Eine Verallgemeinerung der Überwachersynthese mit Hilfe des μ-Kalküls

Zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften

von der Fakultät für Informatik der Universität Fridericiana zu Karlsruhe (TH)

genehmigte

Dissertation

von

Roberto Ziller

aus Porto União / Brasilien

Tag der mündlichen Prüfung:	**8. Juli 2005**
Erster Gutachter:	**Prof. Dr.-Ing. D. Schmid**
Zweiter Gutachter:	**Prof. Dr. rer. nat. P. Deussen**

Wanderer, es gibt keinen Weg.
Er entsteht, indem du ihn gehst.

Antonio Machado, Cantares

Für meine Ibi

Danksagung

Wenn ich mich nun frage, wem ich alles bei der Fertigstellung dieser Dissertation zu danken habe, dann gehen meine Gedanken weit in die Vergangenheit zurück. Der Grund dafür ist, dass Erfahrungen aus verschiedenen Lebenszeiten in eine solche Arbeit einfließen. Jeder Mensch, dem wir begegnen, nimmt etwas von uns auf und gibt uns etwas von sich. Deshalb könnte ich hier auch sehr viele Menschen nennen, die sich ihres Beitrags zu dieser Arbeit gar nicht bewusst sind. Grundsätzlich sollte sich jedoch jeder angesprochen fühlen, der auch nur mit der einfachsten Geste mir sein Wohlwollen gezeigt und mich damit auf meinem bisherigen Lebensweg positiv beeinflusst hat.

Mein besonderer Dank gilt Professor Detlef Schmid, der mit seiner freundlichen und überaus kompetenten Führungsweise schlichtweg ideale Bedingungen für die Forschungsarbeiten an seinem Institut geschaffen hat. Ebenso danke ich Professor Peter Deussen für die Übernahme des Korreferats und die kritische Durchsicht der Arbeit. Wie er haben auch die Professoren Wolfgang Karl, Winfried Görke, Peter Schmitt und Walter Tichy während der Professorenrunde meine Aufmerksamkeit auf unklare Aspekte der Arbeit gelenkt. Professor Wonham von der Universität Toronto danke ich für die fortlaufende Diskussionsbereitschaft, und Professor André Arnold vom LaBRI in Bordeaux für den Gedankenaustausch zum Vergleich unserer Ansätze. Ebenso danke ich Michael Seidl für den Gedankenaustausch über das Beispiel „Telefonnummernauskunft“.

Die vorliegende Dissertation ist das Ergebnis meiner Tätigkeit am Institut für Rechnerentwurf und Fehlertoleranz, heute Institut für Technische Informatik, von Mai 2000 bis November 2005. Dank der guten Stimmung unter Professoren und Kollegen sowie der gegenseitigen Unterstützung wird mir diese Zeit als ein besonders schöner Lebensabschnitt in Erinnerung bleiben. Unter den Kollegen gilt besonders Klaus Schneider mein herzlicher Dank für die vielen, langen Diskussionen über alle möglichen Fragen, in denen er sein schier unerschöpfliches Wissen mit mir teilte. Ebenso gilt Fridtjof Feldbusch, Kai Kapp, Matthias Pfeffer, Andreas Schäfer, Tobias Schüle, Michael Syrjakow und Sebastian Wilhelmi mein herzlichster Dank für die freundschaftliche Zusammenarbeit. Ich danke auch dem Sekretariat, insbesondere Frau Renate Murr-Grobe, sowie der Institutswerkstatt für die freundliche Unterstützung.

Brigitte und Christoph Thomalla sind mir während dieser Zeit in vielen praktischen Lebensfragen behilflich gewesen. Ihre Freundschaft hat mich stets begleitet und mir in unzähligen Ausflügen und Wanderungen die Kräfte erneuert. Ihnen danke ich auch besonders für die Unterstützung zu Beginn der Zeit in Karlsruhe.

Besonders möchte ich Uwe Wrede hervorheben, der mir vor vielen Jahren über seine Promotionsarbeit berichtete und in mir den Traum weckte, eines Tages auch zu promovieren. Ähnlich motivierte mich Klaus de Geus mit stets ermunternden Worten und Taten. Ihnen danke ich auch für das Korrekturlesen, sowie Sascha Gottfried und ganz besonders Petra Malik.

Meine Promotion wäre allerdings nicht ohne den frühen Beitrag meiner Mutter zustande gekommen, die sich auch unter erschwerten Bedingungen unermüdlich für eine möglichst gute Ausbildung ihrer Kinder eingesetzt hat. Dieses Ziel nun erreicht zu haben ist mir auch deshalb eine große Freude. Meiner Schwester Sílvia danke ich für die vielen Worte des Muts und der Zuversicht, und meinem Bruder Heinz nicht zuletzt dafür, dass er mir das Nim-Spiel beigebracht hat.

Mein größter Dank gilt jedoch meiner Frau Diomara, die für mich und mit mir den großen Schritt über den Ozean gewagt und sich in der neuen Umgebung so tapfer geschlagen hat. Ihr Mut war mir in vielen Stunden ein Vorbild, und ohne ihre Liebe und verständnisvolle Unterstützung wäre die Fertigstellung dieser Arbeit unvergleichlich schwerer gewesen.

Kurzfassung

Diese Arbeit befasst sich mit dem Entwurf von Systemen mit diskreten Zustandsräumen. In diese Systemklasse fallen insbesondere informationsverarbeitende Systeme, die ja unseren Alltag in großem Maße beeinflussen. Aufgrund der sehr großen Zustandsräume, die durch die Nebenläufigkeit der darin ablaufenden Prozesse entstehen, kann die korrekte Funktion eines solchen Systems in der Regel nicht durch reines Testen gewährleistet werden. Es ist deshalb naheliegend, nach Entwurfsmethoden zu suchen, die sich von vorn herein an dem Ziel Korrektheit der Funktion orientieren. Die vorliegende Arbeit befasst sich mit zwei Ansätzen, die in den letzten Jahrzehnten erfolgreich in der Handhabung diskreter Systeme gewesen sind: die Überwachersynthese und die auf dem μ-Kalkül basierende Modellprüfung. Beiden gemeinsam ist der formale Charakter. Ansonsten unterscheiden sie sich sowohl in der Herkunft, als auch in der Modellierung der Systeme und in der Zielsetzung bei der Problemstellung, wodurch sich unterschiedliche Vor- und Nachteile ergeben. In dieser Arbeit wird die Überwachersynthese mit Hilfe des μ-Kalküls verallgemeinert. In dem entstehenden Ansatz ergänzen sich beide Verfahren gegenseitig, so dass die unterschiedlichen Vorteile gemeinsam genutzt werden können und wesentliche Nachteile wegfallen.

Die Überwachersynthese ist das wesentliche Merkmal des Ramadge-Wonham-Modells für die Steuerung diskreter Systeme, das seit Beginn der 1980er Jahre in der Arbeitsgruppe von Professor W.M. Wonham an der Universität Toronto entwickelt wird. Das Verfahren verlangt von dem Entwickler eine Beschreibung des zu steuernden Systems und eine Spezifikation für dessen gewünschtes Verhalten. Beide werden als endliche Automaten dargestellt. Die Systembeschreibung berücksichtigt alle zur Laufzeit möglichen Ereignisreihenfolgen, und enthält beispielsweise auch solche, die zu unerwünschten Verklemmungen führen. Mit der Spezifikation kann der Entwickler in einem solchen Fall den Wunsch ausdrücken, dass das System verklemmungsfrei laufen soll. Mit dem Syntheseverfahren wird dann aus der Systembeschreibung und der Spezifikation ein *Überwacher* berechnet, der das System zur Laufzeit so einschränkt, dass keine Verklemmungen auftreten können. Besonders zu beachten ist, dass die hier genannte Spezifikation nicht unbedingt implementierbar ist, zumal sie nur ein gewünschtes Verhalten ausdrückt, das sich unter Umständen nicht realisieren lässt. Somit ist die Überwachersynthese als ein Werkzeug anzusehen, mit dessen Hilfe aus der Spezifikation eines gewünschten Verhaltens eine tatsächlich implementierbare Spezifikation abgeleitet werden kann.

Die Geschichte der Modellprüfung ist wesentlich umfangreicher und von mehreren Forschergruppen geprägt. Im Gegensatz zu der Überwachersynthese wird hier eine vorhandene Spezifikation vorausgesetzt, die auf gewisse Eigenschaften geprüft werden soll. Um dies zu erreichen, muss ein Modell des zu entwerfenden Systems erstellt werden. Ein Algorithmus bestätigt dann die erwarteten Eigenschaften bzw. liefert ein Gegenbeispiel dazu, falls sie nicht zutreffen. Ist dies der Fall, muss der Entwurf angepasst und der Prozess wiederholt werden. Das Ziel ist es, nach wenigen Iterationen ein Modell zu erreichen, das alle Eigenschaften erfüllt. Aus dieser Beschreibung geht hervor, dass die Modellprüfung sich nicht mit der Modellierung an sich beschäftigt. Dadurch wird dem Entwickler eine Aufgabe überlassen, deren Schwierigkeitsgrad nicht zu unterschätzen ist.

