t = t_2$ muß $\neg$ 'Aus' gelten.

Umgangssprachlich ausgedrückt besagt diese LTL-Formel, daß das Ausgangssignal mindestens solange Null ist, bis das Eingangssignal wieder Eins wird, falls bereits vor diesem Zeitpunkt das Eingangssignal einmal wahr war. An diesem Beispiel wird deutlich, daß ein Bezug auf gewisse Zeitpunkte, z.B. den Zeitpunkt t_3, nur über Signaländerungen möglich ist.

Dasselbe Beispiel kann mit Hilfe von S1S folgendermaßen beschrieben werden:

$$\forall t_1 t_2 t_3.[\forall t.Ein(t) \leftrightarrow (t = t_1) \vee (t = t_2) \vee (t = t_3)] \rightarrow [\forall t.Aus(t) \leftrightarrow (t = t_3)]$$

Bei S1S ist im Unterschied zu LTL ein direkter Bezug auf die Zeit möglich. Das bedeutet, daß es möglich ist, unabhängig von den Belegungen der Eingangssignale Zeitpunkte zu definieren. In diesem Beispiel wird zuerst von drei beliebigen Zeitpunkte t_1, t_2, t_3 ausgegangen. Danach wird die Position dieser Zeitpunkte insoweit festgelegt, daß sie mit den 'High-Pegeln' des Eingangssignals übereinstimmen. Das bedeutet, daß über diese drei Zeitpunkte das Eingangsverhalten des Modulo-3-Zählers angegeben werden kann. Die Zeitpunkte t_1, t_2, t_3 definieren

nämlich die Zeiten, wo sich die 1-Impulse des Eingangssignals befinden. In der Conclusio wird dann behauptet, daß das Ausgangssignal genau dann Eins wird, wenn der dritte Zeitpunkt erreicht ist. Das beinhaltet auch, daß das Ausgangssignal nach Erreichen des 'High-Pegels' diesen Zustand genau einen Takt lang beibehält.

Der direkte Bezug auf die Zeit macht S1S zunächst nicht mächtiger als LTL. Er erleichtert aber oft die Beschreibung von Schaltungen, da Impulsdiagramme mit S1S globaler beschrieben werden können als mit LTL, wo jede Änderung der Eingangssignale relativ zu anderen Signaländerungen erfolgt.

S1S ist aber dennoch mächtiger als LTL wie das folgende Beispiel aus [29] illustriert:

Beispiel:

$$\forall Even.\,[Even(0) \wedge (\forall t.Even(S(t)) = \neg Even(t))] \rightarrow [\forall t.Even(t) \rightarrow P(t)]$$

Dieses Beispiel ist in LTL nicht realisierbar, da P für alle ungeraten Zeitpunkte nicht definiert ist.

Nachteilig an S1S ist der hohe Aufwand [6, 4] (nicht elementar: $2^{\cdots^{2^n}}$), falls ganz S1S realisiert wird. Für eine ausgewählte Teilmenge von S1S, kann der Aufwand derart verkleinert werden, daß er mit dem Aufwand von LTL, der exponentiell ist, gleichgesetzt werden kann. Es stellt sich demnach die Frage, in wieweit die S1S eingeschränkt werden muß, um einerseits noch mächtiger zu sein als die LTL. Auf der anderen Seite soll der Aufwand nicht größer als der der LTL sein.

1.4 Aufgabenstellung und Gliederung

In dieser Ausarbeitung wird eine Teilmenge von S1S angegeben, wobei der Aufwand um diese Teilmenge zu entscheiden, nur noch deterministisch exponentiell ist. Diese Teilmenge ist aber immer noch mächtiger als LTL.

Das grundsätzliche Ziel der vorliegenden Arbeit ist

(i) Definition einer Teilmenge von S1S, welche

  (a) LTL umfaßt

  (b) exponentiellen Entscheidungsaufwand hat

  → Es zeigt sich, daß diese Teilmenge eine echte Obermenge von LTL ist.

(ii) Entwicklung eines Entscheidungsverfahrens dazu,

- das sich an Büchis Entscheidungsverfahren orientiert

- aktuelles Wissen wie BDDs oder Präfixautomaten berücksichtigt

Büchis Verfahren beruht auf einer Übersetzung in indeterministische ω-Automaten (zu Details siehe Kapitel 3).

Der schlechte Aufwand des Original-Verfahrens liegt darin begründet, daß eine mehrfache Komplementierung von indeterministischen Automaten notwendig wird. Da im ursprünglichen Verfahren die S1S in einen indeterministischen Automaten transformiert wird, muß demnach an dieser Stelle eine Verbesserung des Verfahrens durchgeführt werden.

In dieser Ausarbeitung wird ein Verfahren vorgestellt, welches ebenfalls auf der Transformation von S1S in ω-Automaten beruht. Im Gegensatz zu Büchis Verfahren wird hier aber ein deterministischer Automat erzeugt. Dieses Verfahren unterscheidet im Gegensatz zu Büchis Verfahren zwischen Transitions-, Sicherheits- und Lebendigkeitseigenschaften. Ziel ist ferner, die Transitionsrelationen als Transitionsfunktionen darzustellen und somit einen deterministischen Automaten zu erhalten.

Sollte ganz S1S entschieden werden müssen, so verbessert sich der theoretische Aufwand in dem hier vorgestellten Algorithmus nicht. Er bleibt nicht elementar. Praktisch sind aufgrund effizienter Implementierung aber auch hier Verbesserungen in den tatsächlichen Laufzeiten zu erwarten! Eine Verbesserung kann jedoch erreicht werden, falls S1S eingeschränkt wird. Die im weiteren Verlauf beschriebenen Schritte zielen darauf hin, die Einschränkung möglichst klein zu halten.

In Kapitel 2 werden kurz die benötigten formalen Grundlagen angegeben. Im darauf folgenden Kapitel 3 wird das Original-Verfahren von Büchi vorgestellt, und in Kapitel 4 wird dann das neue Verfahren angegeben und mit dem Büchi-Verfahren verglichen. Schließlich werden in Kapitel 5 berechnete Beispiele aufgeführt und die Ergebnisse zusammengefaßt.

Kapitel 2

Formale Grundlagen

In diesem Kapitel wird im besonderen Maße auf den Ansatz der Verifikation eingegangen, der sich formaler Sprachen bedient. Um eine solche Sprache erkennen zu können, werden Automaten benötigt, die ja bereits in ihrer Funktionsweise der Hardware nahe stehen. Sowohl Hardware, als auch Automaten empfangen einerseits einen Eingabestrom und erzeugen einen dazu passenden Ausgabestrom. Ebenso besitzen beide endlich viele interne Zustände, die anhand einer Zustandsübergangsfunktion von einem aktuellen Zustand in einen Folgezustand wechseln. Man kann demnach sowohl bei der Hardware, als auch bei Automaten von Übersetzern sprechen, die einen Eingabestrom $\vec{i}$ in einen Ausgabestrom $\vec{o}$ übersetzen.

Sollen nun zwei Hardware-Spezifikationen bzgl. Äquivalenz untersucht werden, so kann man stattdessen die Sprachen untersuchen, die von den dazu passenden Automaten akzeptiert werden. Die Eingabeströme dieser Automaten sind nicht begrenzt, d.h. es handelt sich bei der Eingabe um unendlich lange Wörter. Demzufolge entstehen hierbei Automaten über unendlich langen Eingabesequenzen, die auch ω-Automaten genannt werden.

2.1 Temporale Logiken

Bereits in Kapitel 1.2 wurden einige Formalismen zur Verifikation digitaler Schaltungen aufgezählt. Ein Formalismus, der sich in der Praxis etabliert hat sind die temporalen Logiken. Die in Kapitel 2.4 definierte Arithmetik basiert auf einem linearen Zeitmodell. Daher werden hier nur lineare temporale Logiken behandelt. Neben der LTL spielt QLTL, die eine Obersprache der LTL ist, eine wichtige Rolle.

Definition 1 (Syntax von QLTL) *Über einer gegebenen endlichen Menge von Variablen V_Σ wird die Menge von QLTL-Formeln wie folgt rekursiv definiert:*

- *jede Variable von QLTL ist eine Formel, d.h. $V_\Sigma \subset \mathcal{L}_\Sigma^{LTL}$.*

- *QLTL-Formeln sind bzgl. den booleschen Operationen abgeschlossen, es gilt demnach* $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi \in \mathcal{L}_{\Sigma}^{LTL}$ *falls* $\varphi, \psi \in \mathcal{L}_{\Sigma}^{LTL}$.
- *QLTL-Formeln sind bzgl. den temporalen Operationen abgeschlossen, es gilt demnach* $\mathsf{X}\varphi$, $\mathsf{G}\varphi$, $\mathsf{F}\varphi$, $[\varphi\ \mathsf{U}\ \psi]$, $[\varphi\ \mathsf{W}\ \psi]$ *sowie* $[\varphi\ \mathsf{B}\ \psi] \in \mathcal{L}_{\Sigma}^{LTL}$, *falls* $\varphi, \psi \in \mathcal{L}_{\Sigma}^{LTL}$.
- *falls* $\varphi \in \mathcal{L}_{\Sigma}^{LTL}$ *und* $x \in V_{\Sigma}$, *dann ist auch* $\forall x.\varphi \in \mathcal{L}_{\Sigma}^{LTL}$ *und* $\exists x.\varphi \in \mathcal{L}_{\Sigma}^{LTL}$.

Hierbei haben die temporalen Operatoren folgende Bedeutung:

- $\mathsf{X}\varphi$ bedeutet, daß der Wahrheitswert von φ beim nächsten Zeitpunkt betrachtet wird.
- $\mathsf{G}\varphi$ bedeutet, daß der Wahrheitswert von φ für jeden Zeitpunkt ab dem augenblicklichen Zeitpunkt betrachtet wird, d.h. φ muß ab dem augenblicklichen Zeitpunkt immer gelten.
- $\mathsf{F}\varphi$ bedeutet, daß der Wahrheitswert irgendwann in einem zukünftigen Zeitpunkt betrachtet wird, d.h. φ muß irgendwann einmal in einem zukünftigen Zeitpunkt gelten.
- $[\varphi\ \mathsf{U}\ \psi]$ bedeutet, daß φ vom augenblicklichen Zeitpunkt an gelten muß, bis ψ zum ersten Mal wahr wird. Falls ψ nie wahr wird, dann muß φ immer gelten. Auf den 'Until-Operator' kann verzichtet werden, da er in dem Transformationsalgorithmus nicht benötigt wird.

Neben den hier angegebenen Operatoren kann man auch noch weitere Operatoren definieren:

- $[\varphi\ \mathsf{W}\ \psi]$ bedeutet, daß φ ab dem Zeitpunkt gilt, wenn ψ zum ersten Mal wahr wird. Diesen Operator nennt man 'WHEN-Operator'.
- $[\varphi\ \mathsf{B}\ \psi]$ bedeutet, daß φ bis zu dem Zeitpunkt, an dem ψ zum ersten Mal wahr wird, mindestens einmal gelten muß. Diesen Operator nennt man 'BEFORE-Operator'.

Alle drei Operatoren, den 'UNTIL-Operator', den 'WHEN-Operator' und den 'BEFORE-Operator' kann man durch eine prädikatenlogische Formel ersetzen. Diese Ersetzung beinhaltet die Einbettung von LTL in S1S.

LTL kann leicht in S1S transformiert werden. Dabei wird jeder Operator als eine Funktion vom Typ $\mathbb{N} \rightarrow \mathbb{B}$ interpretiert:

- $\mathsf{X}a = a(S(t))$
- $\mathsf{G}a = \forall t.a(t)$
- $\mathsf{F}a = \exists t.a(t)$

- $[a \mathsf{W} b] = (\forall \delta.\, (\forall t.t < \delta \rightarrow \neg b(t)) \wedge b(\delta) \rightarrow a(\delta))$
- $[a \mathsf{U} b] = \left(\begin{array}{l} ((\forall t.\neg b(t)) \rightarrow (\forall t.a(t))) \wedge \\ \left(\begin{array}{r} \forall \delta.(\forall t.t < \delta \rightarrow \neg b(t)) \wedge b(\delta) \\ \rightarrow (\forall t.t < \delta \rightarrow a(t)) \end{array} \right) \end{array} \right)$
- $[a \mathsf{B} b] = (\forall \delta.\, (\forall t.t < \delta \rightarrow \neg b(t)) \wedge b(\delta) \rightarrow (\exists t.t < \delta \wedge a(t)))$

  wobei a, b aussagenlogische Formeln und t, δ numerische Variablen sind.

Im allgemeinen kann eine Sprache, die durch einen ω-Automaten akzeptiert wird, durch eine aussagenlogische Formel der folgenden Art angegeben werden:

$$\mathcal{A}(\vec{i}) = \exists \vec{P}.\mathcal{I}(\vec{P}) \wedge [\mathsf{G}\, \mathcal{T}(\vec{i}, \vec{P})] \wedge \Theta(\vec{i}, \vec{P})$$

Dabei ist $\mathcal{T}(\vec{i}, \vec{P})$ aussagenlogische Formel in $\vec{i}$, $\vec{P}$ und $\mathsf{X}\vec{P}$, und $\Theta(\vec{i}, \vec{P})$ eine LTL-Formel. Ein Wort $\vec{i}$ wird nun genau dann von dem Automaten akzeptiert, wenn die Formel $\mathcal{A}(\vec{i})$ gilt. Falls es sich bei dem Automaten um einen deterministischen Automaten handelt, ist die Transitionsrelation $\mathcal{T}(\vec{i}, \vec{P})$ eine Übergangsfunktion der Form $\mathsf{G}\left(\mathsf{X}P_k = \Omega_k(\vec{i}, \vec{P})\right)$, wobei $\Omega_k(\vec{i}, \vec{P})$ aussagenlogische Formel in $\vec{i}$, $\vec{P}$ ist.

2.2 Reguläre Sprachen und ω-Automaten

Sei $\Sigma = \{\sigma_1, \ldots, \sigma_m\}$ eine endliche Menge von Symbolen, welche im folgenden Alphabet genannt wird. Eine Aneinanderreihung von Symbolen aus Σ bezeichnet man als Worte. Speziell wird

- mit Σ^* die Menge aller endlichen Aneinanderreihungen von Symbole aus Σ,
- mit Σ^ω die Menge aller unendlichen Aneinanderreihungen von Symbolen aus Σ
- und mit Σ^∞ die Vereinigung von Σ^* und Σ^ω bezeichnet.

Mengen von Worten bezeichnet man als Sprache. Für zwei Sprachen
$L_1 \subseteq \Sigma^*$ und $L_2 \subseteq \Sigma^\infty$ definiert man $L_1 \cdot L_2 := \{\alpha\beta \mid \alpha \in L_1, \beta \in L_2\}$ als die Konkatenation von L_1 und L_2.
Ist $\alpha \in \Sigma^\omega$, so bezeichnet man die einzelnen Symbole aus Σ von α mit $\alpha^{(i)}$,
d.h. $\alpha := \alpha^{(0)}\alpha^{(1)}\alpha^{(2)} \ldots$.

Definition 2 (Reguläre Sprache über Σ) *Eine reguläre Sprache ist eine Sprache, die von einer rechtslinearen Grammatik erzeugt werden kann. Die Menge der regulären Sprachen ist die kleinste Teilmenge der Potenzmenge über Σ $\wp(\Sigma)$, welche die folgenden Eigenschaften erfüllt:*

1. *Jede endliche Teilmenge von Σ^* ist regulär.*

2. *Sind L_1 und L_2 reguläre Sprachen, so sind auch $L_1 \cup L_2$, $L_1 \cdot L_2$ und L_1^* reguläre Sprachen.*

Das Analogon zu den regulären Sprachen auf endlichen Wörtern sind bei Sprachen auf unendlichen Wörtern die sogenannten ω-regulären Sprachen:

Definition 3 (ω-reguläre Sprachen über Σ) *Eine Sprache $L \subseteq \Sigma^\omega$ aus unendlich langen Wörtern heißt* **ω-regulär**, *falls es reguläre Sprachen $A_1, \ldots, A_n, B_1, \ldots, B_n$ gibt, so daß gilt:*

$$L = \bigcup_{i=1}^{n} A_i B_i^\infty$$

.

Definition 4 (Zustandsübergangsgraph) *Ein Zustandsübergangsgraph $\mathcal{G}$ ist ein Tupel (Q, Σ, δ, q_0). Σ und Q sind endliche Mengen, wobei mit Σ das Eingabealphabet und mit Q die Menge der Zustände bezeichnet wird. δ ist eine Funktion von $Q \times \Sigma$ nach $\wp(\Sigma)$ und wird Übergangsrelation genannt. Mit q_0 wird der Anfangszustand des Zustandsübergangsgraphen bezeichnet.*

$\mathcal{G}$ ist deterministisch, falls für alle $(q, \alpha) \in Q \times \Sigma$ die Menge $\delta(q, \alpha)$ aus einem einzigen Element besteht. Ansonsten heißt $\mathcal{G}$ indeterministisch.

Definition 5 (Endlicher Automaten über endlich langen Wörtern) *Sei $\mathcal{G} = (Q, \Sigma, \delta, q_0)$ ein Zustandsübergangsgraph und sei $F \subseteq Q$ eine Menge von Endzuständen. Dann bildet das Tupel $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ einen endlichen Automaten der die folgende Sprache akzeptiert:*

$$\mathcal{L}(\mathcal{A}) := \{\alpha \in \Sigma^* \mid \delta(q_0, \alpha) \in F\}$$

Ein deterministischer Automat $\mathcal{A} = (Q, \Sigma, \delta, q_0, Q_F)$, der einen Eingabestrom α bearbeitet, wird aufgrund einer Zustandsübergangsfunktion δ nach jedem eingelesenen Symbol in einen Folgezustand $q' \in Q$ wechseln.
Die Sequenz $\beta = \beta^{(0)}\beta^{(1)}\beta^{(2)} \ldots$ der Zustände, welche ausgehend von einem Anfangszustand q_0 bei der Abarbeitung des Eingabestroms α durchlaufen wird, nennt man einen **Durchlauf**.

Die Menge aller Durchläufe eines Wortes α durch einen Zustandsübergangsgraphen $\mathcal{G} = (Q, \Sigma, \delta, q_0)$ wird hier mit $\mathbf{run}(\alpha, \mathcal{G})$ bezeichnet.

Die Menge der Zustände, welche von einem Durchlauf $\beta \in \mathbf{run}(\alpha, G)$ unendlich oft durchlaufen werden, wird hier mit $\mathbf{inf}(\beta) := \{q \in Q \mid \forall t_1.\exists t_2.\beta^{(t_1+t_2)} = q\}$ bezeichnet.

2.2.1 Büchi-Automaten

Büchi-Automaten sind endliche Automaten, die durch eine Akzeptanzbedingung in der Lage sind, unendlich lange Sequenzen zu akzeptieren.

Definition 6 (Büchiautomat) *Sei $\mathcal{G} = (Q, \Sigma, \delta, q_0)$ ein Zustandsübergangsgraph und sei ferner $Q_F \subseteq Q$. Dann bildet*

$$\mathcal{A} = (Q, \Sigma, \delta, Q_0, Q_F)$$

einen **Büchiautomaten**, *welcher die folgende Sprache akzeptiert:*

$$L_B(\mathcal{A}) := \{\alpha \in \Sigma^\infty \mid \exists \beta \in \mathbf{run}(\alpha, \mathcal{G}).\mathbf{inf}(\beta) \cap Q_F \neq \emptyset\}$$

Q bezeichnet die Menge der Zustände des Automaten, Σ das Eingabealphabet, δ die Übergangsrelation des Automaten und Q_0 die Menge der Startzustände. Weiter werden mit Q_F die Finalzustände des Automaten bezeichnet. Ein Büchiautomat $\mathcal{A}$ ist deterministisch, genau dann wenn der Zustandsübergangsgraph $\mathcal{G}$ deterministisch ist.

Ein ω-Wort wird von einem Büchi-Automaten genau dann akzeptiert, wenn der Automat bei der Abarbeitung des Wortes mindestens einen der Zustände aus Q_F unendlich oft durchläuft. Das bedeutet, daß es einen unendlichen Zyklus über mindestens einen ausgezeichneten Zustand gibt.