In diesem Kontext ist es naheliegend, die Überwachersynthese mit der Modellprüfung zu kombinieren. Wie aus der obigen Beschreibung hervorgeht, kann die Synthese der Modellprüfung bei der Erstellung der Modelle behilflich sein. Insbesondere aber kann die Synthese selbst mit Hilfe der Modellprüfung verbessert werden. In dieser Arbeit werden die endlichen Automaten für die Überwachersynthese in spezielle Kripke-Strukturen übersetzt. Aus den Synthesealgorithmen wird ein μ-Kalkül-Gleichungssystem gewonnen, dessen Lösung der Synthese des Überwachers entspricht. Damit wird das Überwachersyntheseproblem auf ein Modellprüfungsproblem zurückgeführt. Die Vorteile dieser alternativen Berechnung zeigen sich besonders in den folgenden Punkten:

- *Bekämpfung der Zustandsexplosion:* Die bei der Überwachersynthese verwendeten Automaten können Größenordnungen erreichen, die eine Auflistung ihrer Transitionen unmöglich machen. Dieses Problem tritt auch in der Modellprüfung auf und wurde dort mit Hilfe der *symbolischen Darstellung* in vielen Fällen erfolgreich gelindert. Diese Lösung wird in dieser Arbeit auf die Überwachersynthese übertragen.
- *Überprüfbarkeit des Syntheseergebnisses:* Der Überwacher hat die Aufgabe, das System zur Laufzeit so einzuschränken, dass die gegebene Spezifikation nicht verletzt wird. Allerdings sieht das Verfahren keine praktische Methode vor, den Grad dieser Einschränkungen zu überprüfen. Es kann deshalb vorkommen, dass diese zu stark ausfallen und das Ergebnis für die Praxis unbrauchbar wird. Die Kombination der Überwachersynthese mit der Modellprüfung erlaubt es zum ersten Mal, beide Verfahren auf dasselbe Modell anzuwenden. Dies bedeutet, dass ein System, das mit Hilfe der Überwachersynthese entworfen wird, unmittelbar auf gewünschte Eigenschaften überprüft werden kann. Darüber hinaus erlaubt es die Verallgemeinerung der Überwachersynthese im letzten Kapitel dieser Arbeit, die zu prüfenden Eigenschaften bereits zur Synthesezeit zu berücksichtigen, so dass die nachträgliche Überprüfung des Ergebnisses entfällt.
- *Vereinfachung der Synthesewerkzeuge:* Durch die Formulierung der Synthesealgorithmen im μ-Kalkül vereinfacht sich die Implementierung eines Synthesewerkzeugs im Wesentlichen auf das Einlesen der Modelle und die Lösung eines μ-Kalkül-Gleichungssystems. Ein flexibles Programm sollte allerdings in der Lage sein, verschiedene Gleichungssysteme zu behandeln, was den unterschiedlichen Verbesserungen, die in dieser Arbeit vorgestellt werden, entgegen kommt.
- *Integration der Synthese in vorhandene Modellprüfer:* Der in dieser Arbeit vorgestellte Ansatz ermöglicht es, die Überwachersynthese in bereits vorhandene Werkzeuge für die Modellprüfung zu integrieren. Dabei ist eine weitgehende Wiederverwendung der Quellen zu erwarten, zumal die dazu notwendigen Erweiterungen ausschließlich auf Funktionen beruhen, die auch gängige Modellprüfer benutzen.
- *Weiterentwicklung der Synthesealgorithmen:* In dieser Arbeit werden die bekannten Synthesealgorithmen in μ-Kalkül-Gleichungssysteme übersetzt. Dies ermöglicht nicht nur die Zusammenführung von Überwachersynthese und Modellprüfung, sondern auch eine Weiterentwicklung der Synthesealgorithmen. Der Grund dafür ist, dass es leichter ist, nach den Regeln der booleschen Algebra Änderungen an einem Gleichungssystem vorzunehmen, als einen in natürlicher Sprache (z.B. Deutsch oder Englisch) verfassten Algorithmus entsprechend anzupassen. So dienen die Gleichungssysteme, die in Kapitel 4 aus den Synthesealgorithmen entstehen, als Ausgangspunkt für weitere Verbesserungen. In Kapitel 5 wird ein neues Gleichungssystem für die Überwachersynthese gefunden, dessen Lösung in vielen praktischen Fällen eine geringere Komplexität als die ursprünglichen Algorithmen aufweist. Damit wird deren Effizienz signifikant übertroffen. In Kapitel 6 geht aus den Gleichungssystemen eine Verallgemeinerung der Überwachersynthese hervor.
- *Erweiterung um neue Spezifikationseigenschaften:* In dem Ramadge-Wonham-Modell lassen sich Sicherheits- und Lebendigkeitseigenschaften ausdrücken, während Fortdauer- und Fairnesseigenschaften dagegen nur in einem damit nicht kompatiblem Ansatz von Thistle und Wonham berücksichtigt wurden. Letztere spielen jedoch in reaktiven Systemen eine wichtige Rolle. Die o.g. Verallgemeinerung der Überwachersynthese erlaubt es, jede im μ-Kalkül ausdrückbare Eigenschaft in die Spezifikationen einzubeziehen. Sie behandelt all die o.g. Eigenschaften erstmals einheitlich und schließt damit eine bedeutende Lücke im Syntheseverfahren.
- *Annäherung an informale Spezifikationen:* Da der μ-Kalkül eine Obermenge gängiger Temporallogiken wie CTL, CTL* und LTL ist, kann der Benutzer die informalen Spezifikationen zunächst mit Hilfe einer dieser Logiken beschreiben und diese Ausdrücke dann maschinell in den μ-Kalkül übersetzen. Temporallogische Ausdrücke sind in der Regel leichter zu verstehen als Gleichungssysteme und überbrücken somit die Kluft zwischen menschlicher Intuition und der Syntax des μ-Kalküls. Es entsteht ein Verfahren, das möglichst nah an die informalen Anforderungen anknüpft und nach deren Übersetzung in temporallogische Ausdrücke in der Lage ist, die gesuchte Lösung automatisch zu generieren.
- *Steigerung der Akzeptanz:* Obwohl bereits wichtige Probleme mit Hilfe der Überwachersynthese gelöst wurden, ist diese nicht besonders stark verbreitet. Die hier erzielten Verbesserungen können im Laufe der Zeit eine breitere Anwendung der Überwachersynthese begünstigen.

Abstract

A Generalization of Supervisor Synthesis based on the μ-Calculus

The present work deals with the specification and implementation of systems with discrete state spaces. This large class of systems encompasses, for example, information processing systems, which play an important role in modern life. The dynamics of these systems include concurrent processes, and concurrency can give rise to state spaces and transition relations of such a size that it becomes impossible to explicitly list them in the memory of a computer. As a consequence, it is in general not possible to guarantee that a system will work properly in all situations only by testing, because the number of situations that can be simulated is too small compared to the number of theoretical possibilities. It is therefore natural to think of methods that focus on a correct design from the very beginning. From many approaches following that direction, supervisor synthesis theory and μ-calculus based model checking have proven useful in tackling discrete systems in the last decades. Both have in common the formal treatment of the subject, and differ as well in their origin as in the way used to model a system, and in the goals to be reached. Therefore, they have different advantages and disadvantages. In the present work, supervisor synthesis is extended with the help of μ-calculus model checking. The resulting approach is able to combine the advantages of both components and to avoid the disadvantages that appear when using them independently.

Supervisor synthesis is the main component of the Ramadge-Wonham model for control of discrete, event-driven systems. The model has been under constant development in the group of W.M. Wonham at the University of Toronto since about 1980. The method requires the developer to describe both the system to be controlled and a specification for its desired behavior, both using finite automata. The description of the system is supposed to include every sequence of events that can occur at runtime, and therefore also models situations that have to be avoided, as for example deadlocks. In such a case the specification would be used to state that the system should never run into a deadlock. A synthesis procedure is then used to compute a *supervisor* for the system, whose role is to restrict the behavior of the latter in such a way that it never runs into a dead end state. It is important to note that the above mentioned specification is not necessarily an implementable one. The model accounts for the possibility of specifications that represent a wish that cannot be fulfilled. Supervisor synthesis can therefore be seen as a tool that can take a description of what is ideally wanted and derive a truly implementable specification from it.

The history of μ-calculus based model checking is quite longer than that of supervisor synthesis and involves the work of a large number of scientists. In contrast to supervisor synthesis, an implementable model of the system is supposed to be given, along with some desired property. A model checking algorithm then either confirms the wanted property or else generates a counterexample that shows why it does not hold. In this case, the developer has to make modifications to the model and to repeat the process. Ideally, a model of the system satisfying all desired properties is reached after a few iterations. As is clear from this description, model checking is not concerned with the creation of the model to be checked. This leaves the developer on their own with a task whose degree of difficulty should not be underestimated.

In this context, it is natural to ask whether supervisor synthesis and model checking can be combined. As can be seen from the above description of both methods, synthesis can be used to generate the model to which model checking is to be applied. Moreover, it turns out that supervisor synthesis can be improved with the aid of model checking techniques. In this work, the finite automata used in supervisor synthesis are translated into special Kripke structures, and the synthesis algorithms are converted into μ-calculus equation systems. These are constructed in such a way that their solution amounts to the computation of the supervisor. This reduces supervisor synthesis to a model checking problem, with the following advantages:

- *Alleviation of the state explosion problem:* The automata needed for supervisor synthesis can reach dimensions that make it impossible to list their transition relations explicitly. This problem is also known in the model checking domain and has been successfully dealt with in many practical cases by using a *symbolic representation* of the transition relation of the system. The present work also applies this solution to supervisor synthesis.
- *Verifiability of the synthesis result:* The supervisor is designed to restrict the possible actions of the system at runtime in such a way that the given specification is not violated. It is therefore possible that these restrictions turn out to be too strong, rendering the result useless in practice. With the method described herein, it becomes possible for the first time to apply both supervisor synthesis and model checking to the same representation of a system. This means that the result of the synthesis process can be immediately checked for the desired properties. Moreover, the generalization of the synthesis method achieved in the last chapter of this work allows to consider these properties during the synthesis process itself, thereby making post-synthesis verification superfluous.
- *Simplification of synthesis tools:* The translation of the synthesis algorithms into the μ-calculus simplifies the implementation of synthesis tools. The programmer does no longer have to learn about synthesis and is basically concerned with reading the input automata and solving an equation system. A flexible program should nevertheless be able to accept different equation systems, in order to tackle the different solutions presented herein.
- *Extension of existing model checkers with supervisor synthesis:* The method presented here allows to extend existing model checking tools to also do supervisor synthesis. It is reasonable to expect a high degree of code reusability, since the extensions needed use only functions that are already present in standard model checkers.
- *Improvement of existing synthesis algorithms:* An important step in this work is the translation of the standard synthesis algorithms into μ-calculus equation systems. This translation allows not only to combine synthesis and model checking, but also to derive new synthesis algorithms. This happens because it is easier to make modifications to an equation system using well-known rules from Boolean algebra than to change an algorithm written in natural language – e.g. English or German – accordingly. That way, the equation systems derived from the original algorithms in Chapter 4 serve as a starting point for improvements that go into two directions. In Chapter 5, a new equation system is found, whose solution has a lower complexity than the original one in many important practical applications. This allows to significantly improve the efficiency of the synthesis process. In Chapter 6, the equation systems give raise to a generalization of the supervisor synthesis method.
- *New specification properties:* The Ramadge-Wonham model allows safety and liveness properties to be used in the specification of the desired behavior of a system. In contrast, persistence and fairness properties have been treated only separately in an approach by Thistle and Wonham that uses a different representation of the system. Fairness properties, however, play an important role in reactive systems. The above mentioned generalization allows any property that can be expressed in the μ-calculus to be used in a specification. This allows a uniform handling of all the mentioned properties, thus solving an important open problem in supervisor synthesis theory.
- *Closing the gap to informal specifications:* An important aspect of every formal method is the translation of the given informal requirements into the formal representation employed. Since the μ-calculus subsumes temporal logics like CTL, CTL*, and LTL, it is possible to write down the informal requirements in one of these logics, and then to translate them into the μ-calculus automatically. Since temporal logics are quite intuitive, this helps to close the gap between the informal requirements and the μ-calculus syntax. This results in a method whose input is as close as possible to human language and, after the translation into temporal logics, can solve the synthesis problem automatically.
- *Increasing acceptance of synthesis methods:* Although supervisor synthesis has been used in the past to solve some important problems, it is still not being widely used. The improvements achieved in this work could contribute to improve acceptance and recognition of formal synthesis methods.

Notation

$\Diamond$	Modaloperator zur Bestimmung existentieller Vorgängerzustände
$\diamondsuit$	Modaloperator zur Bestimmung existentieller Nachfolgerzustände
$\Box$	Modaloperator zur Bestimmung universeller Vorgängerzustände
$\boxdot$	Modaloperator zur Bestimmung universeller Nachfolgerzustände
$\exists x.\varphi$	existentielle Quantifizierung der Variablen x in φ
A	universeller Pfadquantor in temporallogischen Ausdrücken
$\mathcal{A}$	Automat
$\mathcal{A}\|_Q$	Einschränkung des Automaten $\mathcal{A}$ auf die Zustandsmenge Q
$\mathcal{A}_\mathcal{E}$	Hilfsautomat zur Erstellung der Spezifikation im Ramadge-Wonham-Modell
$\mathcal{A}_\mathcal{M} \| \mathcal{A}_\mathcal{N}$	parallele Komposition der Automaten $\mathcal{A}_\mathcal{M}$ und $\mathcal{A}_\mathcal{N}$
$\mathcal{A}_\mathcal{M} \times \mathcal{A}_\mathcal{N}$	Produkt der Automaten $\mathcal{A}_\mathcal{M}$ und $\mathcal{A}_\mathcal{N}$
$\mathcal{A}_\mathcal{P}$	Automat für die Systembeschreibung im Ramadge-Wonham-Modell
$\mathcal{A}_\mathcal{S}$	Automat für den Überwacher im Ramadge-Wonham-Modell
$\mathsf{Ac}(\mathcal{A})$	erreichbare Komponente des Automaten $\mathcal{A}$
$\mathsf{act}_\mathcal{A}(q)$	Menge der aktiven Ereignisse im Zustand q des Automaten $\mathcal{A}$
A_{min}	Sprache zur Spezifikation des kleinsten akzeptablen Verhaltens
$\mathsf{Co}(\mathcal{A})$	co-erreichbare Komponente des Automaten $\mathcal{A}$
δ	Transitionsrelation eines Automaten
E	existentieller Pfadquantor in temporallogischen Ausdrücken
F	Zeitoperator in temporallogischen Ausdrücken: irgendwann
$f_\varphi(Q)$	die Zustandsmenge $\left[\!\left[[\varphi]_u^{\varphi_Q} \right]\!\right]$
G	Zeitoperator in temporallogischen Ausdrücken: immer
$\mathcal{I}$	Menge der Initialzustände einer Kripke-Struktur
K	reguläre Sprache
$\overline{K}$	Menge aller Präfixe der Sprache K
$\mathcal{K}$	Kripke-Struktur
$\mathcal{K}_\mathcal{A}$	Kripke-Struktur des Automaten $\mathcal{A}$
$\kappa(\varphi)$	Spiegelung der Zustandsmenge φ auf die jeweils andere Hälfte einer Kripke-Struktur
$\mathcal{L}$	Funktion zur Beschriftung der Zustände eines Automaten bzw. einer Kripke-Struktur
$\mathcal{L}_\mu$	Menge der flachen μ-Kalkül-Formeln
$\vec{\mathcal{L}}_\mu$	Menge der vektoriellen μ-Kalkül-Formeln
$\mathsf{L}_M(\mathcal{A})$	akzeptierte Sprache des Automaten $\mathcal{A}$
$\mathsf{L}_Q(\mathcal{A})$	generierte Sprache des Automaten $\mathcal{A}$
M	Menge der akzeptierenden Zustände eines Automaten
$\mu u.\varphi$	kleinster Fixpunkt der monotonen Funktion φ der Variablen u
$\nu u.\varphi$	größter Fixpunkt der monotonen Funktion φ der Variablen u
P, Q	Zustandsmengen von Automaten oder Kripke-Strukturen
p, q	Zustände eines Automaten oder einer Kripke-Struktur
φ, ψ	boolesche Ausdrücke zur symbolischen Darstellung von Mengen
$[\varphi]_x^y$	Variablensubstitution: φ mit x durch y ersetzt
$[\![\varphi]\!]_\mathcal{A}$	die durch φ auf dem Automaten $\mathcal{A}$ dargestellte Zustandsmenge
$[\![\varphi]\!]_\mathcal{K}$	die durch φ auf der Kripke-Struktur $\mathcal{K}$ dargestellte Zustandsmenge

φ_Q	jeder Ausdruck, der die Menge Q darstellt
$\mathsf{pre}_\forall(Q)$	Menge der universellen Vorgängerzustände der Menge Q
$\mathsf{pre}_\exists(Q)$	Menge der existentiellen Vorgängerzustände der Menge Q
$\overline{Q}$	Komplement der Menge Q
Q^{Ac}	Menge der erreichbaren Zustände eines Automaten
Q^{Co}	Menge der co-erreichbaren Zustände eines Automaten
Q^{Tr}	Menge der getrimmten Zustände eines Automaten
q^0	Initialzustand
$\mathcal{R}$	Transitionsrelation einer Kripke-Struktur
$\mathcal{S}$	Zustandsmenge einer Kripke-Struktur
Σ	Menge der Ereignisse (Alphabet) eines Automaten
Σ_c	Menge der steuerbaren Ereignisse eines Automaten
Σ_u	Menge der nicht steuerbaren Ereignisse eines Automaten
$\mathsf{suc}_\forall(Q)$	Menge der universellen Nachfolgerzustände der Menge Q
$\mathsf{suc}_\exists(Q)$	Menge der existentiellen Nachfolgerzustände der Menge Q
$\mathsf{supC}(K)$	größte steuerbare Teilsprache von K
$\mathsf{Tr}(\mathcal{A})$	getrimmte Komponente des Automaten $\mathcal{A}$
U	Zeitoperator in temporallogischen Ausdrücken: solange, bis
$\underline{\mathsf{U}}$	Zeitoperator in temporallogischen Ausdrücken: strenge Variante von U
u_b	verbotene Zustände
u_{bn}	verbotene oder nicht co-erreichbare Zustände
u_{bv}	verbotene oder nicht co-erreichbare Zustände, die keiner VVK angehören
u_c	co-erreichbare Zustände
u_{cg}	co-erreichbare und gleichzeitig gute Zustände
u_g	gute Zustände
u_n	nicht co-erreichbare Zustände
u_v	nicht co-erreichbare Zustände, die keiner VVK angehören
$\mathcal{V}, \mathcal{V}^x, \mathcal{V}^y$	Mengen boolescher Variablen
X	Zeitoperator in temporallogischen Ausdrücken: nächster Zustand
x, x_i, y, y_i	boolesche Variablen zur Beschriftung von Zuständen
$\neg x, \overline{x}$	Negation der booleschen Variable x
$x \wedge y, xy$	Konjunktion der booleschen Variablen x und y
$x \vee y$	Disjunktion der booleschen Variablen x und y
x_b	Variable zur Beschriftung verbotener Zustände
x_m	Variable zur Beschriftung markierter Zustände
z	Zustände der Lösung des verallgemeinerten Überwachersyntheseproblems