Beispiel:
Sei $\Sigma = \{a, b, c\}$. Definiere $L \subseteq \Sigma^\omega$, wobei $\alpha \in L$ gdw. nach Auftreten eines Symbols a folgt irgendwann mindestens ein Symbol b in α. Das Bild 2.1 zeigt den zu diesem Beispiel gehörigen

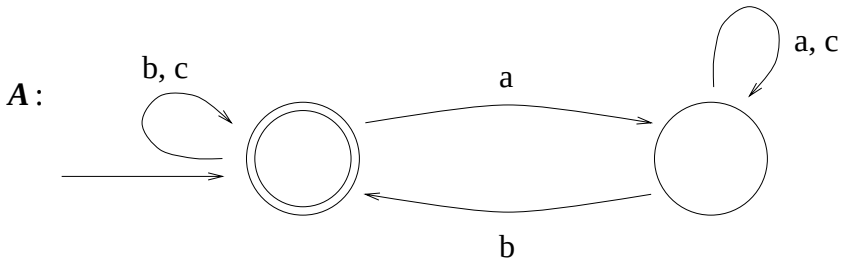

Abbildung 2.1: Beispiel: deterministischer Büchiautomat

Büchiautomaten. Es handelt sich hier um einen deterministischen Büchi-Automaten, da es

1. nur einen Startzustand gibt und
2. die Übergangstransition eine Übergangsfunktion, also für jeden Zustand und jede Eingabe eindeutig ist.

Als nächstes soll dieser Automat nun komplementiert werden, es soll also die Sprache $\Sigma^\infty - L$ dargestellt werden. Der Automat soll demnach alle ω-Worte akzeptieren, die folgende Bedingung erfüllen:

Jedes Wort aus α enthält ein a, ohne daß ein b folgt.

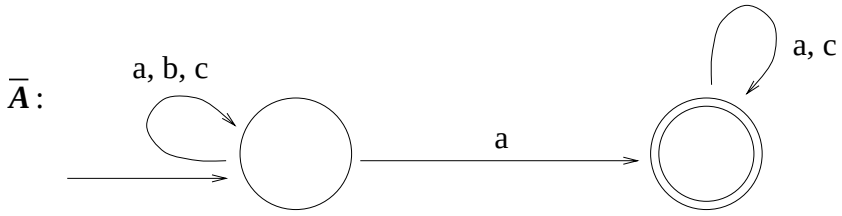

Abbildung 2.2: Beispiel: indeterministischer Büchiautomat

Zum Beispiel erhält man dazu den in Bild 2.2 angegebenen Automaten. In diesem Fall handelt es sich nicht mehr um einen deterministischen Büchiautomaten, da die Übergangstransition nicht mehr eindeutig ist. Formel kann man Büchiautomaten in QLTL beschreiben, und spricht dann von Büchiformeln. Ebenso wie bei den Büchiautomaten wird zwischen deterministischen und indeterministischen Büchiformeln unterschieden.

Alle Automatenformeln von ω-Automaten besitzen:

1. Eine aussagenlogische Formel, genannt *Initialformel*, die die Zustandsmenge definiert, in der der ω-Automat startet. Diese Formel ist zum Zeitpunkt t_0 gültig und hat folgendes Aussehen:

   - für eine deterministische Automatenformel:

     $$\bigwedge_{j=1}^{n} q_j = \omega_j \text{ mit } \omega_j \in \{T,\ F\}$$

   - für eine indeterministische Automatenformel: $\Omega_0[\vec{q}]$

2. Eine aussagenlogische Formel, die die Übergangstransition darstellt. Bei einem deterministischen ω-Automaten ist diese Übergangstransition eine Übergangsfunktion $\vec{\Omega}$ und hat folgendes Aussehen:

   $$\bigwedge_{j=1}^{n} \mathsf{G}\left[\mathsf{X}q_j = \Omega_j(\vec{i}, \vec{q})\right]$$

   Bei einem indeterministischen Automaten hingegen ist die Übergangstransition eine Übergangsrelation $\mathsf{G}\left(\Omega\left(\vec{i}, \vec{q}, \mathsf{X}\vec{q}\right)\right)$, wobei Ω eine beliebige aussagenlogische Formel ist.

Büchiautomaten gehören ebenfalls zu der Klasse der ω-Automaten. Demnach können Büchiformeln folgendermaßen definiert werden:

Definition 7 (Büchiformeln) *Seien Ω_0, Ω und Θ aussagenlogische Formeln. Dann hat eine deterministische Büchiformel folgendes Aussehen:*

$$\mathcal{B}(\vec{i}) := \left(\begin{array}{l} \exists \vec{q}. \\ \quad \left[(\vec{q} \leftrightarrow \vec{\omega}) \wedge \mathsf{G}\left(\mathsf{X}\vec{q} \leftrightarrow \vec{\Omega}(\vec{i}, \vec{q}) \right) \right] \wedge \\ \quad [\mathsf{GF}\Phi(\vec{q})] \end{array} \right)$$

wogegen die indeterministische Büchiformel folgendes Aussehen hat:

$$\mathcal{B}(\vec{i}) := \left(\begin{array}{l} \exists \vec{q}. \\ \quad \left[(\Omega_0[\vec{q}]) \wedge \mathsf{G}\left(\Omega\left(\vec{i}, \vec{q}, \mathsf{X}\vec{q} \right) \right) \right] \wedge \\ \quad [\mathsf{GF}\Phi(\vec{q})] \end{array} \right)$$

Die GF-Formel, $\mathsf{GF}\,(\Phi(\vec{q}))$ definiert die Finalzustandsmenge. Diese Formel wird Akzeptanzbedingung genannt. Hierbei werden mit $\vec{i}$ freie Eingabevariablen und mit $\vec{q}$ Zustandsvariablen und mit $\vec{\omega} \in \mathcal{B}^{\|\vec{\omega}\|}$ ein Tupel von Anfangszuständen bezeichnet.
Automaten, die in der Lage sind unendliche Wörter zu akzeptieren, unterscheiden sich hauptsächlich im Aufbau dieser Akzeptanzbedingung.

2.2.2 Präfix-Automaten

Definition 8 (Präfixautomat) *Sei $\mathcal{G} = (Q, \Sigma, \delta, Q_0)$ ein Zustandsübergangsgraph und sei ferner $Q_F = \{(R_0, S_0), \ldots, (R_f, S_f)\}$ mit $R_j, S_j \subseteq Q$ gegeben. Dann bildet*

$$\mathcal{A} = (Q, \Sigma, \delta, q_0, \{(R_0, S_0), \ldots, (R_f, S_f)\})$$

einen **Präfixautomaten**, *welcher die folgende Sprache akzeptiert:*

$$L_P(\mathcal{A}) := \left\{ \alpha \in \Sigma^\infty \mid \exists \beta \in \mathbf{run}(\alpha, \mathcal{G}). \exists j. \leq f. \forall t. \beta^{(t)} \in S_j \wedge \exists t. \beta^{(t)} \in R_j \right\}$$

Q bezeichnet die Menge der Zustände des Automaten, Σ das Eingabealphabet, δ die Übergangsrelation des Automaten und Q_0 die Menge der Startzustände.

Ein Präfixautomat $\mathcal{A}$ ist deterministisch, genau dann wenn der Zustandsübergangsgraph $\mathcal{G}$ deterministisch ist.

Ein ω-Wort α wird von einem Präfixautomaten genau dann akzeptiert, wenn es ein Paar $(R_j, S_j) \in Q_f$ gibt, so daß ein Durchlauf β von α existiert, welcher von Beginn an nur durch S_j läuft und dann einmal die Menge R_j besucht. Ferner kann man verlangen, daß $R_j \subseteq S_j$ für ein $j \leq f$. q_0 muß aus S_j sein, da sonst die Startbedingung verletzt wird.

Man kann zwei verschiedene Arten von Präfixformeln unterscheiden.

Definition 9 (Präfixformel) *Seien Ω_0, Ω_j und Θ_j aussagenlogische Formeln für $j = 0 \ldots f$. Dann stellt die erste nun folgende Formel eine Präfixformel 1.Art und die darauf folgende Formel eine Präfixformel 2.Art dar:*

$$\mathcal{P}(\vec{i}) := \left(\begin{array}{l} \exists \vec{q}. \\ \quad \Big[(\vec{q} \leftrightarrow \vec{\omega}) \wedge \mathsf{G}\left(\mathsf{X}\vec{q} \leftrightarrow \vec{\Omega}(\vec{i},\vec{q})\right)\Big] \wedge \\ \quad \bigvee\limits_{j=0}^{f} \Big[\mathsf{G}\Phi_j(\vec{i},\vec{q})\Big] \wedge \Big[\mathsf{F}\Psi_j(\vec{i},\vec{q})\Big] \end{array} \right)$$

$$\mathcal{P}(\vec{i}) := \left(\begin{array}{l} \exists \vec{q}. \\ \quad \Big[(\vec{q} \leftrightarrow \vec{\omega}) \wedge \mathsf{G}\left(\mathsf{X}\vec{q} \leftrightarrow \vec{\Omega}(\vec{i},\vec{q})\right)\Big] \wedge \\ \quad \bigwedge\limits_{j=0}^{f} \Big[\mathsf{G}\Phi_j(\vec{i},\vec{q})\Big] \vee \Big[\mathsf{F}\Psi_j(\vec{i},\vec{q})\Big] \end{array} \right)$$

Die Teilformeln $\forall t.\Phi_j(\vec{i},\vec{q})$ und $\exists t.\Psi_j(\vec{i},\vec{q})$ werden Sicherheits- und Lebendigkeitseigenschaften von $P(\vec{i})$ genannt.

Für diese Arbeit sind die Präfixformeln der 2.Art wichtiger, da die S1S in Präfixformeln der 2.Art transformiert wird.

Theorem 1 *Es lassen sich folgende Theoreme angeben [20]*

(i) Deterministische Präfixformeln sind abgeschlossen bzgl. boolescher Operationen [20]

(ii) Deterministische Büchiformeln sind bzgl. der Negation nicht abgeschlossen [34]

(iii) Deterministische Präfixformeln sind eine echte Teilmenge von deterministischen Büchiformeln

(iv) Indeterministische Präfixformeln sind bzgl. boolescher Operationen nicht abgeschlossen [20]

(v) Indeterministische Präfixformeln in Det_{FG} = Komplement von $Det_{Büchi}$ transformiert werden [20]

(vi) Det_{FG} = $\mathcal{N}_{FG}$ mit üblicher Determinisierung [36]

(vi) Safras Konstruktion konvertiert einen Büchiautomaten mit n Zuständen in einen deterministischen Rabinautomaten mit $2^{O(nlog(n))}$ Zuständen und $O(n)$ Akzeptanzpaaren [33]

Indeterministische Automaten starten nicht zwangsweise in einem einzigen Startzustand, sondern beginnen mit einer Startzustandsmenge. Außerdem wechseln diese Automaten aus einer Zustandsmenge in eine nachfolgende Zustandsmenge nicht anhand einer Übergangsfunktion, sondern Anhand einer Übergangsrelation.

Sowohl deterministische als auch indeterministische Automaten können komplementiert werden. Während bei deterministischen Automaten der Aufwand für die Komplementbildung einfach exponentiell ist, ist der Aufwand bei der Komplementbildung indeterministischer Automaten $2^{O(nlog(n))}$. Daher sind deterministische Automaten vorzuziehen, falls eine Komplementierung notwendig ist.

2.3 Prädikatenlogik

Die Prädikatenlogik erster Ordnung (PL1) kann über folgender Signatur definiert werden:

Definition 10 (Signatur der PL1) *Eine Signatur der PL1 ist ein Tripel* $\Sigma_0 = (F_\Sigma,\ P_\Sigma,\ \alpha_\Sigma)$ *mit:*

- F_Σ *und* P_Σ *sind endliche gegebene Mengen, die paarweise disjunkt sind.* F_Σ *und* P_Σ *werden auch Variablen* V_Σ *genannt.*
- F_Σ *ist die Menge der Funktionssymbole und* P_Σ *ist die Menge der Prädikatssymbole.*
- α_Σ *ordnet jedem Funktions- und Prädikatssymbol eine Stelligkeit zu.*
- F_Σ *mit* $\alpha_\Sigma(F_\Sigma) = 0$ *wird Konstante genannt.*

In dieser Arbeit sind nur nullstellige und einstellige Variablen von Bedeutung, also $\alpha_\Sigma(V_\Sigma) \leq 1$. Daher können Typen wie folgt definiert werden:

Definition 11 (Typen der PL1) *Falls* a *und* b *Typen sind, dann sind auch* $a \to b$ *und* $a \times b$ *Typen. Folgende Typen werden in dieser Arbeit benötigt:*

- *für numerische Variablen* t*:* $typ(t) = \mathbb{N}$
- *für nullstellige Prädikatsvariablen* P*:* $typ(P) = \mathbb{B}$
- *für die einstellige Funktion* $\mathsf{SUC}(t)$*:* $typ(\mathsf{SUC}(t)) = \mathbb{N} \to \mathbb{N}$
- *für einstellige Prädikatsvariablen* $P(t)$*:* $typ(P(t)) = \mathbb{N} \to \mathbb{B}$

In der PL1 können Terme, atomare Formeln und Formeln folgendermaßen definiert werden:

Definition 12 (Terme der PL1) *Die Menge der Terme $Term_\Sigma$ über Σ ist induktiv definiert durch:*

- $c \in Term_\Sigma$ *mit* $c \in C_\Sigma$
- $v \in Term_\Sigma$ *mit* $v \in V_\Sigma$
- *für* $f \in F_\Sigma$, $\alpha(f) = n$ *und* $t_1, \ldots t_n \in Term_\Sigma$, *ist auch* $f(t_1, \ldots t_n) \in Term_\Sigma$

Wie bereits erwähnt ist in dieser Arbeit nur die einstellige Funktion SUC von Bedeutung.

Definition 13 (atomare Formeln der PL1) *Die Menge der atomaren Formeln AT_Σ über Σ ist definiert durch:*

$$AT_\Sigma := \{s = t \mid s,\ t \in Term_\Sigma\} \cup \{p(t_1 \ldots t_n) \mid p \in P_\Sigma,\ \alpha_\Sigma(P) = n,\ t_1, \ldots t_n \in Term_\Sigma\}$$

Definition 14 (Formeln der PL1) *Die Menge der Formeln For_Σ über Σ ist induktiv definiert durch:*

- $\{T,\ F\} \in For_\Sigma$
- $AT_\Sigma \in For_\Sigma$
- *für* $v \in V_\Sigma$ *und* $\varphi,\ \psi \in For_\Sigma$ *ist auch*
  - $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi$, $\varphi \rightarrow \psi$ *und* $\leftrightarrow \vee\psi$
  - $\forall x.\varphi$ *und* $\exists x.\varphi$

2.4 Monadische Arithmetik zweiter Ordnung mit einem Sukzessor

Wie bereits in Kapitel 1.4 angedeutet, ist die S1S Schwerpunkt dieser Arbeit. In der folgenden Definition wird die Syntax der S1S angegeben:

Definition 15 (Monadische Arithmetiken zweiter Ordnung mit einem Sukzessor) *Gegeben sei eine Signatur $\Sigma = (C_\Sigma, V_\Sigma, typ_\Sigma)$, dann ist $\mathcal{T}_\Sigma^{S1S}$ die kleinste Menge aller Terme mit:*

- $0 \in \mathcal{T}_\Sigma^{S1S}$
- $x \in \mathcal{T}_\Sigma^{S1S}$ *für jede Variable* $x \in V_\Sigma$ *mit* $typ_\Sigma(x) = \mathbb{N}$

- *falls* $\tau \in \mathcal{T}_\Sigma^{S1S}$*, dann ist auch* $S(\tau) \in \mathcal{T}_\Sigma^{S1S}$

Die Menge aller Formeln $\mathcal{L}_\Sigma^{S1S}$ *bzgl.* Σ *sei wie folgt definiert:*

- *die booleschen Konstanten* **T** *und* **F** *sind Formeln*
- $P(\tau) \in \mathcal{L}_\Sigma^{S1S}$ *für jede Variable* $P \in V_\Sigma$ *mit* $typ_\Sigma(P) = \mathbb{N} \to \mathbb{B}$ *und jeden Term* $\tau \in \mathcal{T}_\Sigma^{S1S}$
- $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \to \psi$, $\varphi \leftrightarrow \psi \in \mathcal{L}_\Sigma^{S1S}$ *falls* $\varphi, \psi \in \mathcal{L}_\Sigma^{S1S}$
- $\forall t.\varphi \in \mathcal{L}_\Sigma^{S1S}$ *und* $\exists t.\varphi \in \mathcal{L}_\Sigma^{S1S}$ *für jede Variable* $t \in V_\Sigma$ *mit* $typ_\Sigma(x) = \mathbb{N}$ *und für jede Formel* $\varphi \in \mathcal{L}_\Sigma^{S1S}$
- $\forall P.\varphi \in \mathcal{L}_\Sigma^{S1S}$ *und* $\exists P.\varphi \in \mathcal{L}_\Sigma^{S1S}$ *für jede Variable* $P \in V_\Sigma$ *mit* $typ_\Sigma(P) = \mathbb{N} \to \mathbb{B}$ *und für jede Formel* $\varphi \in \mathcal{L}_\Sigma^{S1S}$

Zusätzlich zu den Regeln der oben angegebenen Definition kann man noch weitere Konstrukte angeben, die sich aus der oben angegebenen Definition ableiten lassen:

- alle natürlichen Zahlen $n \in \mathbb{N}$ lassen sich in S1S definieren, da aus $1 = S(0)$, $2 = S(1)$ $\ldots n = S^n(0)$ folgt.
- in S1S sind die Relationen $=$ und $<$ definiert als:

$$\begin{aligned}(x = y) &:= \forall P.P(x) \leftrightarrow P(y)\\ (x < y) &:= \forall P.[\forall t.P(t) \to P(S(t))] \to [P(S(x)) \to P(y)]\end{aligned}$$

  Aus diesen beiden Relationen lassen sich die weiteren Relationen: $\leq, \neq, \geq, >$ ableiten. Generell gilt, daß bei der Ersetzung einer Relation durch einen prädikatenlogischen Term mindestens ein neues Signal entsteht und damit auch mindestenss ein neuer Quantor über diesem Signal erforderlich ist. Zu jeder der hier genannten Relation gibt es zwei Varianten: eine Variante erzeugt einen Allquantor über dem neu hinzukommenden Signal und die andere Variante einen Existenzquantor. Diesem Umstand zufolge kann man die Variante wählen, die günstiger für den weiteren Verlauf der Transformation ist. Argumente, die für die eine oder andere Variante sprechen werden in der vorliegenden Arbeit später deutlich. In der nachfolgenden Tabelle sind alle Transformationsvorschriften für die angegebenen Relationen zusammengefaßt:

$$\begin{aligned}(x = y) &:= \forall P.P(x) \leftrightarrow P(y)\\ (x = y) &:= \exists PQ.\left(\begin{array}{l}[\forall t.P(t) \to P(S(t))] \wedge P(S(y)) \wedge \neg P(x) \wedge\\ [\forall t.Q(t) \to Q(S(t))] \wedge Q(S(x)) \wedge \neg Q(y)\end{array}\right)\end{aligned}$$

$$
\begin{array}{lcl}
(x < y) & := & \forall P.[\forall t.P(t) \rightarrow P(S(t))] \rightarrow [P(S(x)) \rightarrow P(y)] \\
(x < y) & := & \exists P.[\forall t.P(t) \rightarrow P(S(t))] \wedge \neg P(x) \wedge P(y)
\end{array}
$$

$$
\begin{array}{lcl}
(x \leq y) & := & \forall P.[\forall t.P(t) \rightarrow P(S(t))] \rightarrow [P(x) \rightarrow P(y)] \\
(x \leq y) & := & \exists P.[\forall t.P(t) \rightarrow P(S(t))] \wedge \neg P(x) \wedge P(S(y))
\end{array}
$$

$$
\begin{array}{lcl}
(x \neq y) & := & \forall PQ.\left(\begin{array}{l} [\forall t.P(t) \rightarrow P(S(t))] \rightarrow [P(S(y)) \rightarrow P(x)] \vee \\ [\forall t.Q(t) \rightarrow Q(S(t))] \rightarrow [Q(S(x)) \rightarrow Q(y)] \end{array} \right) \\
(x \neq y) & := & \exists P.P(x) \not\leftrightarrow P(y)
\end{array}
$$

$$
\begin{array}{lcl}
(x > y) & := & y < x \\
(x \geq y) & := & y \leq x
\end{array}
$$

- mit gewissen Einschränkungen ist sogar die Addition von numerischen Variablen möglich: für ein $P \in V_\Sigma$ mit $typ_\Sigma(P) = \mathbb{N} \rightarrow \mathbb{B}$ und $x_1, \ldots, x_n \in \mathcal{T}_\Sigma^{S1S}$ mit $typ(x_i) \in \mathbb{N}$ für $i \in \{1, \ldots, n\}$ ist $P(x_1 + \ldots + x_n) \in \mathcal{L}_\Sigma^{S1S}$. Die genannten Einschränkung besteht darin, daß ein beliebiges x_i nur in einer Summe vorkommen darf, bei der auch mindestenss alle x_j mit $(j < i)$ vertreten sind.