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die Geschichte unserer wissenschaftlichen Fortschritte zeigt, dass der Mensch nur dann in der Lage ist, komplexe Sachverhalte zu verstehen, wenn er ihre Struktur durchschaut und sie in kleinere, einfachere Einheiten zerlegt hat. In dem Augenblick, in dem ein komplexeres Ergebnis erzielt wird, geschieht dies gerade deshalb, weil es durch die Anstrengungen eines oder mehrerer Menschen über Jahre oder auch über Generationen hinweg nicht mehr komplex, sondern einfach geworden ist. So gesehen, ist der Mensch nur in der Lage, einfache Dinge zu tun. Deshalb ist die Geschichte der Wissenschaft auch die Geschichte der Entwicklung ihrer Werkzeuge. Sie fassen mühsam erarbeitetes Wissen zu Hilfsmitteln zusammen, mit denen Lösungen leichter gemacht werden können. Werkzeuge sind unabdingbar, weil gleichzeitig Ergebnis und Quelle des Fortschritts.

Auch diese Arbeit handelt von einem Werkzeug. Es dient dazu, rechnergestützte Systeme zu entwerfen. Der Systementwurf ist eine komplexe Aufgabe, und auch das hier vorgestellte Verfahren greift auf über Jahrzehnte angesammeltes Wissen zurück, und zwar auf die Überwachersynthese und die μ-Kalkül-basierte Modellprüfung. Die Arbeit stellt beide Ansätze vor und zeigt, wie sie sich gegenseitig ergänzen und zu einem neuen, mächtigeren Werkzeug verschmelzen lassen.

In dieser Einleitung wird zunächst erklärt, welche Systeme behandelt werden können und welche Probleme ihr Entwurf mit sich bringt. Danach werden die Überwachersynthese und die Modellprüfung auf informale Weise vorgestellt. Dadurch wird erkennbar, welche Lösungen notwendig sind, um den Systementwurf zu erleichtern. Darüber hinaus wird insbesondere im Fall der Modellprüfung Wert auf die geschichtliche Entwicklung gelegt. Diese erlaubt es, die vorliegende Arbeit in den Rahmen vorangegangener Werke, ohne die sie nicht hätte zustande kommen können, einzuordnen. Schließlich wird die Gliederung der restlichen Arbeit vorgestellt.

1.1 Kontinuierliche und diskrete Systeme

Das Wort *System* hat je nach Kontext verschiedene Bedeutungen und zählt zu jenen Wörtern, die man leichter als Grundkonzept intuitiv versteht, als formal definiert. Dennoch lässt sich allgemein sagen, ein System ist ein begrenzter Teil des Universums, der mit seiner Außenwelt in Wechselwirkung steht. Unter diese Beschreibung fallen die unterschiedlichsten natürlichen und künstlichen Vorrichtungen. Ziel dieses Abschnittes ist es, die für diese Arbeit relevanten Systeme hervorzuheben. Die Grundlage der gewählten Darstellung bildet die Klassifikation von Cassandras und Lafortune [16].

Bei der Beschreibung eines Systems werden verschiedene Größen angegeben, die sich im Laufe der Zeit verändern können. Beispiele hierfür sind die Position eines Planeten auf seiner Laufbahn oder die Anzahl der verbleibenden Teile in einem Lager. Jedem Wert, den eine solche Größe annehmen kann, wird ein *Zustand* zugeordnet. Die Menge dieser Werte bildet den *Zustandsraum* $\mathcal{S}$ des Systems, so dass sein Verhalten durch eine Funktion $f : t \to \mathcal{S}$ der Zeit t modelliert werden kann.

Die Kontinuität des Zustandsraumes spielt für die Klassifikation der Systeme eine entscheidende Rolle. Systeme wie der Planet im obigen Beispiel werden in der Regel durch kontinuierliche Funktionen beschrieben, die in einem gegebenen Bereich der reellen Zahlen jeden Wert annehmen können. Außerdem sind sie *zeitgesteuert*, d.h., der Zeitfluss allein genügt, um sie von einem Zustand in den nächsten zu überführen. Diese

Eigenschaften haben viele Systeme, die mit Hilfe von physikalischen Gesetzen und Differentialgleichungen erfasst werden. Im Gegensatz dazu kann der Zustand des o.g. Lagers nur ganze Werte annehmen, wodurch sich ein diskreter Zustandsraum ergibt. Die Funktion f weist dann abrupte, stufenartige Veränderungen auf, die eine andere Herangehensweise erfordern. Außerdem ist der Verlauf der Zeit allein nicht ausreichend, um einen Zustandswechsel herbeizuführen. Vielmehr wird hierfür ein Ereignis benötigt, so dass von *ereignisgesteuerten* Systemen die Rede ist. Im Falle des Lagers wird z.B. der Zustand durch die Entnahme oder das Hinzufügen von Teilen verändert. Der Vergleich von Systemen mit kontinuierlichem bzw. diskretem Zustandsraum wird in Abbildung 1.1 verdeutlicht.

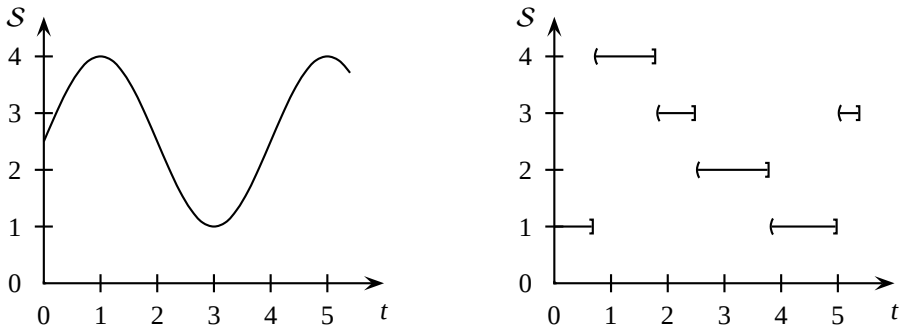


Abbildung 1.1: Kontinuierlicher bzw. diskreter Zustandsraum

Diese Arbeit befasst sich ausschließlich mit ereignisgesteuerten Systemen mit diskreten Zustandsräumen. Diese Systeme können durch weitere Aspekte noch genauer charakterisiert werden. So ist z.B. die Zeit kontinuierlich, da die Ereignisse, welche die Zustandswechsel verursachen, zu jedem beliebigen Zeitpunkt auftreten können. Sie sind *dynamisch*, womit gemeint ist, dass der Zustand, der nach dem Eintreten eines Ereignisses eingenommen wird, nicht nur von dem Ereignis selbst, sondern auch von dem aktuellen Zustand abhängt. Ferner wird das Interesse auf *deterministische* Systeme beschränkt, d.h., Systeme, deren Reaktion auf ein bestimmtes Ereignis in einem gegebenen Zustand eindeutig bestimmt ist. Es wird weiter angenommen, die Systeme seien *zeitinvariant*, d.h., die Reaktion auf ein Ereignis in einem gegebenen Zustand bleibt nach beliebig vielen Versuchen stets die gleiche. Damit wird von einer eventuellen Alterung oder von Fehlern in dem System abstrahiert. Aufgrund der genannten Eigenschaften werden die hier behandelten Systeme *diskrete ereignisgesteuerte Systeme*, kurz diskrete Systeme, genannt.

In diese Systemklasse fallen insbesondere informationsverarbeitende Systeme, die ja unseren Alltag in großem Maße beeinflussen. Die wachsende Komplexität rechnerbasierter Systeme verschärft unsere Abhängigkeit von automatischen Steuerungen. Aufgrund der sehr großen Zustandsräume, die durch die Nebenläufigkeit der darin ablaufenden Prozesse entstehen, kann die korrekte Funktion eines solchen Systems in der Regel nicht durch reines Testen gewährleistet werden. Es ist deshalb naheliegend, nach Entwurfsmethoden zu suchen, die sich von vorn herein an dem Ziel Korrektheit der Funktion orientieren. Obwohl das Interesse an diskreten Systemen dadurch stark zugenommen hat, konnte bisher kein Universalwerkzeug für ihre Handhabung gefunden werden, wie es in etwa die Analysis für die kontinuierlichen Systeme ist. Vielmehr sind im Laufe der Zeit Modelle und Theorien mit unterschiedlichen Schwerpunkten entstanden, wie z.B. Petrinetze, Markovketten, die Warteschlangentheorie, Prozessalgebren, die Max-Plus-Algebra, der μ-Kalkül, Temporallogiken, endliche Automaten und die Sprachentheorie.

Die vorliegende Arbeit befasst sich mit zwei Ansätzen, die in den letzten Jahrzehnten erfolgreich in der Handhabung wichtiger Aspekte diskreter Systeme gewesen sind: die Überwachersynthese und die auf dem μ-Kalkül basierende Modellprüfung. Beiden gemeinsam ist die formale Darstellung der Systeme. Demzufolge sind die Ergebnisse immer dann fehlerfrei, wenn die informalen Angaben, aus denen die formale Eingabe entsteht, richtig interpretiert und in die formale Beschreibung übersetzt werden. Ansonsten unterscheiden sich beide Verfahren sowohl in der Herkunft, als auch in der Modellierung der Systeme und in der Zielsetzung bei der Problemstellung, wodurch sich unterschiedliche Vor- und Nachteile ergeben. In dieser Arbeit wird die Überwachersynthese mit Hilfe des μ-Kalküls verallgemeinert. In dem entstehenden Ansatz ergänzen sich beide Verfahren gegenseitig, so dass die unterschiedlichen Vorteile gemeinsam genutzt werden können und wesentliche Nachteile wegfallen.

1.2 Systeme und Systemeigenschaften

Andere Sichtweisen über die hier behandelten Systeme, die zu einem besseren Verständnis der Arbeit beitragen, beziehen sich auf die Art und Weise, in der ein Benutzer mit ihnen interagiert, sowie auf die Eigenschaften, die in Spezifikationen anzutreffen sind. Die folgenden Einteilungen wurden aus der Systemklassifikation von Schneider [75] entnommen, wobei die unten stehende Klassifikation der Eigenschaften auf Manna und Pnueli [61, 62] zurückzuführen ist.

Bezüglich der Kommunikation zwischen System und Benutzer werden folgende Unterscheidungen gemacht:

- *Transformationsbasierte Systeme* bekommen zu Beginn der Laufzeit bestimmte Eingabedaten, verarbeiten diese und beenden einen Lauf mit der Ausgabe eines Ergebnisses. Ein Beispiel dazu ist ein Compiler, der ein Quellprogramm übersetzt.
- *Interaktive Systeme* laufen solange, bis der Benutzer beschließt, die Sitzung zu beenden. Während der Laufzeit findet eine Wechselwirkung zwischen System und Benutzer statt, die allerdings dadurch gekennzeichnet ist, dass der Benutzer stets auf die Reaktion des Systems warten muss, bevor er neue Eingaben machen kann bzw. diese von dem System akzeptiert werden. Beispiele dazu sind Spiele, in denen der Benutzer einen Zug macht und dann den Gegenzug des Systems abwartet.
- *Reaktive Systeme* laufen wie interaktive Systeme beliebig lang, sind jedoch zusätzlich in der Lage, zu jeder Zeit auf Anforderungen des Benutzers bzw. ihrer Umgebung zu antworten. Beispiele sind Steuerungen für Systeme, in denen Ereignisse wie der Ablauf eines Zeitglieds oder das Ansprechen eines Sensors jederzeit eintreten können und die eine angemessene Reaktion erfordern.

Die Eigenschaften, die in Systemen auftreten, werden wie folgt klassifiziert:

- *Sicherheitseigenschaften* formulieren Einschränkungen, die während der gesamten Laufzeit eingehalten werden müssen. Sie werden auf Systemzustände zurückgeführt, die niemals betreten werden dürfen, wie z.B. der Zustand, in dem die Ampeln an zwei sich kreuzenden Straßen gleichzeitig grün leuchten.
- *Lebendigkeitseigenschaften* stellen die Erreichbarkeit bestimmter Zustände eines Systems sicher. Dabei wird nicht festgelegt, wie lange es dauern kann, bis ein gegebener Zustand erreicht wird. Es spielt auch keine Rolle, wie oft zu diesem Zustand zurückgekehrt werden kann. Ein Beispiel dafür ist die Forderung, dass die Initialisierung eines Systems irgendwann beendet ist, damit der Normalbetrieb aufgenommen werden kann.
- *Fortdauereigenschaften* beziehen sich auf Merkmale die, einmal erreicht, stabil bleiben. Ein Beispiel dazu ist ein Zwischenlager für gefertigte Teile in einer Manufaktur, das ab einer bestimmten Phase der Produktion nicht mehr leer werden darf.
- *Fairnesseigenschaften* fordern, dass Zustände mit bestimmten Eigenschaften stets erreichbar bleiben müssen, unabhängig davon, wie oft sie betreten werden. Diese Eigenschaften sind besonders für die Spezifikation reaktiver Systeme nützlich. Damit kann z.B. ausgedrückt werden, dass sich ein Benutzer bei einem System beliebig oft anmelden können soll.

1.3 Der Entwurf diskreter Systeme

Die Bedingung für den korrekten Entwurf eines diskreten Systems lässt sich auf abstrakter Ebene relativ einfach formulieren. Es genügt, dass die Spezifikation für jeden erreichbaren Zustand und jedes Ereignis, das dort stattfinden kann, eine angemessene Reaktion vorsieht. Diese Aufgabe scheint zunächst nicht besonders schwierig zu sein. Sie ist es auch nicht, solange die Anzahl der zu berücksichtigen Zustände klein bleibt. Aber gerade dies ist in der Praxis normalerweise nicht der Fall.

Dies mag zunächst wie eine Ausnahme klingen, bestehen doch diskrete Systeme meistens aus mehreren Komponenten, die schon wegen des anfangs angesprochenen Bedürfnisses nach Einfachheit nur wenige Zustände

haben. Beispiele dafür sind die einzelnen Geräte in einer Fertigungszelle oder die Schnittstellen zwischen einer Steuerung und den von ihr bedienten Systemteilen. In jedem dieser Teile treten Ereignisse auf, und auch die Zahl der Ereignisse ist in der Regel nicht groß. Was den Zustandsraum in unerwartete Größenordnungen bringt ist erst die Tatsache, dass es einen Unterschied macht, in welcher Reihenfolge die verschiedenen Ereignisse eintreten. Die Anzahl der möglichen Permutationen nimmt exponentiell mit der Länge der Ereignisfolgen zu. Was zunächst die Ausnahme zu sein scheint, ist eher die Regel. Dieses Problem ist als *Zustandsexplosion* bekannt und kann Zustandsräume hervorbringen, deren Größenordnung die Anzahl der Atome im Universum übertrifft.

Aus diesem Grund wurden formale Methoden, die sich zum Ziel setzten, *alle* erreichbaren Zustände eines Systems zu behandeln, oft sehr skeptisch betrachtet. Wie im Laufe dieser Arbeit noch erläutert wird, hat man auch für dieses Problem ein relativ effektives Mittel gefunden. Seitdem gewinnen formale Methoden an Bedeutung, denn nur mit ihnen kann die Lösung der Aufgabe gelingen, für jedes Ereignis in jedem Zustand die richtige Reaktion vorzusehen.

1.4 Modellprüfung und Überwachersynthese

Moderne Verifikationsmethoden wie die temporallogische Modellprüfung gestatten es, eine gegebene Spezifikation auf gewisse Eigenschaften zu prüfen. Um dies zu erreichen, muss ein Modell des zu entwerfenden Systems erstellt werden. Ein Algorithmus bestätigt dann die erwarteten Eigenschaften bzw. liefert ein Gegenbeispiel dazu, falls sie nicht zutreffen. Ist dies der Fall, muss der Entwurf angepasst und der Prozess wiederholt werden. Das Ziel ist es, nach einigen Iterationen ein Modell zu erreichen, das alle Eigenschaften erfüllt. Aus dieser Beschreibung geht hervor, dass die Modellprüfung sich nicht mit der Modellierung an sich beschäftigt. Dadurch wird dem Entwickler eine Aufgabe überlassen, deren Schwierigkeitsgrad nicht zu unterschätzen ist. Zum Beispiel könnte die informale Spezifikation eines Kommunikationsprotokolls vorschreiben, dass bei einer Verbindung über ein verlustbehaftetes Medium die übertragenen Meldungen weder verloren gehen noch dupliziert werden dürfen. Diese Vorschrift ist nicht besonders hilfreich wenn es darum geht, die Regeln festzulegen, aus dem das Protokoll letztendlich hervorgeht. Sie hilft auch nicht viel weiter, wenn ein Gegenbeispiel für einen Versuch gefunden wird. Im Idealfall sollte die Spezifikation des Protokolls von einem Werkzeug generiert werden, das die genannten Anforderungen entgegennimmt und entweder ein korrektes Modell ausgibt oder die Anforderungen zurückweist, wenn sie sich nicht realisieren lassen. Ein solcher Ansatz ist aber in der Verifikationsliteratur nicht zu finden.

Ein Verfahren, das einen Schritt in die genannte Richtung geht, ist die Überwachersynthese. Das zu steuernde System und eine Spezifikation für dessen gewünschtes Verhalten werden als endliche Automaten dargestellt. Der erste dieser Automaten deckt alle physikalisch möglichen Ereignisfolgen ab und beschreibt auf diese Weise sowohl wünschenswertes als auch unerwünschtes Systemverhalten. Der zweite Automat beschreibt eine Einschränkung des physikalisch möglichen Verhaltens. Der Gedanke dabei ist, das System mit Hilfe eines solchen Automaten zur Laufzeit so einzuschränken, dass nur noch die gewünschten Aktionen stattfinden können. Im Gegensatz zu einer aktiven Steuerung, die zwischen möglichen Aktionen wählt und Befehle an das System abgibt, teilt dieser Automat dem System lediglich mit, welche Befehle erlaubt sind, und welche nicht. Ein Automat, der diese Rolle übernimmt, wird deshalb *Überwacher* genannt. Der Überwacher läuft parallel zum System, verfolgt die dort eintretenden Ereignisse und wechselt entsprechend seinen Zustand. Die erlaubten Ereignisse sind stets die, die in dem aktuellen Zustand des Überwachers vorgesehen sind.