  Solch ein Konstrukt wird dann nach einer der folgenden Regeln sukzessive aufgelöst:

$$
\left[\forall x_n.\Phi\left(\sum_{j=1}^{n} x_j\right)\right] = \forall z.\sum_{j=1}^{n-1} x_j \leq z \rightarrow \Phi(z) := \Psi\left(\sum_{j=1}^{n-1} x_j\right)
$$

$$
\left[\exists x_n.\Phi\left(\sum_{j=1}^{n} x_j\right)\right] = \exists z.\sum_{i=1}^{n-1} x_j \leq z \wedge \Phi(z) := \Psi\left(\sum_{j=1}^{n-1} x_j\right)
$$

  usf.

2.5 Normalformen

Bei dem in Kapitel 3.1 vorgestellten Transformationsverfahren werden einige Normalformen von prädikatenlogischen Formeln benötigt. In diesem Kapitel werden die Pränexe Normalform und die Behmannsche Normalform vorgestellt.

Definition 16 (Pränexe Normalform) *Unter einer pränexen Normalform versteht man eine prädikatenlogische Formel, bei der alle Quantoren aus der Formel herausgezogen und vor die*

Formel gezogen werden. Es ensteht somit eine Folge von Quantoren, gefolgt von einer quantorenfreien Restformel.

Die Quantoren können nach folgenden Theoremen aus der Formel herausgezogen werden:

1.

$$\begin{aligned}[\Delta X.\mathcal{A}_1(X)] \vee [\Delta X.\mathcal{A}_2(X)] &= [\Delta X_1.\mathcal{A}_1(X_1)] \vee [\Delta X_2.\mathcal{A}_2(X_2)] \\ &= \Delta X_1.\Delta X_2.[\mathcal{A}_1(X_1) \vee \mathcal{A}_2(X_2)]\end{aligned}$$

2.

$$\begin{aligned}[\Delta X.\mathcal{A}_1(X)] \wedge [\Delta X.\mathcal{A}_2(X)] &= [\Delta X_1.\mathcal{A}_1(X_1)] \wedge [\Delta X_2.\mathcal{A}_2(X_2)] \\ &= \Delta X_1.\Delta X_2.[\mathcal{A}_1(X_1) \wedge \mathcal{A}_2(X_2)]\end{aligned}$$

$\mathcal{A}_1$ und $\mathcal{A}_2$ sind Terme, in denen die Variablen X, X_1, X_2 vorkommen und $\Delta \in \{\forall, \exists\}$.

Die Negation eines Quantors Δ transformiert diesen in sein Komplement. So wird aus einem Allquantor ein Existenzquantor und umgekehrt. Die übrigen boolescher Operatoren $\rightarrow$, $\leftrightarrow$ ... können aus den hier definierten Operatoren abgeleitet werden.

Sollte aus zwei Termen, die über eine Konjunktion verknüpft sind, jeweils ein Allquantor herausgezogen werden, dann kann einer der beiden Quantoren eliminiert und der Bindungsbereich des anderen Allquantors um diesen eliminierten Allquantor erweitert werden. Dies erfordert die Umbenennung der Variablen im Bindungsbereich des gestrichenen Quantors. Das analoge Verfahren kann auch bei einer disjunktiven Verknüpfung in Kombination mit Existenzquantoren durchgeführt werden. Die Theoreme 3 und 4 illustrieren diesen Tatbestand. In Theorem 3 werden zwei Allquantoren herausgezogen und die Variable X_2 in X_1 umbenannt. In Theorem 4 werden zwei Existenzquantoren herausgezogen und ebenso die Variable X_2 in X_1 umbenannt. Durch solch ein Eliminieren von Variablen kann eine Optimierung der Formel erzielt werden.

3. $[\forall X_1.\mathcal{A}_1(X_1)] \wedge [\forall X_2.\mathcal{A}_2(X_2)] = \forall X_1.[\mathcal{A}_1(X_1) \wedge \mathcal{A}_2(X_1)]$

4. $[\exists X_1.\mathcal{A}_1(X_1)] \vee [\exists X_2.\mathcal{A}_2(X_2)] = \exists X_1.[\mathcal{A}_1(X_1) \vee \mathcal{A}_2(X_1)]$

Wenn ein Quantor aus der Formel herausgezogen wird, muß sichergestellt sein, daß der Name der Variablen, die an diesen Quantor gebunden ist, nicht noch ein weiteres Mal in der Formel vorkommt. Ist das der Fall, dann muß die Variable wie in Theorem 1 und 2 umbenannt werden. In diesen beiden Theoremen kommt der Name X jeweils in beiden Teiltermen vor und muß demnach umbenannt werden. X_1 und X_2 dürfen weder in $\mathcal{A}_1$ noch in $\mathcal{A}_2$ vorkommen.

Definition 17 (Behmannsche Normalform [16] *Die Behmannsche Normalform ist eine Formel der Form:*

$$\Delta_1 P_1 \ldots \Delta_m P_m. \quad \varphi$$

wobei $\Delta_i \in \{\forall, \exists\}$*,* P_i *Prädikatsvariablen mit* $typ_\Sigma(P_i) = \mathbb{N} \to \mathbb{B}$ *und* φ *eine Restformel ist, die keine Quantoren über Prädikatsvariablen enthält.* φ *ist eine booleschen Kombination aus folgenden Termen:*

1. $\forall x.\varphi$ *mit* φ *quantorenfrei,* $x \in \mathbb{N}$
2. $\exists x.\varphi$ *mit* φ *quantorenfrei,* $x \in \mathbb{N}$
3. φ *mit* φ *quantorenfrei*

φ hat die Eigenschaft, daß sich die Bindungsbereiche der Quantoren über numerischen Variablen, die in φ enthalten sind, nicht überschneiden. φ kann folglich in disjunkte Bindungsbereiche von Quantoren über numerische Variablen zerlegt werden.

Die Behmannsche Normalform spielt bei dem Entscheidungsverfahren von S1S eine entscheidende Rolle und läßt sich bei S1S aufgrund der Einschränkung auf monadische Prädikate herstellen. Wie bereits in der Definition erwähnt, besteht die Restformel aus einer booleschen Verknüpfung aus sich nicht überlappenden Bindungsbereichen von Quantoren über numerischen Variablen. Falls ein Prädikat von mehr als einer numerischen Variablen abhängen würde, könnte diese Bedingung nicht mehr erfüllt werden.

Die Behmannsche Normalform geht aus einer pränexen Normalform hervor, indem die Quantoren über numerischen Variablen erst mal nach Typen sortiert (siehe Kapitel 3.1) und dann wieder in die quantorenfreie Restformel der pränexen Normalform zurückgeschoben werden. Das genaue Verfahren zur Berechnung der Behmannschen Normalform wird in Kapitel 3 angegeben. Außerdem wird dort auch noch näher auf die Bedeutung dieser Normalform für die Entscheidung von S1S eingegangen.

Kapitel 3

Entscheidungsverfahren nach Büchi

Büchi konnte bereits in den sechziger Jahren zeigen, daß man zum 'Leerheitstest' einer Sprache aus unendlichen Wörtern nur endlich viele Sequenzen überprüfen muß.

Büchi war der erste, der sich mit dieser Problematik beschäftigt hat. Weitere Personen, die ebenfalls auf dem Gebiet endlicher Automaten auf unendlichen Sequenzen bekannt wurden sind u.a. McNaughton [30], Rabin [27] und Muller [11]. Auch sie entwickelten Automaten, die trotz endlich vieler Zustände unendlich lange Sequenzen akzeptieren können. Der Automat, der von Büchi entwickelt wurde, der sogenannte Büchiautomat, wurde bereits in Kapitel 2.2.1 vorgestellt.

Büchi konnte sogar zeigen, daß endliche Automaten für bestimmte monadische Arithmetiken zweiter Ordnung eine Normalform darstellen [18]. Darauf basiert sein Entscheidungsverfahren für die bereits in Kapitel 2.4 vorgestellte monadischen Arithmetik zweiter Ordnung mit einem Sukzessor (S1S). Das Verfahren beruht auf der Überführung von S1S auf Büchiautomaten, die ja bekanntlich entscheidbar sind.

3.1 Büchi's Entscheidungsverfahren

Das von Büchi entwickelte Verfahren dient der Entscheidung von S1S-Formeln basiert auf Σ_n^ω-Formeln:

Definition 18 (Σ_n^ω-Formeln) *Die Menge der Σ_n^ω-Formeln wird rekursiv wie folgt definiert:*

1. *Σ_1^ω ist die Menge der indeterministischen Büchiformeln wie sie in Definition 6 auf Seite 25 definiert wurden.*
2. *Σ_{2n}^ω-Formeln ist die Menge aller Formeln*

$$\forall \vec{x}.\varphi, \text{ mit } typ(x_i) = \mathbb{N} \to \mathbb{B} \text{ und } \varphi \in \Sigma_{2n-1}^\omega.$$

3. Σ^{ω}_{2n+1}*-Formeln ist die Menge aller Formeln*

$$\exists \vec{x}.\varphi, \textit{ mit } typ(x_i) = \mathbb{N} \rightarrow \mathbb{B} \textit{ und } \varphi \in \Sigma^{\omega}_{2n}.$$

Bei Σ^{ω}_{1}-Formel handelt es sich um indeterministische Büchiformeln, ohne Wechsel von Quantoren über Prädikaten, wogegen eine Σ^{ω}_{n}-Formel $(n-1)$ Wechsel von Quantoren über Prädikaten besitzt.

Das Verfahren kann grob in vier Teile unterteilt werden:

- In einem ersten Schritt wird die S1S-Formel in eine Behmannsche Normalform überführt.
- Im zweiten Schritt wird dann diese Behmannsche Normalform in eine Σ^{ω}_{n}-Formel transformiert.
- Im nächsten Schritt wird dann diese Σ^{ω}_{n}-Formel durch n-fache Komplementierung in eine Σ^{ω}_{1}-Formel, also eine Büchi-Formel transformiert.
- Der nun zugrundeliegende Büchiautomat kann dann im letzten Schritt entschieden werden.

Im dritten Teilschritt des Entscheidungsverfahrens müssen diese Σ^{ω}_{n}-Formeln in Σ^{ω}_{1}-Formeln transformiert werden. Dies beruht auf folgendem Satz:

Satz 1 *Zu jeder* Σ^{ω}_{n}*-Formel existiert eine äquivalente* Σ^{ω}_{1}*-Formel.*

Beweis: *Induktionsanfang:* Aus $\varphi \in \Sigma^{\omega}_{1}$ folgt durch die Abgeschlossenheit von Büchiformeln bzgl. der Negation [20], daß auch $\neg\varphi \in \Sigma^{\omega}_{1}$ gilt. Das heißt aber laut Definition der Σ^{ω}_{1}, daß auch $\exists Q.\neg\varphi \in \Sigma^{\omega}_{1}$ gilt.

Jetzt kann wiederum aus der Abgeschlossenheit bzgl. des Komplements gefolgert werden, daß auch $\neg\exists Q.\neg\varphi \in \Sigma^{\omega}_{1}$, also auch $\forall Q.\varphi \in \Sigma^{\omega}_{1}$ gilt. Hieraus folgt $\Sigma^{\omega}_{2} = \Sigma^{\omega}_{1}$.

Der *Induktionsschluß* folgt analog. ■

Theorem 2 (Σ^{ω}_{n}**-Normalformen)** *Jede Formel der monadischen Arithmetik zweiter Ordnung mit einem Sukzessor, in der alle numerischen Variablen gebunden sind, ist zu einer Formel* φ *aus* Σ^{ω}_{n} *äquivalent, in der dieselben Prädikatsvariablen frei vorkommen.*

Beweis: Der Beweis erfolgt konstruktiv in acht Schritten und folgt der Darstellung in [9]. Dieser konstruktive Beweis entspricht den ersten zwei Schritten der oben genannten Unterteilung des Entscheidungsverfahrens. Bild 3.1 illustriert den ersten Teil, also die ersten fünf Transformationsschritte dieses konstruktiven Beweises von den S1S-Formeln bis zu der Erstellung der Behmannschen Normalform.

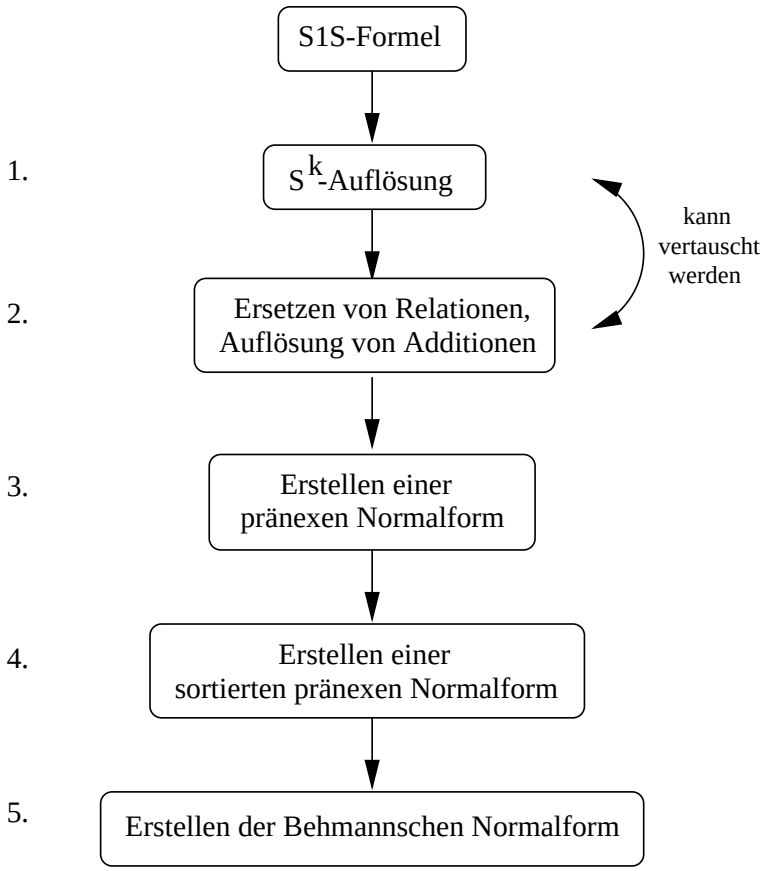

Abbildung 3.1: Erster Teil des Transformationsverfahrens

Die ersten beiden Transformationsschritte dienen der Entfernung von syntaktischem Ballast. Die Schritte drei bis fünf überführen die vereinfachte Formel in eine Behmannsche Normalform. Im Detail sehen diese Schritte wie folgt aus:

1. **Eliminierung mehrfach angewandter Sukzessoren**
   Zuerst werden alle Vorkommen von mehrfach angewandten SUC, also $\mathsf{SUC}^n(x)$ mit $n > 1$ nach einem der beiden folgenden Theoreme in einfache SUC umgewandelt:
   (a)
   $$A(\mathsf{SUC}^{n+1}(x)) \equiv \exists x_1 \ldots \exists x_n.\ x_1 = \mathsf{SUC}\,(x) \wedge \left(\bigwedge_{i=1}^{n-1} x_{i+1} = \mathsf{SUC}\,(x_i)\right) \wedge A(\mathsf{SUC}\,(x_n))$$
   (b)
   $$A(\mathsf{SUC}^{n+1}(x)) \equiv \forall x_1 \ldots \forall x_n.\ x_1 = \mathsf{SUC}\,(x) \wedge \left(\bigwedge_{i=1}^{n-1} x_{i+1} = \mathsf{SUC}\,(x_i)\right) \rightarrow A(\mathsf{SUC}\,(x_n))$$

Hierbei steht A für eine prädikatenlogische Formel, x ist eine numerische Variable, welche in A frei vorkommt, x_i sind numerische Variablen und n ist eine ganze Zahl.

Die Eliminierung von $\mathsf{SUC}^n(x)$ mit $n > 1$ ist bereits an dieser Stelle notwendig, da in Transitionsrelationen des Zielautomaten nur die Verwendung von $\mathsf{X}P$ erlaubt ist. Der hier angegebene X-Operator entspricht der Modifikation eines Argumentes einer Prädikatsvariablen durch ein einfaches SUC.

Numerische Konstanten größer als Eins werden an dieser Stelle ebenfalls eliminiert, da diese Nachfolger von Null sind und auch durch mehrfache Anwendung von SUC entstehen.

2. **Ersetzen von Relationen und anderen Erweiterungen**
Im zweiten Schritt werden Relationen innerhalb der Formel durch ihre Definitionen ersetzt. Dabei steht P wieder für eine Prädikatsvariable:

$$\begin{aligned}(x = y) &:= (\forall P.P(x) \leftrightarrow P(y)) \\ (x < y) &:= (\exists P.P(x) \wedge \neg P(y) \wedge \forall z.P(\mathsf{SUC}\ (z)) \rightarrow P(z))\end{aligned}$$

An diese Stelle werden nur exemplarisch die Relationen '$<$' und '$=$' behandelt. Die übrigen Vergleiche numerischer Variablen, also '$>$, etc. können aus diesen beiden Relationen abgeleitet werden (siehe auch Kapitel 2.4).

Neben der Ersetzung von Relationen können auch noch andere Erweiterungen von S1S durch ihre Definition ersetzt werden. In Kapitel 2.4 werden beispielsweise noch Additionen unter gewissen Einschränkungen unterstützt. Außerdem unterliegen sie noch weiteren Auflagen bzgl. ihrer Einsatzmöglichkeiten. Das folgende Beispiel zeigt ein Theorem zur Ersetzung einer Summe bestehend aus zwei Summanden:

$$(\forall t_1.\exists t_2.P(t_1 + t_2)) \quad = \quad (\forall t_1.\exists t_2.t_1 \leq t_2 \rightarrow P(t_2))$$

Auch die Konstante Null kann in diesem Schritt nach folgendem Theorem ersetzt werden:

$$P(0) = \exists x.P(x) \wedge \forall y.x \leq y$$

Nach Ausführung dieses Transformationsschrittes, treten in der resultierenden Formel die numerischen Variablen ausschließlich als Argumente von Prädikaten bzw. der Funktion SUC auf.

Für den Fall, daß bei der Auflösung von S^k mit $k > 1$ die hierbei entstehenden Relationen eigenständig ersetzt werden, können diese ersten beiden Schritte in beliebiger Reihenfolge ausgeführt werden.

3. **Überführung in pränexe Normalform**
   Die Formel wird nun in eine pränexe Normalform gebracht, d.h. alle Quantoren die in der Formel vorkommen, werden in diesem Schritt aus der Formel hinausgeschoben. Die Formel kann danach in eine Liste von Quantoren und in eine quantorenfreie Restformel zerlegt werden:
   $$\Delta_1 X_1 \ldots \Delta_n X_n. \quad \varphi(\vec{I}, \vec{Q}, \vec{x})$$
   wobei $\Delta_j \in \{\forall, \exists\}$ gilt und φ ein boolescher Term ist, der von freien Prädikatsvariablen $\vec{I}$, gebundenen Prädikatsvariablen $\vec{Q}$ und gebundenen numerischen Variablen $\vec{x}$ abhängt, also $\{Q_1 \ldots\ Q_n\} \cup \{x_1 \ldots x_m\} \subset \{X_1 \ldots X_l\}$.

4. **Überführung in sortierte pränexe Normalform**
   Die Quantoren, die im letzten Schritt aus der Formel herausgeschoben wurden, werden nun anhand der folgenden vier Theoreme [7] in eine Reihenfolge gebracht, bei der die Quantoren über Prädikatsvariablen den Quantoren über numerischen Variablen voranstehen:

   (a) $(\exists x.\exists P.\varphi(P,x)) \;=\; (\exists P.\exists x.\varphi(P,x))$
   (b) $(\forall x.\forall P.\varphi(P,x)) \;=\; (\forall P.\forall x.\varphi(P,x))$
   (c) $(\exists x.\forall P.\varphi(P,x)) \;=\; (\exists Q.\forall P.\forall x.\exists z.[Q(x) \rightarrow \varphi(P,x)] \wedge Q(z))$
   (d) $(\forall x.\exists P.\varphi(P,x)) \;=\; (\forall Q.\exists P.\exists x.\forall z.Q(z) \rightarrow \varphi(P,x) \wedge Q(x))$

   In den beiden letzten Fällen wird jeweils eine neue Prädikatsvariable Q und eine neue numerische Variable z erzeugt. Diese dürfen nicht bereits in der Formel φ vorkommen.