Dabei wird berücksichtigt, dass die Ereignisse bezüglich ihrer Steuerbarkeit unterschiedlicher Natur sein können. Befehle, die z.B. bestimmte Vorrichtungen ein- oder ausschalten, Meldungen in einem Kommunikationssystem versenden oder Zeitglieder starten, können bei Bedarf von dem Überwacher verboten werden, um das Betreten unerwünschter Zustände zu verhindern. Im Gegensatz dazu lassen sich in der Regel Meldungen des Systems an die Steuerung, wie z.B. das Ansprechen eines Sensors, das Eintreffen einer Meldung in einem Kommunikationssystem oder der Ablauf eines Zeitglieds nicht beeinflussen. Dementsprechend wird zwischen *steuerbaren* und *nicht steuerbaren* Ereignissen unterschieden.

Da der Formulierung einer Spezifikation keine weiteren Grenzen gesetzt werden, ist es natürlich möglich, Forderungen zu stellen, die sich nicht realisieren lassen – z.B., dass ein Zeitglied in einem bestimmten Zustand

nicht ablaufen soll, wenn es aber bereits gestartet ist. Allerdings muss es sich in einem solchen Fall nicht unbedingt um einen groben Spezifikationsfehler handeln. Durch die Zustandsexplosion können auch Spezifikationen Fehler enthalten, die auf den ersten Blick einen korrekten Eindruck machen. In der Tat liegt genau hier die Ursache vieler Fehler, die in der Testphase eines Systems unbemerkt bleiben und erst nach dessen Freigabe ans Tageslicht treten. Wird ein System nur nach gesundem Menschenverstand spezifiziert und getestet, kann es immer passieren, dass Zustände erreichbar bleiben, in denen unerwartete Ereignisse auftreten, weil die Anzahl der möglichen Ereignisfolgen einfach zu groß ist, als dass sie alle in der Testphase abgedeckt werden könnten. Die Folge sind unerwünschte Ausnahmesituationen und Systemabstürze.

Die Stärke der Überwachersynthese liegt genau in der Fähigkeit, mit solchen unvollkommenen Spezifikationen umzugehen. Ist dies der Fall, kann mit Hilfe eines Synthesealgorithmus, der die Systembeschreibung und die Spezifikation als Eingabe nimmt, ein dritter Automat berechnet werden, der die größte realisierbare Untermenge der Spezifikation darstellt. Der Gedanke dabei ist, dass dieses Ergebnis als Überwacher eingesetzt werden kann, solange die Einschränkungen, die durch den Syntheseprozess entstehen, nicht zu stark ausfallen. Diese Berechnung erhält den Namen *Überwachersynthese* und bildet einen wichtigen Vorteil gegenüber der Modellprüfung. Während diese es nur erlaubt, ein vom Anwender erstelltes Modell auf bestimmte Eigenschaften zu prüfen, ist die Synthese in der Lage, ein Modell für das System zu generieren.

Der Syntheseprozess hat auch einen Nachteil, nämlich die Tatsache, dass es nicht immer leicht festzustellen ist, ob die Einschränkungen, die der ursprünglichen Spezifikation auferlegt werden, nicht zu weit gehen. So könnte z.B. ein Kommunikationsprotokoll so ausfallen, dass nur eine begrenzte Anzahl von Meldungen verschickt wird, um die Verluste in Grenzen zu halten. Theoretisch sieht die Formulierung des Syntheseproblems zwar die Möglichkeit vor, ein kleinstmögliches Verhalten für das System vorzuschreiben, so dass ein Ergebnis, das diese Forderungen nicht erfüllt, verworfen werden könnte. Damit entsteht jedoch die Frage, wie dieses Verhalten zu spezifizieren ist. Wäre es bekannt, könnte es per Definition als eine Lösung dienen, von der angenommen werden muss, dass sie nicht bekannt ist. In der Praxis verfügt das Verfahren deshalb nicht über ein Kriterium zur Unterscheidung zwischen zulässigen und nicht zulässigen Syntheseergebnissen.

In dieser Arbeit wird gezeigt, dass die o.g. Nachteile der Überwachersynthese und der Modellprüfung durch eine Verschmelzung beider Verfahren weitgehend beseitigt werden können. Zunächst kann die Synthese der Modellprüfung bei der Erstellung des Systemmodells behilflich sein. Dies ist besonders dann zutreffend, wenn die informalen Anforderungen, die dem Anwender zu Beginn des Entwurfs vorliegen, nicht auf eine Implementierung schließen lassen. Wie auch die Beispiele in dieser Arbeit zeigen, lassen sich solche Formulierungen oft direkt auf endliche Automaten übertragen, ohne dass sich der Anwender um alle möglichen Ereignisfolgen im System kümmern muss. Im Gegenzug kann die Modellprüfung der Überwachersynthese die Frage beantworten, ob bestimmte Eigenschaften der Spezifikation den Syntheseprozess überstanden haben. Ein noch besseres Ergebnis ist die in Kapitel 6 erzielte Verallgemeinerung der Überwachersynthese, die es erlaubt, die zu prüfenden Eigenschaften bereits in den Syntheseprozess mit einzubeziehen, wodurch eine Überprüfung des Syntheseergebnisses hinfällig wird. Darüber hinaus lässt die verallgemeinerte Überwachersynthese auch Spezifikationen mit Fortdauer- und Fairnesseigenschaften zu, während der ursprüngliche Ansatz auf Sicherheits- und Lebendigkeitseigenschaften beschränkt ist.

Im folgenden Abschnitt werden historische Aspekte beleuchtet, die es ermöglichen, die vorliegende Arbeit in den Kontext der Verifikation einzuordnen. Dabei spielt im Wesentlichen die Entwicklungsgeschichte der Modellprüfung eine Rolle, weil ihr Prinzip bereits in Jahrhunderte alten Arbeiten zu erkennen ist. Die Geschichte der Überwachersynthese ist im Vergleich dazu sehr kurz und bedarf keiner tiefgründigen Erklärung. Sie wird in die Erläuterungen in den Abschnitten 2.2 und 2.7 mit einbezogen.

1.5 Eine kurze Geschichte der Verifikation

Der Gedanke, die Richtigkeit jeder denkbaren Aussage maschinell zu überprüfen, reicht bis Descartes (1596–1650) zurück. Auch Leibniz (1646-1716) schlug ein ambitioniertes Forschungsprojekt vor, das sich seinen Einschätzungen nach über drei Jahrhunderte erstrecken sollte. Sein Ziel war es, eine formale Sprache zur Beschreibung mathematischer Systeme, die *lingua characteristica*, zu entwickeln. Darüber hinaus sollte der *calculus*

ratiocinator entstehen, mit dem es gelingen sollte, Aussagen in der lingua characteristica automatisch zu überprüfen. Die Entwicklung dieser Gedanken lässt sich bis in die heutige Zeit verfolgen. Die hier gegebene Übersicht beschränkt sich allerdings auf die Tatsachen, die zur Motivation und zur Einbettung der Modellprüfung in den Kontext der Verifikation am wichtigsten sind. Eine ausführliche Schilderung der gesamten Entwicklung findet der Leser z.B. in dem Werk von Schneider [75].

Die ersten Programme waren transformationelle Systeme. Wie Matis [63] und Ifrah [42] festhalten, dienten sie u.a. der automatischen Lösung linearer Gleichungssysteme oder der Erstellung von Tabellen für die Astronomie, die Ballistik und die Luftfahrttechnik, also der Bewältigung zwar intensiver, aber prinzipiell einfacher Rechenaufgaben. Fehler wurden durch schrittweises Testen lokalisiert und beseitigt. Der Aufwand, um auf diese Weise ein fehlerfreies Programm zu bekommen und die dadurch gewonnenen Vorteile standen dabei oft noch in einem günstigen Verhältnis. Als sich aber durch technische Fortschritte immer anspruchsvollere Anwendungsmöglichkeiten für die Rechner ergaben, übertraf die Anzahl der wünschenswerten Testfälle bald alles wirtschaftlich oder gar physikalisch Durchführbare, wie z.B. Myers in [64] belegt. Es ist deshalb nicht überraschend, dass der Wunsch, die Korrektheit eines Programms mit anderen Mitteln als durch reines Testen nachzuweisen, bereits aus der Zeit der ersten kommerziellen Rechner, also aus den 1950er Jahren, stammt.

Unabhängig von dieser Problematik war es damals ebenfalls naheliegend, die neuen Maschinen für automatische Beweisführungen etwa in der Algebra einzusetzen. Der erste maschinelle Beweis wurde 1954 von M. Davis mit Hilfe eines Johniac-Rechners erbracht. Dieser konnte zeigen, dass die Summe zweier gerader Zahlen ebenfalls eine gerade Zahl ist. Diese Anwendungen hatten zunächst wenig mit der Überprüfung von Programmen zu tun, obwohl es möglich war, die Korrektheit von Algorithmen zumindest teilweise sicherzustellen. Somit könnte man hier bereits von Verifikation von Spezifikationen sprechen. Da es aber noch keine formale Übersetzung eines Algorithmus in ein Programm gab, war ein korrekter Algorithmus noch keine Garantie für ein korrektes Programm. Der entscheidende Schritt für die Entstehung der heutigen *Verifikation* von Programmen und Systemen kam in den 1960er Jahren durch die Arbeiten von Floyd [35] und Hoare [39]. Sie stellten speziell auf Programme zugeschnittene Logiken und Beweismethoden vor, mit denen die Programmtexte selbst überprüft werden konnten.