5. **Transformation in Behmannsche Normalform**
   Die bisherigen Schritte führen zu sogenannten 'sortierten pränexen Normalformen' der folgenden Bauart:
   $$\Delta_1 Q_1 \ldots \Delta_m Q_m.\Delta_{m+1} x_1 \ldots \Delta_{m+n} x_n.\varphi(\vec{I}, \vec{Q}, \vec{x}),$$
   wobei $\Delta_j \in \{\forall, \exists\}$ gilt und φ ein boolescher Term ist, der von Prädikatsvariablen $\vec{I}$ und $\vec{Q}$ sowie von den gebundenen numerischen Variablen $\vec{x}$ abhängt.

   Ziel dieses fünften Transformationsschrittes ist es, wie bereits genannt, die numerischen Quantoren wieder zurück in die Formel zu schieben. Dieses Zurückschieben geschieht nach mehrfacher Anwendung der folgenden vier Theoreme:

   (a) $[\forall t.P(t) \wedge Q(t)] \;=\; [\forall t.P(t)] \wedge [\forall t.Q(t)]$
   (b) $[\forall t.P(t) \vee S] \;=\; [\forall t.P(t)] \vee S$
   (c) $[\exists t.P(t) \vee Q(t)] \;=\; [\exists t.P(t)] \vee [\exists t.Q(t)]$
   (d) $[\exists t.P(t) \wedge S] \;=\; [\exists t.P(t)] \wedge S$

wobei $P(t)$ und $Q(t)$ Terme sind, die die Variable t enthalten und S ein Term ist, der entweder konstant ist, oder aber nicht von der Variablen t abhängt.

Ein Quantor einer numerischen Variablen t darf erst zurückgeschoben werden, wenn eine Zerlegung in ausschließlich von t abhängige und von t unabhängige Terme erreicht werden kann. An dieser Stelle wird die Voraussetzung benötigt, daß es sich bei den Prädikaten in S1S um **monadische Prädikate** handeln muß, d.h. daß die Prädikate von höchstens einer numerischen Variablen abhängen dürfen. Sollten die Prädikate von mehr als einer numerischen Variablen abhängen, wäre eine derartige Zerlegung nicht möglich.

Solch eine Zerlegung kann erreicht werden, wenn sich die Formel im Falle eines Allquantors $\Delta_{m+n} = \forall$ in einer konjunktiven Normalform und im Falle eines Existenzquantors in einer disjunktiven Normalform befindet. Sollte demnach ein Allquantor geschoben werden, muß die Kernformel vorher in eine konjunktive Normalform und bei einem Existenzquantor in eine disjunktive Normalform gebracht werden.

Das Ergebnis dieses Schrittes ist dann eine Formel der Bauart:

$$\Delta_1 Q_1 \ldots . \Delta_m Q_m . \bigvee_{j=1}^{a} \left(\begin{array}{l} \mathcal{E}_j[\vec{I}^{(0)}, \vec{Q}^{(0)}] \wedge \\ \left(\forall t . \mathcal{G}_j[\vec{I}^{(t)}, \vec{Q}^{(t)}, \vec{Q}^{(\mathsf{SUC}\ (t))}]\right) \wedge \\ \left(\bigwedge_{k=1}^{b_j} \exists t . \mathcal{H}_{j,k}[\vec{I}^{(t)}, \vec{Q}^{(t)}, \vec{Q}^{(\mathsf{SUC}\ (t))}] \right) \end{array} \right)$$

$\mathcal{E}$, $\mathcal{G}_j$ und $\mathcal{H}_{j,k}$ sind hierbei quantorenfreie aussagenlogische Formeln. Diese Behmannsche Normalform kann in eine indeterministische Präfixformel transformiert werden, indem man sie folgender Transformationsvorschrift unterzieht:

function Behmann2LTL
CASE φ, ψ OF

$P^{S^n(0)}$, $P^{S^n(t)}$: return $\mathsf{X}^n P$

$\varphi \wedge \psi$: $Behmann2LTL(\varphi) \wedge Behmann2LTL(\psi)$

$\varphi \vee \psi$: $Behmann2LTL(\varphi) \vee Behmann2LTL(\psi)$

$\neg\varphi$: $\neg Behmann2LTL(\varphi)$

$\forall t.\varphi$: $\mathsf{G}\ Behmann2LTL(\varphi)$ für $typ(t) = \mathbb{N}$

$\exists t.\varphi$: $\mathsf{F}\ Behmann2LTL(\varphi)$ für $typ(t) = \mathbb{N}$

$\exists P.\varphi$: $\exists P.Behmann2LTL(\varphi)$

$\exists P.\varphi$: $\exists P.Behmann2LTL(\varphi)$

Satz 2 *Sei $\varphi \in \mathcal{L}_\Sigma^{S1S}$ in Behmannscher Normalform. Dann ist $Behmann2LTL(\varphi) \in \mathcal{L}_\Sigma^{S1S}$ mit*

$$\models \varphi = Behmann2LTL(\varphi)$$

Nachdem die Behmannsche Normalform derart transformiert wurde erhält man folgende quantifizierte LTL-Formel:

$$\Delta_1 Q_1 \ldots \Delta_m Q_m. \bigvee_{j=1}^{a} \underbrace{\mathcal{E}_j[\vec{I},\vec{Q}]}_{\mathcal{I}} \wedge \underbrace{\left(\mathsf{G}\left(\mathcal{G}_j[\vec{I},\vec{Q},\mathsf{X}\vec{Q}]\right)\right)}_{\mathsf{G}\mathcal{T}} \wedge \underbrace{\left(\bigwedge_{k=1}^{b_j} \mathsf{F}\left(\mathcal{H}_{j,k}[\vec{I},\vec{Q},\mathsf{X}\vec{Q}]\right)\right)}_{Lebendigkeitsbedingung}$$

In der obigen Formel sind die Disjunktionsterme fast schon ω-Automaten: sie enthalten bereits eine Initialformel $\mathcal{E}[\vec{I},\vec{Q}]$, zum anderen existiert auch eine Übergangstransition $\mathsf{G}\,\mathcal{G}_j[\vec{I},\vec{Q},\mathsf{X}\vec{Q}]$. Der dritte Teil

$$\left(\bigwedge_{k=1}^{b_j} \mathsf{F}\left(\mathcal{H}_{j,k}[\vec{I},\vec{Q},\mathsf{X}\vec{Q}]\right)\right)$$

kann als Akzeptanzbedingung eines ω-Automaten aufgefaßt werden. Derartige Automatenkonzepte gibt es allerdings nicht. Eine Reduktion auf Σ_n^ω-Formeln erfolgt in den nächsten Schritten:

Der zweite Teil dieses Transformationsverfahrens überführt demnach die QLTL-Formel in eine Σ_n^ω-Formeln. Bild 3.2 zeigt den weiteren Verlauf des Transformationsverfahrens.

6. **Erzeugen der Akzeptanzbedingung**
   In dem nun folgenden Schritt ersetzt man die Akzeptanzbedingungen, also die Konjunktion der mittels des F-Operators gebundenen Terme $\left(\bigwedge_{k=1}^{b} \mathsf{F}\left(\mathcal{H}_{j,k}[\vec{I},\vec{Q}]\right)\right)$ mit Hilfe des folgenden Theorems durch eine Akzeptanzbedingung in Form einer GF-Formel:

$$\left(\bigwedge_{j=1}^{m} \mathsf{F}a_j\right) \equiv \left(\begin{array}{l} \exists \vec{P}. \left[\bigwedge_{j=1}^{m} (P_j = \mathsf{F}) \wedge (\mathsf{G}\,(\mathsf{X}P_j \leftrightarrow P_j \vee a_j))\right] \wedge \\ \qquad \left[\mathsf{GF}\left(\bigwedge_{j=1}^{m} P_j\right)\right] \end{array}\right)$$

Bei den neu hinzugekommenen Prädikaten P_j handelt es sich um sogenannte Wächterelemente. Diese Prädikate werden mit F initialisiert und beobachten die Gültigkeit der a_j. Wenn a_j zum ersten mal Eins wird, dann werden die P_j ebenfalls Eins und behalten diesen Wert bei. Nachdem alle a_j einmal Eins waren, bleibt $\bigwedge_{k=1}^{b_j} P_k$ auf Eins. Damit gilt sogar $\mathsf{FG}\left(\bigwedge_{k=1}^{b_j} P_k\right)$, erst recht aber $\mathsf{GF}\left(\bigwedge_{k=1}^{b_j} P_k\right)$.

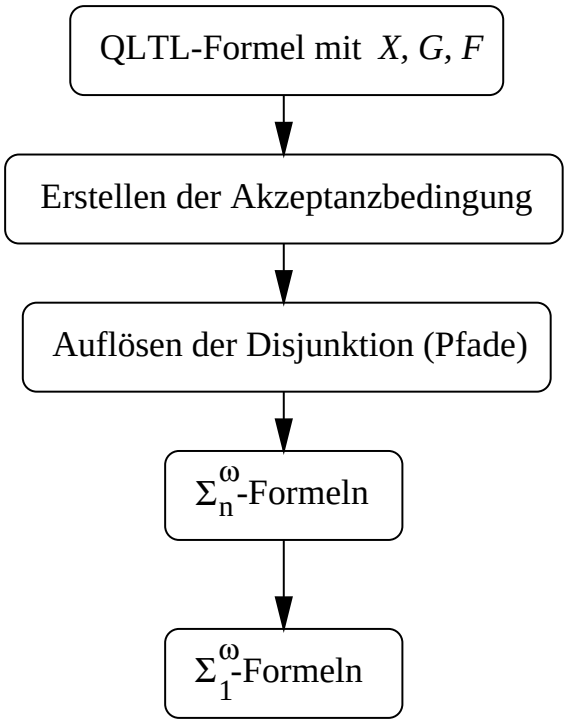

Abbildung 3.2: Zweiter Teil des Transformationsverfahrens

Da es sich bei diesen Prädikaten und bei ihren Konjunktionen um monoton steigende Prädikate handelt, ist es nicht notwendig ein GF-Konstrukt einzuführen, sondern es genügt $\mathsf{F}\left(\bigwedge_{k=1}^{b_j} P_k\right)$ oder $\mathsf{FG}\left(\bigwedge_{k=1}^{b_j} P_k\right)$ Dies wird später verwendet.

Nach der Anwendung dieses Theorems, schiebt man die neu eingeführten Quantoren über $\vec{P}$ wieder nach außen. Das Ergebnis dieses Schrittes sind nun Formeln, welche das folgende Aussehen haben:

$$\Delta_1 Q_1 \ldots . \Delta_m Q_m . \exists \vec{P}. \bigvee_{j=1}^{a} \left(\begin{array}{l} \mathcal{E}_j[\vec{I},\vec{Q}] \wedge \bigwedge_{k=1}^{b_j} \neg P_k \wedge \\ \mathsf{G}\left(\mathcal{G}_j[\vec{I},\vec{Q},\mathsf{X}\vec{Q}] \wedge \bigwedge_{k=1}^{b_j} \left[\mathsf{X}P_k \leftrightarrow P_k \vee \mathcal{H}_{j,k}[\vec{I},\vec{Q},\mathsf{X}\vec{Q}]\right]\right) \wedge \\ \mathsf{GF}\left(\bigwedge_{k=1}^{b_j} P_k\right) \end{array} \right)$$

$\mathcal{E}_j$, $\mathcal{G}_j$ und $\mathcal{H}_{j,k}$ sind hierbei ebenfalls quantorenfreie aussagenlogische Terme.

7. **Eliminierung der Disjunktion**
   In diesem Schritt wird nun die auf die Quantoren folgende Disjunktion eliminiert. Dies erfolgt durch mehrmalige Anwendung des folgenden Theorems:

$$\left(\begin{array}{l}\left(\begin{array}{l}\mathcal{A}_1\\ \wedge[\mathsf{G}\,(\mathcal{B}_1)]\\ \wedge[\mathsf{GF}\,(\mathcal{C}_1)]\end{array}\right)\vee\\ \left(\begin{array}{l}\mathcal{A}_2\\ \wedge[\mathsf{G}\,(\mathcal{B}_2))]\\ \wedge[\mathsf{GF}\,(\mathcal{C}_2))]\end{array}\right)\end{array}\right)\equiv\left(\begin{array}{l}\exists P.\\ \quad(P\Rightarrow\mathcal{A}_1|\,\mathcal{A}_2)\wedge\\ \quad\left[\begin{array}{l}\mathsf{G}\,(\mathsf{X}P\leftrightarrow P)\wedge\\ \quad(P\Rightarrow\mathcal{B}_1|\,\mathcal{B}_2)\end{array}\right]\wedge\\ \quad[\mathsf{GF}\,((P\Rightarrow\mathcal{C}_1|\,\mathcal{C}_2))]\end{array}\right)$$

Die neu eingeführten Quantoren über Prädikaten schiebt man wiederum nach außen.

8. Als Ergebnis der Transformation erhält man nun folgende Formeln:

$$\Delta_1 Q_1 \ldots . \Delta_m Q_m . \bigvee_{j=1}^{a} \mathcal{E}'_j[\vec{I},\vec{Q}] \wedge \left(\mathsf{G}\left(\mathcal{G}'_j[\vec{I},\vec{Q},\mathsf{X}\vec{Q}]\right)\right) \wedge \mathsf{GF}\left(\mathcal{H}[\vec{Q}]\right)$$

Bei diesen Formeln handelt es sich um Σ_n^ω-Formeln, wobei n die Anzahl der Blöcke von gleichartigen Quantoren über Prädikatsvariablen angibt.

Wie bereits zu Beginn dieses Kapitels gezeigt, ist es möglich, Σ_n^ω-Formeln auf Σ_1^ω-Formeln abzubilden. Folglich kann man alle Formeln der monadischen Sukzessor-Arithmetik zweiter Ordnung auf Büchiformeln abbilden.

■

3.2 Komplementbildung

In [20] wird gezeigt, daß einige temporale Logiken gleich mächtig sind wie Büchiautomaten, d.h. sie akzeptieren die gleiche Sprache, also die gleiche Menge von ω-Worten. Um die Erfüllbarkeit dieser Logiken zu überprüfen, kann man demnach einen Büchiautomaten erzeugen, der genau die Sprache akzeptiert, die der logischen Formel entspricht.

Bei der Transformation von Σ_n^ω-Formeln auf Σ_1^ω-Formeln muß die Komplementierung dieser Formeln durchgeführt werden. Da es sich bei diesen Formeln um indeterministische Formeln handelt, hat die Komplementierung einen Aufwand von $O(2^{nlog(n)})$ [33]. Das in diesem Kapitel vorgestellte Komplementierungsverfahren stammt aus [5] und hat einen Aufwand von (16^{n^2}).

Sei $\mathcal{A}$ ein Büchiautomat, der die Sprache $\mathcal{L}_\omega(\mathcal{A})$ akzeptiert, dann wird ein Büchiautomat $\bar{\mathcal{A}}$ gesucht, der die Sprache $\mathcal{L}_\omega(\bar{\mathcal{A}}) = \Sigma^\omega - \mathcal{L}_\omega(\mathcal{A})$ akzeptiert.

Da die Klasse der ω-regulären Sprachen unter Komplementbildung, Vereinigung und Schnitt abgeschlossen sind, sind auch Büchiautomaten unter diesen Operationen abgeschlossen, d.h. nichtdeterministische Büchiautomaten besitzen ein Komplement.

Um einen deterministischen Automaten zu komplementieren genügt es die Finalzustände zu komplementieren, d.h. man ersetze die Finalzustände Q_F durch $Q - Q_F$. Bei deterministischen Automaten, also auch bei den deterministischen Büchiautomaten ist dies nicht weiter schwierig. Problematisch wird die Komplementierung jedoch bei nichtdeterministischen Büchiautomaten. Zur Komplementbildung müssen alle möglichen Sequenzen betrachtet werden, die für zwei beliebige Zustandspaare mit dem ersten Zustand beginnen und im zweiten Zustand enden. Dies hat zur Folge, daß ein Automat $\mathcal{A}$ mit n Zuständen nach der Komplementierung bis zu (16^{n^2}) Zustände haben kann [5].

Da nichtdeterministische Büchiautomaten mächtiger sind als deterministische, ist eine direkte Überführung nicht möglich. Daher ist es notwendig eine Familie deterministischer Büchiautomaten zu erzeugen, die dem nichtdeterministischen Automaten entsprechen.

Sei $\mathcal{A} = (Q, \Sigma, \delta, q_0, Q_F)$ ein Büchiautomat. Zuerst wird eine Familie deterministischer Automaten $\{\tilde{\mathcal{A}}_i\}$ auf endliche Wörter gebildet, die dasselbe Verhalten wie $\mathcal{A}$ haben. Dabei ist folgendes Verhalten gefragt: gegeben sei ein endliches nichtleeres Wort α und zwei Zustände $u, v \in Q$.

1. gibt es einen Durchlauf durch $\mathcal{A}$ mit Eingabe α von u nach v
2. gibt es einen Durchlauf durch $\mathcal{A}$ mit Eingabe α von u nach v und werden Zustände aus Q_F durchlaufen

Die folgende Konstruktion kann als eine Generalisierung der 'Rabin und Scott's subset construction' [5] angesehen werden.

- Sei $Q = \{q_1, \ldots, q_n\}$ eine Menge von Zuständen aus $\mathcal{A}$.
- Definiere $Q' := Q \times \{0, 1\}$ und $Q^* := (2^{Q'})^n$.
  Q^* hat m Zustände bezeichnet mit p_1 bis p_m, wobei $m = 4^{n^2}$.

  Q^* bezeichnet ein n-Tupel von Zustandsmengen aus Q, die zusätzlich mit 0 oder 1 kodiert sind, je nachdem, ob ein Endzustand im Durchlauf enthalten ist, oder nicht. Man benötigt ein n-Tuppel, da der Startzustand nicht vorgeschrieben ist und jeder Zustand als Startzustand benutzt werden kann.
- Die Zustandsmenge des Automaten $\{\tilde{\mathcal{A}}_i\}$ ist demnach $\tilde{Q} = Q^* \cup \{p_0\}$ wobei p_0 ein spezieller Startzustand ist.

Die deterministische Übergangstransition $\tilde{\delta} : \tilde{Q} \times \Sigma \to Q^*$ ist jetzt folgendermaßen definiert:

- $< Q_1, \ldots, Q_n >= \tilde{\delta}(p_0, \sigma)$ gdw.

$$Q_i = \{< u, 0 >: u \in \delta(q_i, \sigma)\} \cup \{< u, 1 >: u \in \delta(q_i, \sigma) \cap Q_F\}$$

- $< Q_1, \ldots, Q_n >= \tilde{\delta}(< T_1, \ldots, T_n >, \sigma)$ gdw.

$$Q_i = \{< u, 0 >: u \in \delta(v, \sigma) \text{ für einige } < v, j > \in T_i\}$$
$$\cup\{< u, 1 >: u \in \delta(v, \sigma) \text{ für einige } < v, 1 > \in T_i\}$$
$$\cup\{< u, 1 >: u \in \delta(v, \sigma) \cap Q_F \text{ für einige } < v, j > \in T_i\}$$

Jetzt kann man die deterministischen Automaten $\tilde{A}_i$, mit $1 \leq i \leq m$, auf endlichen Wörtern

$$\tilde{A}_i = (\tilde{Q}, \Sigma, \tilde{\delta}, p_0, \{p_i\})$$

definieren. Diese Automaten akzeptieren dann endliche Wörter X_i, die Partitionen von Σ^+ sind.

Die soeben beschriebene Konstruktion kann nun dazu benutzt werden, um Büchiautomaten zu komplementieren. Hierzu definiert man sich eine Sprache: $Y_{ij} = X_i X_j^\omega$ mit $1 \leq i, j \leq m$. Falls die folgenden Bedingungen

- $X_i X_i \subseteq X_i$
- $X_j X_j \subseteq X_j$

erfüllt sind, ist Y_{ij} passend und es gelten folgende Aussagen:

- $\Sigma^\omega = \bigcup \{Y_{ij} : Y_{ij} \text{ ist passend}\}$
- Für $1 \leq i, j \leq m$ gilt:

entweder $L_\omega(A) \cap Y_{ij} = \emptyset$
oder $L_\omega(A) \cap Y_{ij} = Y_{ij}$

- $\bar{L_\omega}(A) = \bigcup \{Y_{ij} : Y_{ij} \cap L_\omega(A) = \emptyset \text{ und } Y_{ij} \text{ ist passend}\}$

Die Beweise zu diesen Behauptungen findet man in [5].