Im Idealfall sollte ein solches Verifikationsverfahren voll automatisierbar sein. Es wären dann *Theorembeweiser* denkbar, die – angenommen, sie wären fehlerfrei – jedes Programm verifizieren könnten. Ein solcher Beweiser müsste lediglich festgelegte Schlussregeln auf eine Menge bekannter Axiome in der richtigen Reihenfolge anwenden, um jede erdenkliche Wahrheit zu produzieren. Allerdings basiert die Beweisführung auf Peanos Axiomen der Arithmetik ganzer Zahlen, insbesondere auf dem Prinzip der Induktion [siehe z.B. 75, Kap. 6]. Wie Gödel in seinen Arbeiten über Widerspruchsfreiheit und Entscheidbarkeit [38] jedoch zeigen konnte, muss es in jedem System, das mächtig genug ist um die endlichen Zahlen wie die Peano-Arithmetik zu beschreiben, Aussagen geben, die zwar wahr sind, sich aber nicht beweisen lassen. Deshalb lassen sich Theorembeweiser nicht voll automatisieren. In der Regel werden sie als Hilfswerkzeuge betrachtet, die dem Benutzer die Arbeit zwar erheblich erleichtern, aber nicht ohne menschliche Steuerung auskommen. Dadurch bleibt das Verfahren einem relativ kleinen Kreis von Experten vorbehalten, und der Zeitaufwand für einen Verifikationsvorgang, der durchaus Tage oder gar Monate dauern kann, ist in vielen Anwendungsbereichen wirtschaftlich nicht vertretbar. Dementsprechend ist die Anwendung der beweisbasierten Verifikation relativ selten, obwohl sie einen hohen Stellenwert in der Informatik eingenommen hat.

Die Verwendung der ganzen Zahlen – also einer unendlichen Menge – für die Darstellung der Variablen bedeutet außerdem, dass die Verifikation nicht auf die endliche Bitbreite der Rechner eingeht, auf denen das zu verifizierende Programm später laufen soll. Dies hat Vor- und Nachteile. Vorteilhaft ist, dass die Modellierung vereinfacht wird, wenn nicht auf physikalische Eigenschaften der Rechner eingegangen werden muss. Dies ist zum Teil auch nur bedingt möglich, z.B. wenn ein Programm auf verschiedenen Rechnern laufen soll und noch keine genauen Angaben über diese vorliegen. In solchen Fällen möchte man davon abstrahieren und die eigentliche Prozedur, die das Programm darstellt, verifizieren. Zum Beispiel sei ein neuer Algorithmus zur Sortierung von Listen gegeben, der sehr effizient, dafür aber sehr komplex ist. Dann ist es zunächst am wichtigsten nachzuweisen, dass er jede Liste korrekt sortiert, auch wenn es theoretisch Probleme geben kann, falls eine Liste doch einmal zu groß werden sollte. Der Nachteil besteht darin, dass solche Probleme nicht erkannt werden. So

hätte zum Beispiel die Software der Rakete Ariane 5, die 1996 in Folge eines Registerüberlaufs abstürzte [79], einen derartigen Verifikationstest bestanden.

In Anbetracht dieses Nachteils ist nun die Frage berechtigt, ob sich das oben beschriebene Verfahren an die endliche Bitbreite eines realen Rechners anpassen lässt. Würde es gelingen, an Stelle der ganzen Zahlen eine endliche Zahlenmenge einzusetzen, wäre dies auf natürliche Weise gegeben. Außerdem wäre jedes Problem entscheidbar. Diese Vorteile müssten in vielen Fällen die zusätzlichen Schwierigkeiten bei der Modellierung ausgleichen. Das Problem mit diesem Ansatz ist, dass sich die Peanoschen Axiome nicht auf die für diese Darstellung notwendige Modulo-Arithmetik übertragen lassen. Damit wird aber die Beweisführung stark eingeschränkt. Es ist zwar noch möglich, spezifizierte Eigenschaften nachzuweisen, jedoch sinnvoller, Ansätze zu wählen, die sich leichter automatisieren lassen bzw. geringere Anforderungen an den Benutzer stellen. Deshalb bleibt die beweisbasierte Verifikation jenen Systemen zugeordnet, die so modelliert werden, als gäbe es keine Grenze für die Bitbreite ihrer Register. Für endliche Systeme sind insbesondere temporallogische Ansätze einfacher.

Temporallogiken haben ihre Wurzeln in der Philosophie und dienten ursprünglich der Betrachtung von Zeitausdrücken in der menschlichen Sprache [41]. In den 1970er Jahren wurden diese Logiken erstmals für die Spezifikation von Rechnersystemen eingesetzt. Burstall [15] und Kröger [50] konnten damit zunächst Eigenschaften sequentieller Programme beschreiben, während Pnueli die Betrachtung nebenläufiger Programme gelang [66]. Diese Verfahren waren jedoch nicht automatisiert und konnten sich deshalb in ihrer ursprünglichen Form nur selten durchsetzen. Pnuelis Ansatz wurde allerdings zu Beginn der 1980er Jahre von Bochmann [7] sowie von Malachi und Owicki [59] der Verifikation sequentieller Schaltungen angepasst, wodurch sich der Begriff der Verifikation auch auf den Hardwareentwurf erstreckte. Noch wichtiger ist aber, dass der gleiche Ansatz sowohl von Clarke und Emerson [17] als auch von Quielle und Sifakis [68, 69] zu einem automatischen Verfahren weiterentwickelt wurde, das ab Anfang der 1980er Jahre als *Modellprüfung* (engl. *model checking*) bekannt wurde.

Die Automatisierung des Verfahrens beruht auf Erkenntnissen von Emerson und Clarke, die 1980 zeigten, dass die Korrektheit eines Programms auf die Berechnung von Fixpunkten monotoner Funktionen zurückgeführt werden kann [27]. Die Formalisierung dieser Fixpunktberechnungen durch Pratt [67] und Kozen [47, 48] sowie Arnold und Crubille [1] führte zur Entwicklung des *μ-Kalküls*, einer überaus mächtigen Logik zur Beschreibung von Fixpunkten. Der μ-Kalkül hat allerdings eine oft schwer verständliche Syntax. Deshalb war es naheliegend, nach einer besser lesbaren Beschreibung der Fixpunkte zu suchen. Hier kommen die Temporallogiken ins Spiel, zumal sich die Berechnungen temporallogischer Ausdrücke ebenfalls auf Fixpunktberechnungen reduzieren. Somit ist es möglich, an die Stelle vieler μ-Kalkül-Gleichungen temporallogische Ausdrücke zu setzen, die dasselbe bedeuten, aber für den Menschen leichter verständlich sind.

Da Temporallogiken jedoch immer nur ein Fragment des μ-Kalküls abdecken, entstand die Frage, welche Temporallogik für die Verifikation am geeignetsten sei. Einerseits sollten sich wichtige Eigenschaften in gut lesbaren Formeln ausdrücken lassen, andererseits sollte sich die Komplexität deren Auswertung in akzeptablen Grenzen halten. Die genannten Arbeiten von Clarke und Emerson sowie von Quielle und Sifakis benutzten die Logik CTL bzw. eine Untermenge davon, während die ursprünglichen Arbeiten von Pnueli auf LTL basierten[1]. Die Debatte zwischen Verfechtern beider Logiken erstreckt sich bereits auf über zwei Jahrzehnte [28, 34, 52, 56, 82]. Zunächst hat sich CTL zumindest im industriellen Anwendungsbereich einen deutlichen Vorsprung verschafft, obwohl sich manche wichtige Eigenschaften, z.B. starke Fairness, damit nicht ausdrücken lassen, sehr wohl aber in LTL.

Diesen Vorsprung hat CTL zwei Ergebnissen zu verdanken. Erstens haben Clarke, Emerson und Sistla 1986 gezeigt, dass die Zeit für die Auswertung von CTL-Formeln linear mit dem Produkt der Größe der Transitionsrelation des Systems und der Länge der Formeln ansteigt [18]. Dagegen benötigt die Auswertung von LTL-Formeln eine exponentielle Laufzeit [56]. Dies war aber noch nicht genug, um den Einzug der Modellprüfung in die Welt der Industrie zu gewährleisten, denn die Zustandsexplosion macht den Vorteil der linearen Laufzeit wieder zunichte. Es fehlte deshalb nicht an Kritikern, die der Modellprüfung, wenn überhaupt, eine höchst bescheidene Zukunft im akademischen Bereich voraussagten, wie z.B. in [19] nachzulesen ist. Das

[1] CTL und LTL stehen für *computation tree logic* bzw. *linear time logic*.

zweite Ergebnis, das CTL von anderen Logiken deutlich abhob, basiert auf der *symbolischen Darstellung* der Transitionsrelation des zu verifizierenden Systems mit Hilfe von binären Entscheidungsdiagrammen.