Abschließend kann man sagen, daß es möglich ist einen Büchiautomaten mit n Zuständen zu komplementieren, wobei den komplementäre Automat bis zu $O(16^{n^2})$ Zustände hat.

Kapitel 4

Neues Entscheidungsverfahren zur Erfüllbarkeit von S1S-Formeln

Das Entscheidungsverfahren, das in diesem Kapitel vorgestellt wird, ist an das in Kapitel 3 besprochene Entscheidungsverfahren von Büchi angelehnt. Demnach ist die Sprache, von der ausgegangen wird, ebenfalls die in Kapitel 2.4 definierte monadische Arithmetik mit einem Sukzessor. Büchi hat mit seinem Entscheidungsverfahren in erster Linie ein theoretisches Resultat angestrebt, bei dem es ihm auf die Entscheidbarkeit von S1S an sich ankam. Sein Verfahren war zunächst nicht für eine Implementierung bestimmt, und somit wurden auch keine Vorkehrungen getroffen, einen bezüglich Laufzeit und Speicherverbrauch effizienten Algorithmus zu entwickeln.

In der vorliegenden Arbeit werden einige Punkte vorgeschlagen, um Büchis Verfahren hinsichtlich der Effizienzverbesserung zu modifizieren. Um die geforderte Verbesserung der Effizienz erreichen zu können, ist es notwendig von dem ursprünglichen Transformationsalgorithmus abzuweichen. Dies äußert sich beispielsweise in der Verwendung neuer Datenstrukturen wie BDDs oder aber in der Neuentwicklung bzw. Abwandlung von Verfahrensschritten. Die verbesserte Effizienz der optimierten Version des Originalverfahrens ist im wesentlichen auf die folgenden Punkte zurückzuführen:

1. Es wird strengstens zwischen Übergangstransitionen und Sicherheitsbedingungen unterschieden.

2. Die Zielautomaten sind im Gegensatz zu dem Verfahren von Büchi keine indeterministischen Büchiautomaten, sondern deterministische Präfixformeln, um zumindest eine Komplementbildung zu erleichtern.

Durch die Vermeidung indeterministischer Automaten und durch eine gezielte Einschränkung von S1S kann ein deterministisch exponentieller Aufwand für die Laufzeit erreicht werden.

Zum Abschluß dieses Kapitels und in Kapitel 5 wird auf die dadurch notwendige Einschränkung der S1S genauer eingegangen und es wird auch eine Begründung geliefert, warum das Verfahren durch die Einschränkung von S1S nicht an Praktikabilität verliert.

4.1 Boolesche Abgeschlossenheit deterministischer ω-Automaten

Bei näherer Betrachtung des von Büchi angegeben Algorithmus stellt sich heraus, daß nach der Erstellung der Behmannschen Normalform erstmals Büchiautomaten eingeführt werden. Liegt nämlich die Formel in der BNF vor, so schreibt das Originalverfahren die Transformation von Konjunktionen von Lebendigkeitsbedingungen, also Konjunktionen von Formeln der Form $\mathsf{F}\varphi$, in die Akzeptanzbedingungen der Büchiautomaten vor. Staiger und Wagner geben stattdessen in [22] eine Klasse von ω-Automaten an, die als boolescher Abschluß von Sicherheits- bzw. Lebendigkeitsbedingungen angesehen werden kann. Schneider entwickelte für diese Sprachklasse deterministische ω-Automaten [20], welche im folgenden Präfixautomaten genannt werden. In dem folgenden optimierten Entscheidungsverfahren werden eben diese Präfixautomaten benutzt, um die Behmannsche Normalform auf Präfixautomaten abzubilden. In dem darauf folgenden Schritt wird dann der Abschluß bzgl. der Disjunktion berechnet. Diese beiden Schritte können in dieser Form durchgeführt werden, weil Präfixautomaten bzgl. aller booleschen Verknüpfungen abgeschlossen sind [21, 22].

Das folgende Lemma ist gerade für deterministische Übergangsfunktionen $\mathcal{DT}(\vec{I}, \vec{P})$ von Bedeutung, da hierdurch die Komplementierung deterministischer Automaten definiert wird.

Lemma 1 Abgeschlossenheit *Gegeben seien zwei deterministische Übergangstransitionen $\mathcal{DT}_1(\vec{I}\vec{P})$ und $\mathcal{DT}_2(\vec{I}\vec{Q})$, die verschiedene Zustandsvariablen P und Q haben. Dann gelten folgende Beziehungen:*

(1.)

$$\left[\exists \vec{P}.\mathcal{DT}_1(\vec{I},\ \vec{P}) \wedge \Theta_1(\vec{I},\ \vec{P})\right] = \left[\forall \vec{P}.\mathcal{DT}_1(\vec{I},\ \vec{P}) \rightarrow \Theta_1(\vec{I},\ \vec{P})\right]$$

(2.)

$$\neg\left[\exists \vec{P}.\mathcal{DT}_1(\vec{I},\ \vec{P}) \wedge \Theta_1(\vec{I},\ \vec{P})\right] = \left[\exists \vec{P}.\mathcal{DT}_1(\vec{I},\ \vec{P}) \wedge \neg\Theta_1(\vec{I},\ \vec{P})\right]$$

(3.)

$$\begin{aligned}&\left[\exists \vec{P}.\mathcal{DT}_1(\vec{I},\ \vec{P}) \wedge \Theta_1(\vec{I},\ \vec{P})\right] \wedge \left[\exists \vec{Q}.\mathcal{DT}_2(\vec{I},\ \vec{Q}) \wedge \Theta_2(\vec{I},\ \vec{Q})\right] \\ &= \left[\exists \vec{P}\,\vec{Q}.\mathcal{DT}_1(\vec{I},\ \vec{P}) \wedge \mathcal{DT}_2(\vec{I},\ \vec{Q}) \wedge \Theta_1(\vec{I},\ \vec{P}) \wedge \Theta_2(\vec{I},\ \vec{Q})\right]\end{aligned}$$

Der Beweis dieses Lemmas beruht auf der Tatsache, daß es für jede Eingabesequenz genau einen Durchlauf durch den Automaten gibt, d.h. es gilt:

$$\forall \vec{P}\, \vec{Q}.\mathcal{DT}(\vec{I},\, \vec{P}) \wedge \mathcal{DT}(\vec{I},\, \vec{Q}) \rightarrow (\vec{P} \leftrightarrow \vec{Q})$$

. Letzteres wiederum impliziert, daß eine Klasse von deterministischen ω-Automaten genau dann bzgl. boolescher Operationen abgeschlossen ist, wenn die zugehörige boolesche Operation der Akzeptanzbedingung als ein solcher ω-Automat dargestellt werden kann.

Das nun folgende Verfahren bildet die S1S auf **quantifizierte Präfixformeln** ab. Diese Formeln besitzt in dem hier vorliegenden Fall keine 'Fairness'-Eigenschaften, d.h. die Formeln haben folgendes Aussehen (ohne zusätzliche Quantifizierung):

$$\mathcal{P}(\vec{I}) := \left(\begin{array}{l} \exists Q_1 \ldots Q_s. \\ \quad \bigwedge_{k=0}^{s} \left[(Q_k = \omega_k) \wedge \mathsf{G}(\mathsf{X}Q_k = \Omega_k(\vec{I}, \vec{Q})) \right] \wedge \\ \quad \bigwedge_{m=0}^{a} \left[\mathsf{G}\Phi_m(\vec{I}, \vec{Q})) \right] \vee \left[\mathsf{F}\Psi_m(\vec{I}, \vec{Q})) \right] \end{array} \right)$$

Dabei nennt man die Formeln $\Phi_m(\vec{I}, \vec{Q}))$ die Sicherheitseigenschaften und $\Psi_m(\vec{I}, \vec{Q}))$ die Lebendigkeitseigenschaften der Präfixformel ($\Phi_m(\vec{I}, \vec{Q}))$ und $\Psi_m(\vec{I}, \vec{Q}))$ sind aussagenlogische Formeln). Präfixformeln, die keine 'Fairness'-Eigenschaften besitzen, nennt man auch **einfache Präfixformeln**.

4.2 Transformationsalgorithmus

Da das Original- und das modifizierte Verfahren einen ähnlichen Aufbau besitzen, werden sie in diesem Kapitel einander gegenübergestellt und es wird hauptsächlich auf die Unterschiede eingegangen. Bild 4.1 zeigt beide Verfahren im Vergleich. Der Ablauf des Verfahrens von Büchi folgt den hellen Pfeilen, und entspricht dem in Kapitel 3.1 vorgestellten Verfahren. Der Ablauf des neuen Verfahrens wird in Bild 4.1 durch die dunklen Pfeile dargestellt.

Der neue Algorithmus ist ebenfalls in mehrere Schritte unterteilt. Im Gegensatz zu Büchis Verfahren wird die Eliminierung von mehrfachen Sukzessoren erst später und auf grundsätzlich andere Art und Weise durchgeführt. Zu Beginn wird auch hier syntaktischer Ballast entfernt, d.h. Relationen und die Addition werden eliminiert. Die weiteren Schritte bis zur Behmannschen Normalform unterscheiden sich nur in der verwendeten Datenstruktur. Die darauf folgenden Schritte wurden grundsätzlich verändert und folgen dem zu Beginn dieses Kapitels vorgestellten Ansatz.

Theorem 3 *Zu jeder Formel der monadischen Arithmetik zweiter Ordnung mit einem Sukzessor, in der alle numerischen Variablen gebunden sind, existiert eine äquivalente quantifizierte deterministische einfache Präfixformel.*

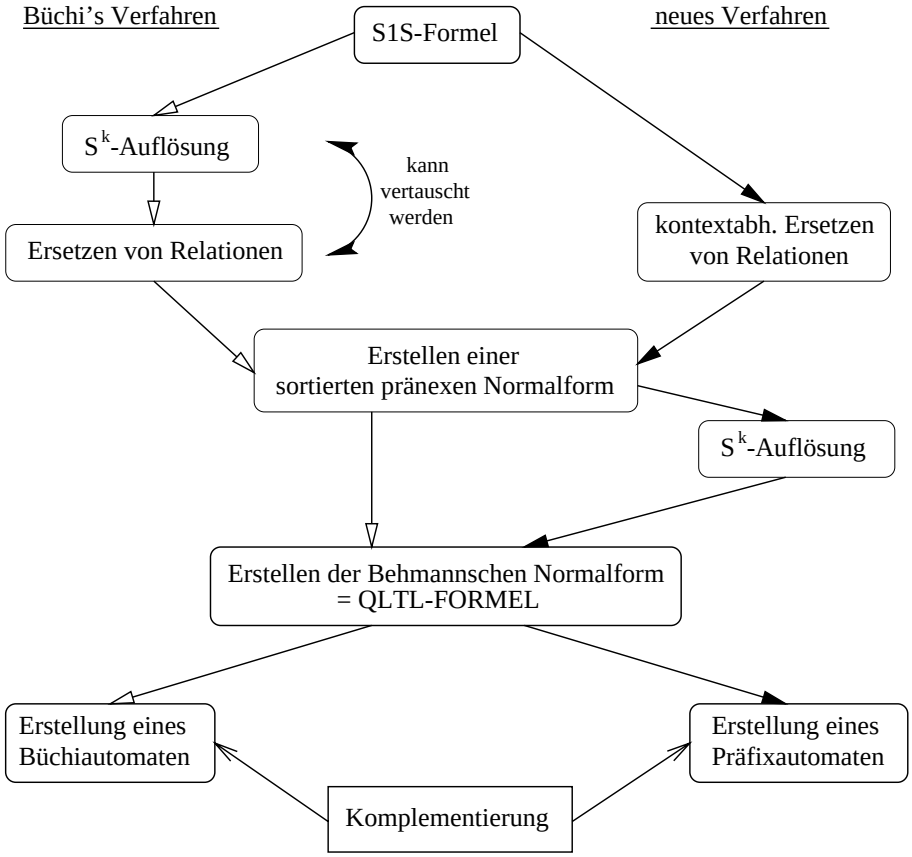

Abbildung 4.1: Direkter Vergleich der beiden Verfahren

Beweis:

1. **Kontextabhängiges Ersetzen von Relationen und anderen Erweiterungen**
   Dieser Schritt dient, wie schon bei Büchis Entscheidungsverfahren, der Eliminierung von syntaktischem Ballast. In S1S sind Relationen zwischen numerischen Variablen zugelassen. Eine weitere Berechnung mit diesen Relationen ist jedoch nicht möglich, so daß diese Relationen in prädikatenlogische Formeln umgewandelt werden müssen. Büchis Verfahren fordert lediglich die Eliminierung dieser Relationen, so daß jede beliebige kongruente Ersetzung legitim ist. Wie sich aber in Kapitel 3 herausgestellt hat, sind Wechsel von Quantoren über Prädikaten für den nicht elementaren Aufwand verantwortlich. Daher ist es erstrebenswert eben diese Wechsel möglichst zu vermeiden. In Kapitel 2.4 wurden für jede in S1S definierte Relation zwei mögliche Varianten angegeben. So kann beispielsweise die Relation $x \leq y$ auf folgende Arten ersetzt werden:

$$\begin{aligned}(x \leq y) &:= \forall P.[\forall t.P(t) \rightarrow P(S(t))] \rightarrow [P(x) \rightarrow P(y)] \\ (x \leq y) &:= \exists P.[\forall t.P(t) \rightarrow P(S(t))] \wedge \neg P(x) \wedge P(S(y))\end{aligned}$$

Bei der ersten Ersetzung entsteht ein Allquantor über dem neuen Prädikat P, während bei der zweiten Ersetzung ein Existenzquantor entsteht.

Wenn sich die zu ersetzende Relation im Bindungsbereich eines Allquantors befindet, dann ist die erste Version günstiger als die zweite, da dann ein Quantorenwechsel vermieden wird. Falls der äußere Kontext einer zu ersetzenden Relation stattdessen von einem Existenzquantor abhängt, wählt man besser die zweite Variante.

Um solche Fallunterscheidungen sinnvoll einsetzen zu können, muß jedoch sichergestellt sein, daß alle Negationen so weit wie möglich – also bis zu den Prädikaten oder den Relationen selbst – in die Formel hineingeschoben wurden. Bekanntlich entsteht bei der Negation eines Allquantors ein Existenzquantor und umgekehrt. Eine nachträgliche Negation kann daher die gezielte Auswahl des jeweils passenden Theorems vereiteln und zu einem unpassenden Quantor führen.

Bei den Ergänzungen zur S1S wurde auch eine Addition definiert. Die folgenden Formeln geben zwei mögliche Varianten für die Ersetzung der Addition an:

$$\left[\forall x_n.\Phi\left(\sum_{j=1}^{n} x_j\right)\right] = \forall z.\sum_{j=1}^{n-1} x_j \leq z \rightarrow \Phi(z)$$

$$\left[\exists x_n.\Phi\left(\sum_{j=1}^{n} x_j\right)\right] = \exists z.\sum_{i=1}^{n-1} x_j \leq z \wedge \Phi(z)$$

Genau wie bei den Relationen liegt auch bei der Addition ein Optimierungspotential in der Auswahl des passenden Ersetzungstheorems vor.

2. **Pränexe Normalform**
   Die Pränexe Normalform berechnet sich analog zu der Berechnung in Büchis Verfahren. In diesem Verfahren wurde darauf geachtet, daß beim Herausziehen der Quantoren möglichst häufig gemeinsame Prädikate verwendet werden um Quantoren einzusparen. Die Voraussetzung für eine Eliminierung von Variablen wurde in Kapitel 2.5 angegeben und erfolgt anhand der folgenden Theoreme:

3. $[\forall X_1.\mathcal{A}_1(X_1)] \wedge [\forall X_2.\mathcal{A}_2(X_2)] = \forall X_1.[\mathcal{A}_1(X_1) \wedge \mathcal{A}_2(X_1)]$
4. $[\exists X_1.\mathcal{A}_1(X_1)] \vee [\exists X_2.\mathcal{A}_2(X_2)] = \exists X_1.[\mathcal{A}_1(X_1) \vee \mathcal{A}_2(X_1)]$

   wobei $X_1,\ X_2$ Variablen und $\mathcal{A}_1\ \mathcal{A}_2$ prädikatenlogische Terme sind.

   Desweiteren wird bereits an dieser Stelle darauf geachtet, daß Wechsel von Quantoren möglichst vermieden werden. Aus diesem Grund werden dieser und der nächste Schritt

gemeinsam ausgeführt, d.h. es werden nicht zuerst alle Quantoren aus der Formel herausgezogen und anschließend vertauscht, sondern die richtige Reihenfolge der Quantoren in jedem Teilterm erstellt.

3. **Sortierte Pränexe Normalform**
Auch bei diesem Schritt verfährt man äquivalent zu dem Verfahren von Büchi. Wie bereits in Kapitel 3 genau beschrieben, müssen an dieser Stelle die Quantoren über den Prädikaten und die Quantoren über numerischen Variablen derart sortiert werden, daß eine Trennung erfolgt. Zwei Quantoren vom gleichen Typ können ohne weiteres vertauscht werden. Falls jedoch zwei Quantoren unterschiedlichen Typs vertauscht werden müssen, so ist die Erzeugung neuer Quantoren unumgänglich. Zusätzlich entsteht ein Quantorwechsel, es gilt nämlich:

$$(\exists x.\forall P.\varphi(P,x)) \;=\; (\exists Q.\forall P.\forall x.\exists z.[Q(x) \rightarrow \varphi(P,x)] \wedge Q(z))$$

für einen $\exists x.\forall P$-Wechsel und

$$(\forall x.\exists P.\varphi(P,x)) \;=\; (\forall Q.\exists P.\exists x.\forall z.Q(z) \rightarrow \varphi(P,x) \wedge Q(x)) \tag{4.1}$$

für einen $\forall x.\exists P$-Wechsel. Durch gemeinsames Hinausschieben mit sofortigem 'Sortieren' ist es möglich, die Anzahl der Wechsel von Quantoren über Prädikaten minimal zu halten.

Beispiel:

Gegeben sei das Axiom:

$$\forall t_1.\exists t_2.t_1 < t_2$$

Wie man an diesem Beispiel leicht feststellt, ist in diesem Fall ein Wechsel von Quantoren über Prädikaten unumgänglich, da bei der Ersetzung der Relation $t_1 < t_2$ entweder ein Existenzquantor oder ein Allquantor erzeugt werden muß und demnach in beiden Fällen ein 'Konflikt' mit den der Formel voranstehen Quantoren stattfindet. In diesem Fall ist es jedoch günstiger einen Existenzquantor zu erzeugen. Man erhält dann folgende Umformung:

$$\forall t_1.\exists t_2.\exists P.\forall u.\underbrace{(\neg P(t_1) \wedge P(t_2) \wedge [P(u) \rightarrow P(S(u))])}_{=:\varphi(P,t_1,t_2,u)}$$

Die erste Vertauschung ($\exists t_2.\exists P$ zu $\exists P.\exists t_2$) bereitet keinerlei Schwierigkeiten, da es sich um gleichartige Quantoren handelt. Die Vertauschung von $\exists P$ und $\forall t_1$ ist nach der Ersetzungsvorschrift 4.1 durchzuführen, damit erhält man:

$$\forall Q.\exists P.\exists t_1.\forall t_2.\forall u\; z.Q(z) \rightarrow (\varphi(P,t_1,t_2,u) \wedge Q(t_1))$$

Durch die Vertauschung von $\forall t_1$ und $\exists P$ wurde hier ein neues Prädikat Q und damit ein Quantorenwechsel über den gebundenen Prädikatsvariablen erzeugt.

4. **Eliminierung mehrfach angewandter Sukzessoren**
   Bei dem Büchi-Verfahren ist es notwendig, die mehrfach angewandten Sukzessoren gleich zu Beginn zu eliminieren. Das ist hauptsächlich dadurch begründet, daß die Zustandsübergangsformeln aus der ursprünglichen S1S-Formel entnommen werden.

   Bei dem neuen Verfahren, werden diese Formeln zu Sicherheitsbedingungen und dürfen somit bis zum Schluß mehrfach angewandte Sukzessoren besitzen. Die Zustandsübergangsformeln entstehen erst bei Erstellung der Präfixformel durch boolesche Abschlüsse. Auf diese Art und Weise kann sogar sichergestellt werden, daß es sich bei diesen Formeln um deterministische Formeln handelt, und man somit deterministische Automaten erhält.

   Ein weiterer Unterschied liegt in der Art der Eliminierung. Bei dem Büchi-Verfahren werden die mehrfach angewandten Sukzessoren auf einfache Sukzessoren transformiert, indem weitere numerische Variablen eingeführt werden. Die Ersetzung mit numerischen Variablen führt jedoch viele Relationen ein und bei der Ersetzung dieser Relationen werden dann ebenfalls Prädikatsvariablen erzeugt. Bei dem neuen Verfahren hingegen werden alle Sukzessoren auf die Stufe des am häufigsten angewandten Sukzessors angehoben, indem neue Prädikatsvariablen eingeführt werden. Zu den vorhandenen Prädikatsvariablen P_i (für $i = \{1, \ldots, n\}$ und $typ(P_i) = \mathbb{N} \rightarrow \mathbb{B}$) kommt ein Satz neuer Prädikatsvariablen hinzu, deren Anzahl sich aus dem maximalen 'Exponenten' von S ($S_{\max}$) aller P_i-Atome und der Anzahl unterschiedlicher Prädikatsvariablen n, wie folgt berechnet:

$$\Sigma_{i=1}^{n}(S_{\max} - S_{\min}^{(i)}).$$

   $S_{\min}^{(i)}$ ist hierbei der minimale 'Exponent' von S in P_i-Atomen für $i = \{1, \ldots, n\}$.

   Beispiel:

   Seien folgende P_i-Atomen in einer Formel gegeben:

$$\begin{array}{ll} P_1(x),\ P_1(S(x)),\ P_1(S^2(x)) & \Rightarrow\ S_{\min}^{(1)} = 0 \\ P_2(S(x)) & \Rightarrow\ S_{\min}^{(2)} = 1 \\ P_3(S^3(x)) & \Rightarrow\ S_{\min}^{(3)} = 3 \end{array}$$

   Daraus folgt: $S_{max} = 3$ und $n = 3$. Folgende Prädikatsvariablen müssen demnach erzeugt werden:

$$\begin{array}{ll} P_1: & P_{1,3}(S^3(x)),\ P_{1,2}(S^3(x)),\ P_{1,1}(S^3(x)) \\ P_2: & P_{2,2}(S^3(x)),\ [P_{2,1}(S^3(x))] \\ P_3: & \text{keine neue Variable} \end{array}$$

   $P_{2,1}$ tritt in der Formel nicht auf, muß aber dennoch erzeugt werden. Es müssen demnach $(S_{\max} - S_{\min}^{(1)}) + (S_{\max} - S_{\min}^{(2)}) + (S_{\max} - S_{\min}^{(3)}) = (3-0) + (3-1) + (3-3) = 5$ neue Prädikatsvariablen erzeugt werden.

Durch diese Erweiterung von Prädikatsvariablen erhält man eine Verschlechterung bzgl. der Anzahl Variablen im Vergleich zu Büchis Verfahren, wenn die Differenz von $S_{\max} - S^i_{\min}$ für $1 \leq i \leq n$ groß wird, während beim Verfahren von Büchi nur für tatsächlich vorhandene Sukzessoren neue Prädikatsvariablen eingeführt werden. Auf der anderen Seite werden diese Variablen zum einen erst zum Schluß der Transformation eingeführt, d.h. sie tauchen bei der Transformation bis einschließlich Schritt fünf, vor allem bei der Erstellung der BNF, nicht auf. Zum anderen sind diese Variablen dafür verantwortlich, daß man abschließend einen deterministischen Automaten erhält, da diese Variablen die Zustandsvariablen des Automaten sind. Mit Hilfe der Funktion PNNF (pränexe NEXT-Normalform) kann eine Formel φ um Zustandsübergangsgleichungen ergänzt werden.

function maxSUCdepth(φ)
CASE φ, ψ OF

- $P(S^k(t))$: return k ($t \in Var_\Sigma$)
- $P(S^k(0))$: return k
- $\neg\varphi$: return maxSUCdepth(φ)
- $\varphi \wedge \psi$: return max{maxSUCdepth(φ), maxSUCdepth(ψ)}
- $\varphi \vee \psi$: return max{maxSUCdepth(φ), maxSUCdepth(ψ)}
- $\forall t.\varphi$: return maxSUCdepth(φ)
- $\exists t.\varphi$: return maxSUCdepth(φ)

function liftSUC $(\varphi,\ d)$
CASE φ, ψ OF

- $P(S^k(t))$:

  IF $k < d$
  $Q =$ newvar;
  add $(\mathcal{E},\ \mathsf{G}[\mathsf{X}Q = P])$;
  liftSUC $\Big(Q(S^{k+1}(t)),\ d\Big)$;
  ELSE return $P(S^k(t))$

- $P(S^k(0))$:

  IF $k < d$
  $Q =$ newvar;
  add $(\mathcal{E},\ \mathsf{G}[\mathsf{X}Q = P])$;
  liftSUC $\Big(Q(S^{k+1}(0)),\ d\Big)$;
  ELSE return $P(S^k(0))$

- $\neg\varphi$: return $\neg$ liftSUC $(\varphi,\ d)$

- $\varphi \wedge \psi$: return liftSUC $(\varphi,\ d) \wedge$ liftSUC $(\psi,\ d)$
- $\varphi \vee \psi$: return liftSUC $(\varphi,\ d) \vee$ liftSUC $(\psi,\ d)$
- $\forall t.\varphi$: return $\forall t.$ liftSUC $(\varphi,\ d)$
- $\exists t.\varphi$: return $\exists t.$ liftSUC $(\varphi,\ d)$

function PNNF (φ)

$\mathcal{E} = \emptyset$;

$\Phi =$ liftSUC $(\varphi,$ maxSUCdepth $(\varphi))$

return $\left(\bigwedge_{\psi \in \mathcal{E}} \psi \wedge \Phi \right)$

Beispiel: Gegeben sei folgende Formel:

$$\forall t.P(S^2(t)) \vee P(t)$$

Die folgenden Zeilen, die eine Simulation der obigen Funktionen darstellen, demonstrieren die Erweiterung der Formel φ um Zustandsübergangsgleichungen:

Aufruf: PNNF$([\forall t.P(S^2(t)) \vee P(t)])$

$\mathcal{E} = \emptyset$
$\rightarrow$ liftSUC $([\forall t.P(S^2(t)) \vee P(t)],\ 2)$ $\{k = ?\}$
$\rightarrow$ liftSUC $([P(S^2(t)) \vee P(t)],\ 2)$ $\{k = ?\}$
$\rightarrow$ liftSUC $([P(S^2(t))],\ 2)$ $\{k = 2\}$
$[k = 2] : \leftarrow$ return $P(S^2(t))$
$\rightarrow$ liftSUC $([P(t)],\ 2)$ $\{k = 0\}$
$[k < 2] :$
$\{Q_1 =$ newvar;
$\mathcal{E} = \{\mathsf{G}[\mathsf{X}Q_1 = P]\};$
$\rightarrow$ liftSUC $([Q_1(S(t))],\ 2)\}$ $\{k = 1\}$
$[k < 2] :$
$\{Q_2 =$ newvar;
$\mathcal{E} = \{\mathsf{G}[\mathsf{X}Q_2 = Q_1]\} \cup \{\mathsf{G}[\mathsf{X}Q_1 = P]\};$
$\rightarrow$ liftSUC $([Q_2(S^2(t))],\ 2)\}$ $\{k = 2\}$
$[k = 2] : \leftarrow$ return $Q_2(S^2(t))$
$\leftarrow$ return $P(S^2(t)) \vee Q_2(S^2(t))$
$\leftarrow$ return $\forall t.P(S^2(t)) \vee Q_2(S^2(t))$

$$\leftarrow \left(\begin{array}{l} \exists Q_2.\ Q_2 = 0 \wedge \mathsf{G}[\mathsf{X}Q_2 = Q_1] \wedge \\ \exists Q_1.\ Q_1 = 0 \wedge \mathsf{G}[\mathsf{X}Q_1 = P] \wedge \\ (\forall t.P(S^2(t)) \vee Q_2(S^2(t)) \end{array} \right)$$

Hierbei soll $\rightarrow$ einen Funktionsaufruf und $\leftarrow$ eine Funktionswertrückgabe darstellen.

Die hierbei entstehenden Quantoren über den Prädikatsvariablen tragen nicht zu Quantorenwechseln bei, da es sich hier um Zustandsvariablen und nicht um Eingabevariablen handelt.

5. **Erstellen einer Behmannschen Normalform**
   Dieser Schritt unterscheidet sich insofern von dem ursprünglichen Verfahren, daß hier die Behmannsche Normalform unter Zuhilfenahme von BDDs erzeugt wird. Das ursprünglichen Verfahren erfordert ein häufiges Umformen der Kernformel, also der quantorenfreien Formel, in eine disjunktive Normalform bzw. eine konjunktive Normalform. Dieser Wechsel zwischen den beiden Normalformen verursacht jedoch einen hohen Aufwand, da hierdurch die Formel entweder exponentiell wächst, oder eine zusätzliche Minimierung der resultierenden Formel erforderlich ist. Die Minimierung selbst hat jedoch wieder einen exponentiellen Aufwand. Gesucht wird demnach eine Datenstruktur, die das Hineinschieben der Quantoren über den numerischen Variablen erlaubt, ohne daß die zeitaufwendige Umformung erfolgen muß.

   Eine für diese Zwecke günstige Datenstruktur sind BDDs [31]. Weil mittels einer BDD-Darstellung in den meisten Fällen sowohl ein exponentielles Wachstum, als auch eine exponentielle Laufzeit vermieden werden kann, wird die Formel an dieser Stelle in einen BDD überführt. Da jedoch auch BDD-Darstellungen nicht immer polynomiale Größe besitzen, ist auch durch diese Maßnahme nicht sichergestellt, daß deren Größe nicht exponentiell ansteigt.

   Mit Hilfe eines BDDs kann man jedoch nur aussagenlogische Formeln realisieren. Die Grammatik von S1S erlaubt allerdings auch prädikatenlogische Formeln, d.h. die bestehende Datenstruktur muß in eine Form gebracht werden, die die Verwendung von BDDs gestattet. Folgende Kombinationen von Variablen und Funktionen müssen behandelt werden:

   (a) Quantoren über numerischen Variablen

   (b) die Funktion SUC

   (c) Prädikate

   (d) numerische Variablen

   (e) die Konstante 0

   Da Quantoren über Prädikatsvariablen bei der Erstellung der Behmannschen Normalform keine Rolle spielen, können diese gesondert behandelt, d.h. außerhalb des BDD gespeichert werden. Quantoren über numerischen Variablen gehen ebenfalls nicht direkt in die BDD-Variablen ein.

Aus den noch ausstehenden vier Komponenten müssen demnach die BDD-Variablen gebildet werden, d.h. für jede in der Formel vorkommende Kombination von Prädikaten, der Funktion SUC , numerischen Variablen und der Konstanten Null wird eine BDD-Variable gebildet. Für zwei identische Kombinationen wird nur eine BDD-Variable erzeugt und in beiden Vorkommen verwendet. Diese BDD-Variablen werden nun gruppiert, und zwar bilden alle BDD-Variablen, die von derselben numerischen Variablen oder der Konstanten Null abhängen eine Gruppe und haben somit den gleichen *Rang*. Alle BDD-Variablen werden durch diese Gruppierung erfaßt, da alle atomaren Formeln der $\mathcal{L}_{\Sigma}^{S1S}$ von genau einer Variablen vom Typ $\mathbb{N}$, also von einer numerischen Variablen oder der Konstanten Null abhängen. Es gibt somit folgende Kombinationen: $P(S^k(0))$ oder $P(S^k(t))$, wobei P ein Prädikat, S die Sukzessorfunktion und t eine numerische Variable ist. Sind $P_1 \dots P_l$ die in einer Formel vorkommenden Prädikatsvariablen mit den Vorkommen $P_1(S^{k_{1,n}}(x_n)) \dots$, dann definiert man folgende partielle Ordnung:

$$\boxed{\begin{array}{c} P_1(S^{k_{1,n}}(x_n)) \\ \vdots \\ P_l(S^{k_{l,n}}(x_n)) \end{array}} \prec \boxed{\begin{array}{c} P_1(S^{k_{1,n-1}}(x_{n-1})) \\ \vdots \\ P_l(S^{k_{l,n-1}}(x_{n-1})) \end{array}} \prec \dots \prec \boxed{\begin{array}{c} P_1(S^{k_{1,1}}(x_1)) \\ \vdots \\ P_l(S^{k_{l,1}}(x_1)) \end{array}} \prec \boxed{\begin{array}{c} P_1(S^{k_{1,0}}(0)) \\ \vdots \\ P_l(S^{k_{l,0}}(0)) \end{array}}$$

Die numerischen Quantoren gehen zwar nicht direkt in die BDD-Variablen ein, sie bestimmen jedoch die Reihenfolge dieser Gruppen und damit die Position der BDD-Variablen im BDD-Graphen. Die Position von BDD-Variablen innerhalb einer Gruppe ist nicht festgelegt und darf zu Optimierungszwecken geändert werden.

Beispiel: Mit diesem Beispiel soll die Erstellung einer Behmannschen Normalform mit Hilfe von BDDs vorgeführt werden.

Sei folgende Formel gegeben:

$$\forall xz.\ (x < z) \vee (x \geq z)$$

Die sortierte pränexe Normalform dieser Formel lautet:

$$\forall P_1 P_2.\, \forall xz.\, \exists t.$$
$$[P_1(t) \wedge \neg P_1(S(t))] \vee [\neg P_1(S(x))] \vee [P_1(z)] \vee$$
$$[P_2(t) \wedge \neg P_2(S(t))] \vee [\neg P_2(z)] \vee [P_2(x)]$$

Abbildung 4.2 zeigt den zu dieser sortierten pränexen Normalform zugehörigen BDD. Der grau unterlegte Bereich in der Abbildung entspricht dem Bindungsbereich der numerischen Variablen t. Im ersten Schritt wird dieser Bereich, der der Formel:

$$\neg P_1(S(t)) \wedge P_1(t) \vee \neg P_2(S(t)) \wedge P_2(t)$$

entspricht, durch eine neue BDD-Variable $Subst_1$ ersetzt und mit dem Quantor $\exists t$ versehen. Diese BDD-Variable besitzt den höchsten Rang und wird somit im BDD ganz nach oben geschoben. Nach dieser Ersetzung gibt es im BDD keine Variable mehr, die von t abhängt. Im weiteren Verlauf der Transformation werden noch zwei weitere Substitutionen durchgeführt. Es entstehen die BDD-Variablen $Subst_2 = \forall z.\neg P_2(z) \vee P_1(z)$ für die numerische Variable z und $Subst_3 = \forall x.P_2(x) \vee \neg P_1(S(x))$ für die numerische Variable x. Nachdem alle drei Quantoren in die Formel hineingeschoben wurden, erhält man den Ergebnis-BDD, aus Abbildung 4.3. Dieser BDD weist drei Bindungsbereiche auf, wie man der Anzahl von Substitutionen entnehmen kann.

Abhängig davon, ob eine quantifizierte numerische Variable an der Bildung der BDD-Variablen beteiligt ist oder nicht, werden die BDD-Variablen in zwei Klassen eingeteilt:

(a) gebundene BDD-Variablen (von numerischen Variablen abhängig)

(b) freie BDD-Variablen (entweder von der Konstanten Null abhängig oder es handelt sich um Substitutionen).

Freie BDD-Variablen haben immer einen höheren Rang als gebundene, stehen im BDD folglich näher an der Wurzel. Nach einer gebundenen BDD-Variablen kann keine freie mehr folgen.

Die Abbildung 4.4 zeigt einen BDD und die Unterteilung der Variablen in Gruppen. Dieser BDD enthält fünf Gruppen und bei dem Hineinschieben des ersten Quantors werden zwei Substitutionen S_1 und S_2 gebildet.

Nachdem die Formel in einen BDD gewandelt wurde, ist durch die oben angegebene Reihenfolge der BDD-Variablen sichergestellt, daß diese nach Gruppen sortiert im BDD vorkommen. Daraus folgt aber, daß es eine Gruppe gibt, die den niedrigsten Rang hat, d.h. nach deren BDD-Variablen keine BDD-Variablen aus anderen Gruppen mehr folgen. Diese Gruppe, die den niedrigsten Rang einnimmt, wird jedoch gerade von der numerischen Variablen definiert, deren Quantor im nächsten Schritt in den Graphen geschoben

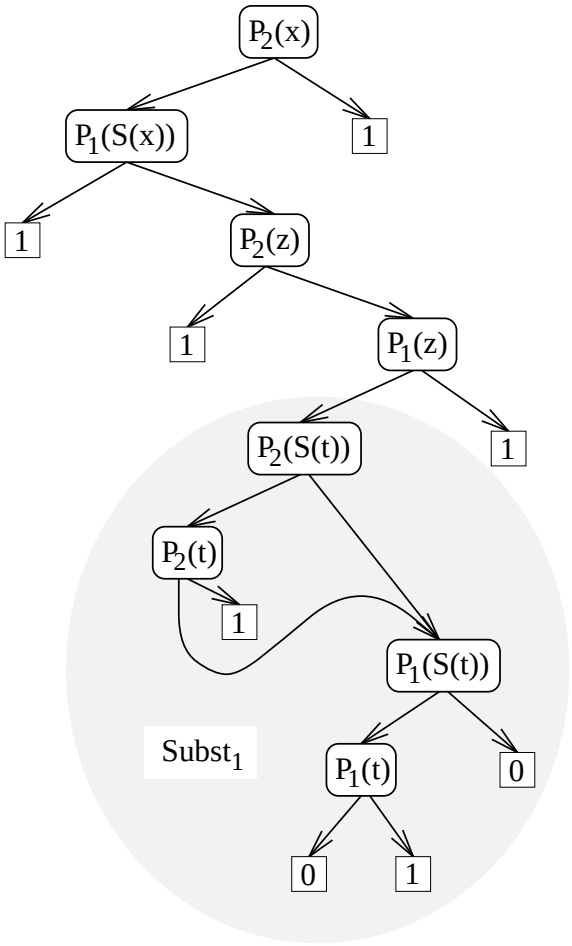

Abbildung 4.2: Beispiel: BDD

werden soll. Unter den gegebenen Bedingungen ist es jetzt aber gleichgültig, ob es sich bei dem Quantor um einen Allquantor, oder um einen Existenzquantor handelt, wie sich leicht zeigen läßt:

- Ein Allquantor wird hineingeschoben:

$$
\begin{aligned}
&\forall t.a \to b|c \text{ mit } t \notin a \\
&= \forall t.(a \to b) \wedge (\neg a \to c) \\
&= (\forall t.a \to b) \wedge (\forall t.\neg a \to c) \\
&= (a \to \forall t.b) \wedge (\neg a \to \forall t.c) \\
&= a \to (\forall t.b) \mid (\forall t.c)
\end{aligned}
$$

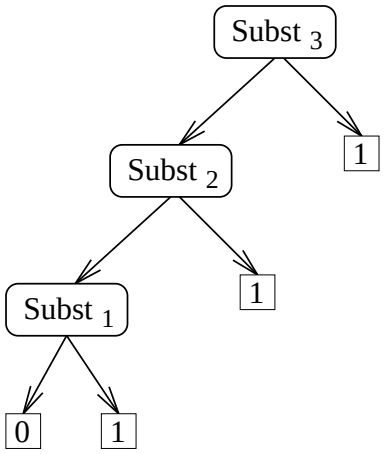

Abbildung 4.3: Beispiel: Ergebnis-BDD

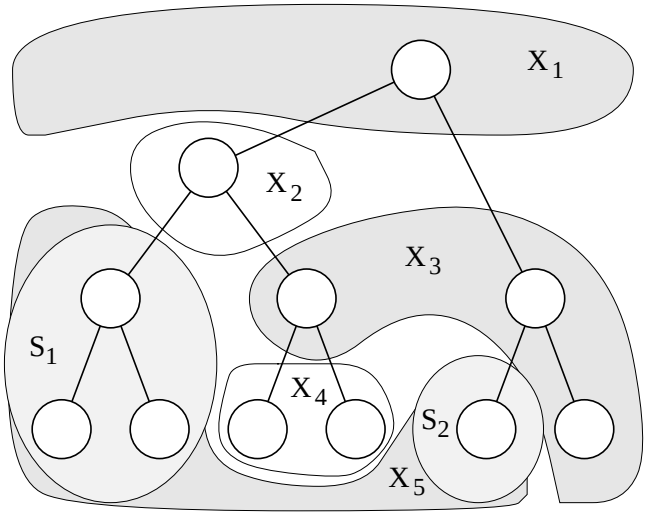

Abbildung 4.4: Gruppen im BDD

- Ein Existenzquantor wird hineingeschoben:

$$
\begin{aligned}
&\exists t.a \to b|c \text{ mit } t \notin a \\
&= \exists t.(a \wedge b) \vee (\neg a \wedge c) \\
&= (\exists t.a \wedge b) \vee (\exists t.\neg a \wedge c) \\
&= (a \wedge \exists t.b) \vee (\neg a \wedge \exists t.c) \\
&= a \to (\exists t.b) \mid (\exists t.c)
\end{aligned}
$$

Bei dem oben verwendeten Operator $a \to b|c$ handelt es sich um einen (if-then-else)-Operator, d.h. $a \to b|c = a \wedge b \vee \neg a \wedge c$.

Der am weitesten innen stehende Quantor über numerischen Variablen, also der Quantor, der zu der Gruppe mit niedrigstem Rang gehört, kann nun in den BDD hineingeschoben

werden. Da nach dem ersten Auftauchen einer BDD-Variablen mit niedrigstem Rang keine BDD-Variable aus einer anderen Gruppe mehr folgt, bildet der Teilgraph, der diese BDD-Variable als Wurzel hat, einen Bindungsbereich des aktuellen Quantor. Alle diese Teilgraphen werden daraufhin durch neue freie BDD-Variablen, also Variablen deren Rang höher ist, als der Rang jeglicher gebundener Variablen, substituiert.

Nach der darauffolgenden Umschichtung des BDDs (die neuen BDD-Variablen wandern Richtung BDD-Wurzel), kann der nächste Quantor in die Formel hineingeschoben werden.

Im Verlauf der Transformation werden alle numerischen Quantoren in die Formel hineingeschoben und gleichzeitig alle Teilgraphen, aus jeweils nur Variablen einer Variablengruppe bestehend, substituiert. Zu Schluß besteht der BDD dann nur noch aus freien Variablen. Jede Substitution entspricht einem Bindungsbereich der Behmannschen Normalform.

Die Behmannsche Normalform wird nun nach der in Kapitel 3.1 vorgestellten Funktion 'Behmann2LTL' in eine QLTL-Formel transformiert. Danach wird diese QLTL-Formel in eine Pränex-X-Normalform überführt (siehe hierzu auch [20, 21]). Das Ergebnis dieser Transformationen hat dann folgendes Aussehen:

$$\begin{aligned}\Delta_1 Q_1 \ldots \Delta_m Q_m . \exists P_1 \ldots P_n . \\ &\bigwedge_{j=1}^{n} [P_j = 0] \wedge [\mathsf{G}[\mathsf{X} P_j = X_j]] \wedge \\ &\bigvee_{j=1}^{a} \mathcal{I}_j[\vec{I}, \vec{Q}] \wedge \underbrace{\left[\mathsf{G}\mathcal{G}_j[\vec{I}, \vec{Q}]\right] \wedge \left(\bigwedge_{k=1}^{b_j} \mathsf{F}\mathcal{H}_{j,k}[\vec{I}, \vec{Q}]\right)}_{\text{Präfixformel}}\end{aligned}$$

Die Variable X_j ist eine der Variablen $\vec{I}$, $\vec{Q}$ oder $\vec{P}$, mit $X_j \neq P_j$ für $j = 1, \ldots, n$ und bei der Formeln $\mathcal{I}_j[\vec{I}, \vec{Q}]$, $\mathcal{G}_j[\vec{I}, \vec{Q}]$ und $\mathcal{H}_{j,k}[\vec{I}, \vec{Q}]$ handelt es sich um aussagenlogische Formeln.

$\mathcal{G}_j[\vec{I}, \vec{Q}]$ und $\mathcal{H}_{j,k}[\vec{I}, \vec{Q}]$ sind bereits Präfixformeln und die Konjunktion von zwei Präfixformeln ist wieder eine Präfixformel.

Die Formel $\mathcal{I}_j[\vec{I}, \vec{Q}]$ ist jedoch keine Präfixformel und muß daher in eine Präfixformel $\mathcal{J}_j[\vec{I}, \vec{Q}]$ transformiert werden. Dies geschieht mit Hilfe des folgenden Theorems [20, 21]:

$$\mathcal{J}_j[\vec{I}, \vec{Q}] = \left(\begin{array}{l} \exists P. \\ \quad (P = 0) \wedge (\mathsf{G}[\mathsf{X} P = 1]) \wedge \\ \quad \mathsf{G}[\mathcal{I}_j[\vec{I}, \vec{Q}] \wedge P] \end{array} \right)$$

Bei den Formeln $\mathsf{G}\,\mathcal{G}_j[\vec{I}, \vec{Q}]$ und $\mathsf{F}\,\mathcal{H}_j[\vec{R}_j]$ handelt es sich um Sicherheits- und Lebendigkeitseigenschaften.

Da die Konjunktion von einfachen Präfixformeln wiederum einfache Präfixformeln ergibt, kann für $j = 1, \ldots, a$:

$$\mathcal{J}[\vec{I}, \vec{Q}] \wedge \left[\mathsf{G}\, \mathcal{G}_j[\vec{I}, \vec{Q}]\right] \wedge \left[\mathsf{F}\, \mathcal{H}_j[\vec{I}, \vec{Q}]\right]$$

durch eine einzelne deterministische Präfixformel $\mathcal{P}_j$ ersetzt werden [20, 21]. $\bigvee \mathcal{P}_j$ ist aber aufgrund der booleschen Abgeschlossenheit von Präfixformeln ebenfalls eine Präfixformel (siehe [20]).

Nach Abarbeitung dieser Schritte erhält man somit eine quantifizierte deterministische Präfixformel:

$$\Delta Q_1 \ldots \Delta Q_m . \mathcal{P}[\vec{R}]$$

wobei $\mathcal{P}[\vec{R}]$ eine deterministische einfache Präfixformel ist.

■

Das Ergebnis der Transformation ist demnach eine quantifizierte einfache deterministische Präfixformel. Die Gültigkeit der Formel $\mathcal{P}[\vec{R}]$ kann einfach in ein CTL-Modellprüfungsproblem überführt werden und dann beispielsweise mit dem 'SMV'-Beweiser entschieden werden [20, 21]. Schwierigkeiten bei der weiteren Entscheidung von $\Delta Q_1 \ldots \Delta Q_m . P[\vec{R}]$ bereiten die voranstehenden Quantoren $\Delta Q_1 \ldots \Delta Q_m$. Analog zu Büchis Verfahren können diese Quantoren eliminiert werden. Ziel dieser Eliminierung ist es, ausschließlich Allquantoren zu erhalten.

In [20] wird gezeigt, daß einfache deterministische bzw. indeterministische Präfixformeln in deterministische bzw. indeterministische ω-Automaten mit einer Akzeptanzbedingung der Form $\mathsf{FG}\Psi$, wobei Ψ eine aussagenlogische Formel ist, transformiert werden können. Im folgenden sei diese deterministische Automatenklasse mit $\mathcal{D}_{FG}$ und die indeterministische Automatenklasse mit $\mathcal{N}_{FG}$ bezeichnet.

Ein $\mathcal{N}_{FG}$-Automat kann mit Hilfe der 'subset construction' in einfach exponentieller Zeit in einen $\mathcal{D}_{FG}$-Automaten transformiert werden.

Die folgenden Fälle können in einfach exponentieller Laufzeit entschieden werden:

- $\forall \vec{P} . \mathcal{P}[\vec{P}, \vec{X}]$:
  Diese Formeln sind gültig, wenn die Formel $\mathcal{P}[\vec{P}, \vec{X}]$ gültig ist. Dieses Entscheidungsproblem kann auf eine CTL-Modellprüfung gleicher Größe abgebildet werden [20, 21].

- $\forall \vec{P} . \exists \vec{Q} . \mathcal{P}[\vec{P}, \vec{Q}, \vec{X}]$:
  In diesem zweiten Fall liegt ein Wechsel von Quantoren über Prädikaten vor. Die Formel $\exists \vec{Q} . \mathcal{P}[\vec{P}, \vec{Q}, \vec{X}]$ kann als eine indeterministische Präfixformel angesehen werden, bei der die Zustandsvariablen $\vec{Q}$ keiner Einschränkung unterliegen.

  Folgende Schritte sind zur Entscheidung notwendig:

  - Sei $\mathcal{R}[\vec{P},\vec{Q},\vec{X}]$ ein $\mathcal{D}_{FG}$-Automat für die Präfixformel $\mathcal{P}[\vec{P},\vec{Q},\vec{X}]$. Dann kann $\exists\vec{Q}.\mathcal{R}[\vec{P},\vec{Q},\vec{X}]$ als ein $\mathcal{N}_{FG}$-Automat angesehen werden, da für die zusätzlichen Zustandsvariablen $\vec{Q}$ keine Einschränkung vorliegt.
  - Dieser $\mathcal{N}_{FG}$-Automat kann in einfach exponentieller Zeit in einen $\mathcal{D}_{FG}$-Automaten $\mathcal{R}'[\vec{P},\vec{X}]$ transformiert werden.
  - Dann folgt wie im ersten Fall, daß $\forall\vec{P}.\mathcal{R}'[\vec{P},\vec{X}]$ gilt, wenn $\mathcal{R}'[\vec{P},\vec{X}]$ gilt.
  - $\mathcal{R}'[\vec{P},\vec{X}]$ kann in eine CTL-Modellprüfung gleicher Größe überführt werden.

- $\exists\vec{P}.\forall\vec{Q}.\mathcal{P}[\vec{P},\vec{Q},\vec{X}]$:
  Die Formel $\exists\vec{P}.\forall\vec{Q}.\mathcal{P}[\vec{P},\vec{Q},\vec{X}]$ ist äquivalent zu $\exists\vec{P}.\neg\exists\vec{Q}.\neg\mathcal{P}[\vec{P},\vec{Q},\vec{X}]$ und kann über folgende Vorschrift entschieden werden:

  - Zuerst muß das Komplement $\mathcal{P}'[\vec{P},\vec{Q},\vec{X}]$ von der Präfixformel $\mathcal{P}[\vec{P},\vec{Q},\vec{X}]$ berechnet werden. Da es sich bei $\mathcal{P}[\vec{P},\vec{Q},\vec{X}]$ um eine deterministische Präfixformel handelt, ist dies effizient (lineare Zeit) durchführbar.
  - Danach wird $\mathcal{P}'[\vec{P},\vec{Q},\vec{X}]$ in einen deterministischen $\mathcal{D}_{FG}$-Automaten $\mathcal{R}[\vec{P},\vec{Q},\vec{X}]$ übersetzt (lineare Zeit).
  - $\exists\vec{Q}.\mathcal{R}[\vec{P},\vec{Q},\vec{X}]$ ist dann wie im zweiten Fall wieder ein $\mathcal{N}_{FG}$-Automat, der deterministisch gemacht werden muß. Man erhält einen $\mathcal{D}_{FG}$-Automaten $\mathcal{R}'[\vec{P},\vec{X}]$ (exponentielle Zeit).
  - Als nächstes muß $\mathcal{R}'[\vec{P},\vec{X}]$ komplementiert werden. Dies geschieht indem die Akzeptanzbedingung $\mathsf{FG}\Psi$ in eine $\mathsf{GF}\neg\Psi$ gewandelt wird. Man erhält einen deterministischen Büchiautomaten $\mathcal{B}[\vec{P},\vec{X}]$ (konstante Zeit).
  - Mit $\exists\vec{P}.\mathcal{B}[\vec{P},\vec{X}]$ erhält man einen indeterministischen Büchiautomaten, der dann aber auf ein entsprechendes Modell zur Entscheidung überführt werden kann.

Für jeden weiteren Wechsel von Quantoren über Prädikaten muß auch in diesem Verfahren ein indeterministischer Büchiautomat komplementiert werden. Dies geschieht wie in [9] beschrieben.

Der Vorteil dieses Verfahren gegenüber dem Verfahren von Büchi ist, daß hier zuerst eine deterministische Formel entsteht, die wie in Lemma 1 gezeigt, effizient komplementiert werden kann. Es ist demnach möglich, Formeln, die sich auf einen Quantorenwechsel beschränken in exponentieller Zeit zu entscheiden.

Der Nachteil an diesem Verfahren ist die Einschränkung von S1S. Für den Fall, daß mehr als ein Quantorwechsel über Prädikaten entsteht, ist der Aufwand wieder nicht elementar und demzufolge mit dem Aufwand des ursprünglichen Entscheidungsverfahrens gleichzusetzen.

Somit ist es möglich das oben bewiesene Theorem zu modifizieren:

Theorem 4 *Zu jeder Formel der monadischen Arithmetik zweiter Ordnung mit einem Sukzessor, in der alle numerischen Variablen gebunden sind existiert eine äquivalente quantifizierte deterministische einfache Präfixformel. Diese Präfixformel kann in einfach exponentieller Zeit entscheiden werden, falls die Anzahl von Quantorenwechsel über Prädikaten einen Wechsel nicht überschreitet.*

Kapitel 5

Beispiele und experimentelle Ergebnisse

Die bisherigen Kapitel beschäftigten sich mit der theoretischen Seite dieser Arbeit. In diesem Kapitel soll nun anhand von Beispielen die praktische Seite dieses Verfahrens bewertet werden. In den Abschnitten 5.1.1 und 5.1.2 werden zwei Formeln auf theoretischer Ebene berechnet. In Abschnitt 5.2 werden dann experimentell gemessene Ergebnisse angegeben.

5.1 Beispiele

Dieses Kapitel demonstriert die Funktionsweise des Transformationsverfahrens anhand einiger Beispiele, welche zur Illustration des Verfahrens im Detail durchgerechnet werden.

5.1.1 Beispiel: 0 ist das Minimum von $\mathbb{N}$

$$
\begin{aligned}
\forall x.0 \leq x &= \\
&= \forall x.\forall P.[\exists t.P(t) \wedge \neg P(S(t))] \vee \neg P(0) \vee P(x) \\
&= \forall P.\forall x.\exists t.(P(t) \wedge \neg P(S(t)) \vee \neg P(0) \vee P(x) \\
&= \forall P.[\exists t.P(t) \wedge \neg P(S(t)] \vee \neg P(0) \vee [\forall x.P(x)] \\
&= \forall P.[\mathsf{F}[P \wedge \neg \mathsf{X}P]] \vee \neg P \vee [\mathsf{G}P] \\
&= \forall P.\exists P_1.[P_1 = 0 \wedge \mathsf{X}P_1 = P] \wedge ([\mathsf{F}[\mathsf{X}P_1 \wedge \neg \mathsf{X}P]] \vee \neg \mathsf{X}P_1 \vee [\mathsf{G}\mathsf{X}P_1])
\end{aligned}
$$

Zur Entscheidung dieses Beispiels wurden fünf Zustandsvariablen eingeführt, wobei von den 32 möglichen Zuständen 14 erreicht werden können. Die Berechnung dauerte auf einer SPARC10-Workstation insgesamt $0,22$ Sekunden, wobei der Beweiser 'SMV' $0,06$ Sekunden benötigte.

5.1.2 Beispiel: Modulo-3-Zähler

In diesem Beispiel soll der Modulo-3-Zähler, der in Kapitel 1.3 Bild 1.3 spezifiziert wurde, erneut betrachtet werden. Da bereits bei diesem Beispiel umfangreiche Formeln entstehen, soll die Berechnung auch nur zum Teil durchgeführt werden. Schwerpunkt dieser Betrachtung soll die Entwicklung der Quantoren sein, d.h. quantorenfreie prädikatenlogische Formeln werden meist nur abgekürzt dargestellt. Prädikatsvariablen seien im folgenden wieder mit großen Buchstaben $(P_i,\ Q_j.\ R_k)$ und numerische Variablen mit kleinen Buchstaben $(t_i,\ s_j,\ r_k)$ bezeichnet. Die Prädikate Ein und Aus bezeichnen, wie bereits in Kapitel 1.3 angegeben, das Eingabe- und das Ausgabesignal. Sei demnach folgende S1S-Formel gegeben:

$$\forall t_1 t_2 t_3.[\forall s_1.Ein(s_1) \leftrightarrow (s_1 = t_1) \vee (s_1 = t_2) \vee (s_1 = t_3)] \rightarrow [\forall s_2.Aus(s_2) \leftrightarrow (s_2 = t_3)]$$

In einem ersten Schritt wird diese Formel derart umgeformt, daß alle booleschen Operatoren auf $\wedge$, $\vee$, $\neg$ zurückgeführt werden. Dies ist erforderlich, da die übrigen booleschen Operatoren implizit die Negation enthalten, die zu Beginn des Verfahrens möglichst weit in die Formel hineingeschoben werden muß. Die Formel hat dann folgendes Aussehen:

$$\begin{array}{lll} \forall t_1 t_2 t_3. & [\exists s_1. & Ein(s_1) \wedge (s_1 \neq t_1) \wedge (s_1 \neq t_2) \wedge (s_1 \neq t_3) \vee \\ & & \neg Ein(s_1) \wedge (s_1 = t_1) \vee \\ & & \neg Ein(s_1) \wedge (s_1 = t_2) \vee \\ & & \neg Ein(s_1) \wedge (s_1 = t_3)] \vee \\ & [\forall s_2. & (Aus(s_2) \vee (s_2 \neq t_3)) \wedge \\ & & (\neg Aus(s_2) \vee (s_2 = t_3))] \end{array}$$

Im nächsten Schritt werden nun die Relationen '$=$' und '$\neq$' ersetzt. Bei dieser Ersetzung ist es erforderlich, in der ersten Teilformel, die von s_1 abhängt, Existenzquantoren über neuen Prädikaten zu erzeugen und in der zweiten Teilformel, die von s_2 abhängt, Allquantoren über neuen Prädikaten zu erzeugen. Man erhält demnach folgende Formel:

$$\begin{array}{lllll} \forall t_1 t_2 t_3. & [\exists s_1. & [Ein(s_1) \wedge & (\exists P_1.P_1(s_1) \not\leftrightarrow P_1(t_1)) \wedge \\ & & & (\exists P_2.P_2(s_1) \not\leftrightarrow P_2(t_2)) \wedge \\ & & & (\exists P_3.P_3(s_1) \not\leftrightarrow P_3(t_3))] \vee \\ & & [\neg Ein(s_1) \wedge & (\exists P_4 P_5.\forall r_1.\varphi_1)] \vee \\ & & [\neg Ein(s_1) \wedge & (\exists P_6 P_7.\forall r_2.\varphi_2)] \vee \\ & & [\neg Ein(s_1) \wedge & (\exists P_8 P_9.\forall r_3.\varphi_3)]] \ \vee \\ & [\forall s_2. & [Aus(s_2) \vee & (\forall Q_1 Q_2.\exists r_4.\varphi_4)] \wedge \\ & & [\neg Aus(s_2) \vee & (\forall Q_3.Q_3(s_2) \leftrightarrow Q_3(t_3))]] \end{array}$$

Die Formeln φ_i sind quantorenfreie prädikatenlogische Formeln, wobei

- φ_1 von den Variablen $P_4,\ P_5,\ t_1,\ s_1$ und r_1 abhängt, mit

$$\varphi_1 \ := \ \exists P_4 P_5. \left(\begin{array}{l} [\forall r_1.P_4(r_1) \rightarrow P_4(S(r_1))] \wedge P_4(S(t_1)) \wedge \neg P_4(s_1) \wedge \\ [\forall r_1.P_5(r_1) \rightarrow P_5(S(r_1))] \wedge P_5(S(s_1)) \wedge \neg P_5(t_1) \end{array} \right),$$

- φ_2 von den Variablen $P_6,\ P_7,\ t_2,\ s_1$ und r_2 abhängt, mit

$$\varphi_2 \ := \ \exists P_6 P_7. \left(\begin{array}{l} [\forall r_2.P_6(r_2) \rightarrow P_6(S(r_2))] \wedge P_6(S(t_2)) \wedge \neg P_6(s_1) \wedge \\ [\forall r_2.P_7(r_2) \rightarrow P_7(S(r_2))] \wedge P_7(S(s_1)) \wedge \neg P_7(t_2) \end{array} \right),$$

- φ_3 von den Variablen $P_8,\ P_9,\ t_3,\ s_1$ und r_3 abhängt, mit

$$\varphi_3 \ := \ \exists P_8 P_9. \left(\begin{array}{l} [\forall r_3.P_8(r_3) \rightarrow P_8(S(r_3))] \wedge P_8(S(t_3)) \wedge \neg P_8(s_1) \wedge \\ [\forall r_3.P_9(r_3) \rightarrow P_9(S(r_3))] \wedge P_9(S(s_1)) \wedge \neg P_9(t_3) \end{array} \right) \text{ und}$$

- φ_4 von den Variablen $Q_1,\ Q_2,\ t_3,\ s_2$ und r_4 abhängt, mit

$$\varphi_4 \ := \ \forall Q_1 Q_2. \left(\begin{array}{l} [\forall r_4.Q_1(r_4) \rightarrow Q_1(S(r_4))] \rightarrow [Q_1(S(t_3)) \rightarrow Q_1(s_2)] \vee \\ [\forall r_4.Q_2(r_4) \rightarrow Q_2(S(r_4))] \rightarrow [Q_2(S(s_2)) \rightarrow Q_2(t_3)] \end{array} \right).$$

Die nächsten beiden Schritte im Transformationsverfahren bringen die vorliegende Formel in eine sortierte pränexe Normalform. In diesen Schritten können Variablen durch die Eliminierung von Prädikatsvariablen eingespart werden. Als Zwischenergebnis erhält man:

$$\begin{array}{lll} \forall t_1 t_2 t_3. & [\exists s_1. & [\exists P_1 P_2 P_3.\psi] \vee \\ & & [\exists P_4 P_5.\forall r_1.\neg Ein(s_1) \wedge \varphi_1] \vee \\ & & [\exists P_6 P_7.\forall r_2.\neg Ein(s_1) \wedge \varphi_2] \vee \\ & & [\exists P_8 P_9.\forall r_3.\neg Ein(s_1) \wedge \varphi_3]] \ \vee \\ & [\forall s_2. & [\forall Q_1 Q_2.\exists r_4.Aus(s_2) \vee \varphi_4] \wedge \\ & & [\forall Q_3.\neg Aus(s_2) \vee (Q_3(s_2) \leftrightarrow Q_3(t_3))]] \end{array}$$

wobei

$$\begin{array}{rcll} \psi & := & Ein(s_1) \wedge & \\ & & (P_1(s_1) \neq P_1(t_1)) \wedge & \\ & & (P_2(s_1) \neq P_2(t_2)) \wedge & \\ & & (P_3(s_1) \neq P_3(t_3)) & \text{(abgekürzte Version)} \end{array}$$

In der ersten Teilformel können die Prädikatsvariablen $P_4,\ P_6$ und P_8 durch P_1, sowie die Prädikatsvariablen $P_5,\ P_7$ und P_9 durch P_2 ersetzt werden. In der zweiten Teilformel kann die Prädikatsvariable Q_3 durch Q_1 ersetzt werden. Man erhält dann:

$$\begin{array}{lll} \forall t_1 t_2 t_3. & [\exists P_1 P_2 P_3.\exists s_1.\forall r_1 r_2 r_3.\Psi_1] \vee \\ & [\forall Q_1 Q_2.\forall s_2.\exists r_4.\Psi_2] \end{array}$$

Ψ_1 und Ψ_2 sind wieder prädikatenlogische Formeln, mit

$$\begin{array}{rcl} \Psi_1 & := & \psi \vee \neg Ein(s_1) \wedge \left([\varphi_2]^{P_1 P_2}_{P_4 P_5} \vee [\varphi_3]^{P_1 P_2}_{P_6 P_7} \vee [\varphi_4]^{P_1 P_2}_{P_8 P_9} \right) \text{ und} \\ \Psi_2 & := & (Aus(s_2) \vee \varphi_4) \wedge \\ & & \neg Aus(s_2) \vee (Q_1(s_2) \leftrightarrow Q_1(t_3)). \end{array}$$

Wie die obige Formel zeigt, werden bereits an dieser Stelle die Quantoren in den einzelnen Teilformeln 'sortiert'. Nachdem alle Quantoren in optimaler Reihenfolge aus der Formel herausgezogen wurden, erhält man:

$$\forall Q_1 Q_2.\, \forall t_1 t_2 t_3.\, \exists P_1 P_2 P_3.\, \exists s_1.\, \forall s_2.\, \forall r_1 r_2 r_3.\, \exists r_4.\, [\Psi_1 \vee \Psi_2]$$

Eine endgültige Sortierung bereitet an dieser Stelle jedoch Schwierigkeiten, da die numerischen Variablen $t_1,\ t_2$ und t_3 über Allquantoren gebunden sind, die Prädikatsquantoren $P_1,\ P_2$ und P_3 aber über Existenzquantoren gebunden sind. An dieser Stelle müssen die Vertauschungstheoreme für Quantoren unterschiedlichen Typs (siehe Seite 35) eingesetzt werden. Man erhält dann folgende Formel:

$$\forall R_3 R_2 R_1 Q_1 Q_2.\, \exists P_1 P_2 P_3.\, \exists t_1 t_2 t_3 s_1\, \forall s_2 r_1 r_2 r_3\, \exists r_4.\, \forall z_1 z_2 z_3.\, \Phi$$

Bei diesen Vertauschungen entstehen die Prädikatsvariablen $R_1,\ R_2$ und R_3 und die numerischen Variablen $z_1,\ z_2$ und z_3. Wie leicht zu überprüfen ist, entsteht bei dieser Formel ein einfacher Quantorenwechsel über Prädikaten.

Die Anzahl aller in der Formel vorkommenden Kombinationen von Variablen konnte durch Eliminierung von Prädikatsvariablen auf 33 reduziert werden. Durch die Vertauschung der Quantoren von P_i und t_j entstehen aber wiederum sechs neue Variablenkombinationen, so daß insgesamt 39 Variablenkombinationen entstehen, d.h. zur weiteren Abarbeitung muß diese Formel nun in einen BDD mit 39 BDD-Variablen transformiert werden. Wie man leicht nachvollziehen kann trägt die Berechnung der Behmannschen Normalform wesentlich zur benötigten Laufzeit des Algorithmus bei, da die Größe eines BDD exponentiell von der Anzahl seiner Variablen abhängen kann.

Die nun folgenden Schritte können an dieser Stelle nicht weiter fortgeführt werden, da dieses Beispiel hierfür zu umfangreich ist.

5.2 Laufzeiten

In Tabelle 5.2 werden Laufzeiten wiedergegeben, die bei der Entscheidung einiger HOL-Theoreme aus der Theorie 'arithmetic' gemessen wurden. Einigen Theoreme, wie beispielsweise das Theorem: 'NOT-NUM-EQ' wurden in konjunktive Teilformeln unterteilt und dann in mehreren Schritten bewiesen. Diese Unterteilung bewirkt eine Verbesserung der Laufzeit des Programmes.

Da im Verlaufe dieser Arbeit der letzte Transformationsschritt, also die Komplementierung der Präfixautomaten, nicht implementiert wurde, können nur Formeln bewiesen werden, bei denen kein Quantorenwechsel über Prädikaten entsteht.

Zeit-Modifikation	+0	+1	+2	+3	+4
'SMV'-Zeit:	2,4s	6,1s	14,2s	32,8s	80,6s
Transf.-Zeit:	1,1s	1,2s	1,3s	1,4s	1,4s

Tabelle 5.1: Laufzeiten eines 'Zeitmodifizierten' HOL-Theorems

Das Programm, das während dieser Arbeit erstellt wurde, wurde in C++ geschrieben und benutzt zum Einlesen der Formeln ein mit Hilfe der Parsergeneratoren FLEX/BISON erstellten Parser. Die Überführung in Präfixformeln und später auf CTL-Formeln erfolgt über eine von Dr. Klaus Schneider bereitgestellte C++Bibliothek. Diese Bibliothek stellt u.a. Funktionen zur Verarbeitung von QLTL-Formeln, Präfixformeln und Fair- Präfixformeln zur Verfügung. Die Auswertung der CTL-Formeln erfolgt mit Hilfe des Beweisers 'SMV'. Zur Erzeugung der Behmannschen Normalform wurde eine Bibliothek verwendet, die BDD-Funktionen zur Verfügung stellt.

Die Tests und Laufzeitmessungen erfolgten auf einer SPARC10-Workstation mit dem Betriebssystem SPARC-Solaris (Version 5.5.1) und 64MB Hauptspeicher.

Eine weitere Laufzeitmessung demonstriert, daß der zeitliche Aufwand der Transformation von dem maximalen Exponenten von S, der mit S_{max} bezeichnet wird, in nur sehr geringem Maße abhängt. In Tabelle 5.1 wird das HOL-Theorem ('LESS-NOT-SUC'):

$$\forall mn.((m < n) \wedge \neg(n = S(m))) \rightarrow (S(m) < n)$$

insoweit modifiziert, daß alle Variablenvorkommen von m und n durch $S^i(m)$ und $S^i(n)$ ersetzt wurden, wobei mit i die Zeitmodifikation angegeben wird. Aus der dritten Zeile von Tabelle 5.1 ist ersichtlich, daß im Vergleich zu der Gesamtzeit der Berechnung die Transformationszeit annähernd konstant bleibt.

Neben der Komplementierung der Präfixformeln bestimmen die BDD-Operationen zum großen Teil die Gesamtlaufzeit der Transformationsverfahrens. Wie bereits angegeben ist auch hier ein exponentieller Aufwand vorhanden. Das Verfahren reagiert sehr empfindlich auf eine große Anzahl BDD-Variablen. Zusätzlich kommt noch hinzu, daß bei dem Hineinschieben von numerischen Quantoren, die Anzahl der Substitutionen, die erzeugt werden, von der Größe des BDD, also indirekt von der Anzahl BDD-Variablen, abhängt. Diese Substitutionen werden dann wieder zu BDD-Variablen, was die Größe des BDD noch weiter erhöht. Aus diesen Gründen wirkt sich eine frühzeitige Aufspaltung der zu entscheidenden Formel in Konjunktionsterme, positiv auf die Laufzeit des Entscheidungsverfahrens aus, da dadurch die Anzahl potentieller BDD-Variablen in den meisten Fällen verringert werden kann.

Dieses Entscheidungsverfahren eignet sich hauptsächlich für kleinere Formeln. Das Beispiel aus Kapitel 5.1.2 zeigte, daß kleine Beispiele durchaus schnell entschieden werden können.

Theorem		**Laufzeit**	
Name	Formel	gesamt	SMV
SUC-NOT	!n. (0=suc(n));	0,34	0,16
LESS-MONO-REV	!m.!n.(suc(m)<suc(n))->(m<n);	0,24	0,05
LESS-MONO-EQ	!m.!n.(suc(m)<suc(n))<->(m<n);		
(Teil 1)	!m.!n.(suc(m)<suc(n))->(m<n);	0,29	0,11
(Teil 2)	!m.!n.(m<n)->(suc(m)<suc(n));	0,35	0,17
LESS-TRANS	!m.!n.!p.((m<n)&(n<p))->(m<p);	0,37	0,19
LESS-ANTISYM	!m.!n. ((m<n)&(n<m));	0,30	0,13
LESS-LESS-SUC	!m.!n. ((m<n)&(n<suc(m)));	0,30	0,10
LESS-OR	!m.!n.(m<n)->(suc(m)<=n);	0,27	0,09
OR-LESS	!m.!n.(suc(m)<=n)->(m<n);	0,32	0,12
LESS-EQ	!m.!n.(m<n)<->(suc(m)<=n);		
(Teil 1)	!m.!n.(m<n)->(suc(m)<=n);	0,26	0,09
(Teil 2)	!m.!n.(suc(m)<=n)->(m<n);	0,33	0,10
LESS-SUC-EQ-COR	!m.!n.((m<n)& (suc(m)=n))->(suc(m)<n);	2,21	1,90
LESS-NOT-SUC	!m.!n.((m<n)& (n=suc(m)))->(suc(m)<n);	2,17	1,95
LESS-0-CASES	!m.(0=m)|(0<m);	0,34	0,17
LESS-CASES-IMP	!m.!n.((m<n)& (m=n))->(n<m);	0,96	0,64
LESS-CASES	!m.!n.(m<n)|(n<=m);	0,35	0,15
LESS-EQ-SUC-REFL	!m.m<=suc(m);	0,30	0,11
LESS-EQ-ANTISYM	!m.!n. ((m<n)&(n<=m));	0,27	0,12
NOT-LESS	!m.!n. (m<n)<->(n<=m);		
(Teil 1)	!m.!n. (m<n)->(n<=m);	0,29	0,12
(Teil 2)	!m.!n.(n<=m)-> (m<n);	0,28	0,11
LESS-SUC-NOT	!m.!n.(m<n)-> (n<suc(m));	0,29	0,15
LESS-EQ-TRANS	!m.!n.!p.((m<=n)&(n<=p))->(m<=p);	0,45	0,20
LESS-EQ-REFL	!m.m<=m;	0,25	0,05
EQ-LESS-EQ	!m.!n.(m=n)<->((m<=n)&(n<=m));	1,36	0,77
NOT-NUM-EQ	!m.!n. (m=n)<->((suc(m)<=n)|(suc(n)<=m));		
(Teil 1)	!m.!n. (m=n)->((suc(m)<=n)|(suc(n)<=m));	1,02	0,70
(Teil 2)	!m.!n.(suc(m)<=n)-> (m=n);	0,34	0,13
(Teil 3)	!m.!n.(suc(n)<=m)-> (m=n);	0,40	0,16

Tabelle 5.2: Laufzeiten zur Entscheidung von HOL-Theoremen

Das Programm kann entweder als eigenständiger Beweiser fungieren und dazu benutzt werden, kleinere Formeln zu beweisen. Andererseits kann es auch innerhalb eines anderen Beweisersystems wie beispielsweise HOL agieren und das übergeordnete System ergänzen.

Die bereits angesprochene Einschränkung, nämlich die Begrenzung der Wechsel von Quantoren über Prädikaten auf einen Wechsel, wirkt sich nicht gravierend auf die Praktikabilität aus, da die meisten Formeln, u.a. ganz LTL mit einem Quantorenwechsel über Prädikaten spezifiziert werden können.

Literaturverzeichnis

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

[2] A. PNUELI. Applications of temporal logic to the specification and verification of reactive systems: A survey of current trends. In *Current Trends in Concurrency* (New-York, 1986), J.W. Baker, W.-P. de Roever, and G. Rozenberg, Eds., vol. 224 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 510–584.

[3] A.P. SISTLA, AND E.M. CLARKE. The complexity of propositional linear temporal logics. *Journal of Assoc. Comput. Mach. 32*, 3 (July 1986), 733–749.

[4] A.P. SISTLA, M.Y. VARDI, AND P. WOLPER. The complementation problem for büchi automata with applications to temporal logic. In *Proceedings of the Twelfth International Colloquium on Automata, Languages and Programming* (New York, 1987), vol. 194 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 465–474.

[5] A.P. SISTLA, M.Y. VARDI, AND P. WOLPER. The complementation problem for büchi automata with applications to temporal logic. *Theoretical Computer Science 49* (1987), 217–237.

[6] A.R. MEYER. Weak monadic second order theory of successor is not elementary recursive. In *Proc. of Logic Colloquium* (1980), Springer-Verlag, pp. 132–154.

[7] C.C. ELGOT. Decision problems of finite automata design and related arithmetics. *Trans. Amer. Math. Soc. 98* (1961), 21–52.

[8] D. A. BASIN, AND N. KLARLUND. Hardware verification using monadic second-order logic. In *Proceedings of the 7th International Conference On Computer Aided Verification* (Liege, Belgium, July 1995), P. Wolper, Ed., vol. 939 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 31–41.

[9] D. SIEFKES. *Decidable Theories I: Büchi's Monadic Second Order Successor Arithmetic*. Lecture Notes in Mathematics. Springer-Verlag, 1970.

[10] D. SIEFKES. The recursive sets in certain monadic second order fragments of arithmetic. *Arch. Math. Logik 17* (1975), 71–80.

[11] D.E. MULLER. Infinite sequences and finite machines. In *Proc. of 4th Annual Symposium on Switching Circuit Theory and Logical Design* (New York, 1963), pp. 3–16.

[12] E.A. EMERSON. Temporal and Modal Logic. In *Handbook of Theoretical Computer Science* (Amsterdam, 1990), J. van Leeuwen, Ed., vol. B, Elsevier Science Publishers, pp. 996–1072.

[13] 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* (New York, January 1985), ACM, pp. 84–96.

[14] E.M. CLARKE, AND E.A. EMERSON. Synthesis of Synchronization Skeletons for Branching Time Temporal Logic. In *Logics of Programs: Workshop* (Yorktown Heights, New York, May 1981), vol. 131 of *Lecture Notes in Computer Science*, Springer-Verlag.

[15] 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 (April 1986), 244–263.

[16] H.BEHMANN. Beiträge zur Algebra der Logik, insbesondere zum Entscheidungsproblem. *Mathematische Annalen 86* (1922), 163–229.

[17] J. HALPERN, Z. MANNA, AND B. MOSZKOWSKI. A hardware semantics based on temporal intervals. In *Proceeding of the Tenth International Colloquium on Automata, Languages, and Programming* (New York, 1983), vol. 154 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 278–291.

[18] J.R. BÜCHI. On a decision method in restricted second order arithmetic. In *Proceedings of the 1960 International Congress on Logic, Methodology and Philosophy of Science* (Stanford, CA, 1960), E. Nagel, Ed., Stanford University Press, pp. 1–12.

[19] K. GÖDEL. Über formal unentscheidbare sätze der Principia Mathematica und verwandter Systeme. *Monatshefte für Mathematik und Physik 38* (1931), 173–198.

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

[21] K. SCHNEIDER. Translating LTL Model Checking to CTL Model Checking. Tech. Rep. SFB358-C2-3/96, Universität Karlsruhe, Institut für Rechnerentwurf und Fehlertoleranz, January 1996. ftp://goethe.ira.uka.de/pub/techreports/SFB358-C2-3-96.ps.gz.

[22] L. STAIGER, AND K.W. WAGNER. Automatentheoretische Charakterisierungen topologischer Klassen regulärer Folgenmengen. *Elektron. Informationsverarb. Kybernet. 10* (1974), 379–392.

[23] M. FISHER, AND R. LADNER. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences 18,* 2 (1979), 194–211.

[24] M. MACHTEY, AND P.YOUNG. *An Introduction to the General Theory of Algorithms*. North-Holland, 1978.

[25] M. YOELI. Formal verification of hardware design. Tech. rep., IEEE Computer Society Press, Los Alamitos, 1990.

[26] M.J.C. GORDON, AND T.F. MELHAM. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*. Cambridge University Press, 1993.

[27] M.O. RABIN. Decidability of second-order theories and automata on infinite trees. *Trans. Amer.Math.Soc. 141* (1969), 1–35.

[28] M.VARDI, AND L. STOCKMEYER. Improved upper and lower bounds for modal logics of programs. In *Proceeding of the Seventh Annual ACM Symposium On Theory of Computing* (1985), pp. 240–251.

[29] P. WOLPER. Temporal logic can be more expressive. *Information and Control 56* (1983), 72–99.

[30] R. MCNAUGHTON. Testing and generating infinite sequences by a finite automaton. *Information and Control 9* (1966), 521–530.

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

[32] S. OWRE, J.M. RUSHBY, N. SHANKAR, AND M.K. SRIVAS. A tutorial on using PVS for hardware verification. In *Proc. 2nd International Conference on Theorem Provers in Circuit Design (TPCD94)* (Bad Herrenalb, Germany, September 1994), T. Kropf and R. Kumar, Eds., vol. 901 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 258–279. published 1995.

[33] S. SAFRA. On the complexity of ω automata. In *29th Annual IEEE Symp.on Foundations of Computer Science* (1988), pp. 319–327.

[34] W. THOMAS. Automata on infinite objects. In *Handbook of Theoretical Computer Science* (Amsterdam, 1990), J. van Leeuwen, Ed., vol. B, Elsevier Science Publishers, pp. 133–191.

[35] W. THOMAS. Infinite trees and automaton definable relations over ω words. In *7th Ann.Symp.STACS90* (1990), C.Choffrut and T.Lengauer, Eds., Springer-Verlag, pp. 263–277.

[36] W.M. WAITE, AND G. GOOS. *Compiler Construction*. Texts and Monographs in Computer Science. Springer Verlag, 1984.