Binäre Entscheidungsdiagramme sind seit den Arbeiten von Lee in den 1950er Jahren bekannt [54]. Bryant entwickelte dafür 1986 eine kanonische Form [11], mit der boolesche Funktionen in eindeutiger und oft sehr kompakter Weise dargestellt werden können. Damit lassen sich Transitionsrelationen repräsentieren, deren explizite Darstellung die Speicherkapazität jedes noch so großen Rechners um mehrere Größenordnungen überschreiten würde. Auch die grundlegenden Mengenoperationen können auf symbolischer Ebene durchgeführt werden. Dies ermöglicht u.a. eine effiziente Breitensuche in der zu verifizierenden Transitionsrelation, was wiederum CTL zugute kommt. Dies wurde 1990 von zwei Forschergruppen, die unabhängig voneinander arbeiteten, erkannt: Die Arbeiten von Burch, Clarke, McMillan, Dill und Hwang [12, 13, 14] sowie die von Berthet, Coudert und Madre [4] führten die symbolische Darstellung der Zustandsräume mit Hilfe von binären Entscheidungsdiagrammen in die Verifikation ein. Damit ließen sich ab Beginn der 1990er Jahre mehrere Protokolle und Schaltungen verifizieren, wobei auch in bereits veröffentlichten Richtlinien noch subtile Fehler gefunden wurden [20].

Im Gegensatz zu CTL ist LTL auf eine Tiefensuche angewiesen, die durch die symbolische Darstellung nicht wesentlich beschleunigt wird. Moderne Ansätze gleichen dies allerdings durch eine Übersetzung der LTL-Formeln in Automaten wieder aus. Somit sind in den gängigen Verifikationswerkzeugen letztendlich beide Logiken vertreten. Des Weiteren wurden wegen der nicht ausreichenden Ausdrucksmöglichkeiten von CTL andere Logiken vorgeschlagen, wie z.B. CTL*, eine Obermenge von CTL und LTL. Die vorliegende Arbeit entspricht in dieser Hinsicht dem Stand der Technik, zumal sie nicht auf eine der speziellen Logiken, sondern auf dem μ-Kalkül basiert und dadurch alle temporallogischen Ausdrücke verwenden kann.

1.6 Gliederung

Die Gliederung der vorliegenden Arbeit in zwei Teile lässt bereits die Grenze zwischen der Aufarbeitung vorhandenen Wissens und den neuen Beiträgen zu der Theorie erkennen. Teil I besteht aus den Kapiteln 2 und 3 und stellt im Wesentlichen die Überwachersynthese und die Modellprüfung vor. Allerdings sind die Darstellung des μ-Kalküls in Kapitel 3 bzw. die der in Anhang B ausgelagerten Beweise bereits neu. Zwar werden hier keine neuen Fakten aufgedeckt, jedoch ist die Darstellung speziell auf die darauf folgenden Kapitel zugeschnitten. Der Aufwand dafür wird dadurch gerechtfertigt, dass es keine Quelle gab, in der die notwendigen Grundkenntnisse in passender Form zu finden gewesen wären. Einerseits wird der μ-Kalkül in den ursprünglichen Texten anders als heute üblich erklärt, andererseits würde eine direkte Übernahme der modernen Darstellung z.B. aus [75] das Ziel ebenso verfehlen, weil der μ-Kalkül dort als Untermenge einer allgemeineren Spezifikationssprache geführt wird. Stattdessen wird dem Leser eine kürzere Darstellung geboten, die eine schnellere Aneignung der notwendigen Grundkenntnisse ermöglicht und auch als Ausgangspunkt für weitere Arbeiten auf diesem Gebiet dienen kann.

Kapitel 2 führt die Überwachersynthese mit einem motivierenden Beispiel ein und geht schrittweise in die Formalisierung des Verfahrens über. Es werden das Überwachersyntheseproblem und zwei gängige Algorithmen zu dessen Lösung vorgestellt. Ein Vergleich zwischen den Komplexitäten dieser Lösungen bereitet den Kontext für die Einbettung einer neuen Lösung vor, die in Kapitel 5 vorgestellt wird. Es folgt eine Übersicht über die verschiedenen Varianten der Überwachersynthese, die im Laufe der Zeit aus dem Grundmodell von Ramadge und Wonham entstanden sind. Den Abschluss des Kapitels bildet eine Aufzählung verschiedener Verbesserungsmöglichkeiten, die im Verlauf dieser Arbeit behandelt werden.

In Kapitel 3 wird der μ-Kalkül vorgestellt. Dazu werden in Abschnitt 3.1 zunächst die Kripke-Strukturen erläutert und die symbolische Darstellung von Transitionsrelationen eingeführt. Abschnitt 3.2 ist dem flachen μ-Kalkül gewidmet, in dem alle Formeln, aber keine Gleichungssysteme abgedeckt werden. Unter anderem wird erkennbar, weshalb die Ermittlung von Zustandsmengen einer Transitionsrelation auf die Berechnung von Fixpunkten monotoner Funktionen mit Hilfe des Tarski-Knaster-Theorems zurückzuführen ist. Abschnitt 3.3 behandelt die Gleichungssysteme des vektoriellen μ-Kalküls. Hier liegt der Schwerpunkt auf der Verbindung zwischen den flachen Formeln und den Gleichungssystemen, aus der auch die Algorithmen hervorgehen, die

letztendlich für die neuen Lösungen des Überwachersyntheseproblems eingesetzt werden. Die Verbesserungsmöglichkeiten für die μ-Kalkül-basierte Modellprüfung werden in Abschnitt 3.4 zusammengefasst.

Teil II besteht aus den Kapiteln 4 bis 6 und stellt einen neuen Ansatz zur Überwachersynthese vor. In Kapitel 4 wird die konventionelle Überwachersynthese in den μ-Kalkül überführt. Da die Synthese auf endlichen Automaten, die Modellprüfung aber auf Kripke-Strukturen aufbaut, muss zunächst eine gemeinsame Darstellungsform der Systeme gefunden werden. In Abschnitt 4.1 wird zu diesem Zweck der Automat, der für die Synthese benötigt wird, in eine spezielle Kripke-Struktur übersetzt. Für diese Kripke-Struktur wird dann in Abschnitt 4.2 ein Gleichungssystem gefunden, dessen Lösung eine Zustandsmenge berechnet die, zurück auf den Automaten abgebildet, den Überwacher ergibt.

Kapitel 5 baut auf die Ergebnisse in Kapitel 4 auf, um zu einem neuen Synthesealgorithmus zu gelangen. Dieser wird in den Abschnitten 5.1 und 5.2 beschrieben. Der neue Algorithmus ist effizienter als die bisher bekannten, weil er für eine große Klasse praktischer Probleme eine lineare Laufzeit aufweist, während die herkömmlichen Lösungen quadratisch sind. Die neu entdeckte Problemklasse wird in Abschnitt 5.3 in den Kontext der bereits bekannten Klassen eingebettet.

Anschließend werden zwei Synthesebeispiele vorgestellt. Sie dienen dazu, die Modellierung realer Systeme und die Effizienzsteigerung des neuen Algorithmus zu veranschaulichen. Abschnitt 5.4 beschreibt eine Anwendung der Überwachersynthese zur Steuerung einer Telefonnummernauskunftzentrale. Das Beispiel stammt von Seidl et al. [77] und wird hier noch tiefgründiger als in deren Veröffentlichung analysiert. Dabei wird auch ein subtiler Fehler entdeckt, der sich allerdings mit den Mitteln der herkömmlichen Überwachersynthese nicht beheben lässt, weil die Spezifikation dazu eine Fairnessbedingung enthalten müsste. Der Fehler kann jedoch mit der verallgemeinerten Überwachersynthese in Kapitel 6 behoben werden. Abschnitt 5.5 beschreibt ein zweites Synthesebeispiel, in dem es darum geht, eine Gewinnstrategie für ein Zweipersonenspiel zu finden. Eine solche Gewinnstrategie ist eine Metapher für eine Steuerung, die einem System (dem Gegner) zwar alle nach den Regeln erlaubten Züge zugesteht, es aber gleichzeitig durch eine geschickte Auswahl der ihr zu Verfügung stehenden Möglichkeiten in einen bestimmten Zustand zwingen möchte. Die Skalierungsmöglichkeiten des Beispiels werden genutzt, um die Laufzeit und den Speicherplatzbedarf des alten und des neuen Algorithmus zu vergleichen.

Kapitel 6 greift ebenfalls auf die Ergebnisse in Kapitel 4 zurück und erstreckt die Klasse der zulässigen Spezifikationen auf alle μ-Kalkül-Ausdrücke. Dies wird dem Anwender durch die Einbeziehung temporallogischer Ausdrücke erleichtert, welche die gewünschten Eigenschaften in einer leichter verständlichen Syntax darstellen. Damit wird die Überwachersynthese in die Lage versetzt, Fortdauer- und Fairnesseigenschaften genauso wie die bisher zugelassenen Sicherheits- und Lebendigkeitseigenschaften zu behandeln. Abschnitt 6.1 enthält eine Reihe temporallogischer Ausdrücke, die in dem neuen Ansatz eingesetzt werden können. Abschnitt 6.2 stellt die Veränderungen an dem Gleichungssystem aus Kapitel 4 vor, die zu der gewünschten Verallgemeinerung führen. In den Abschnitten 6.3 und 6.4 wird gezeigt, wie die herkömmliche Überwachersynthese zum Sonderfall des neuen Ansatzes wird und wie Fairnessbedingungen in die Synthese einbezogen werden können. Abschnitt 6.5 nimmt das Beispiel mit der Telefonnummernauskunft aus Abschnitt 5.4 wieder auf und löst das dort entdeckte Problem mit Hilfe der verallgemeinerten Überwachersynthese. Das Beispiel zeigt, dass der neue Ansatz in der Lage ist, unfaire Zustände in einer gegebenen Spezifikation zu erkennen und ein System so zu steuern, dass nur noch faire Zustände betreten werden.

Schließlich wird in Abschnitt 6.6 der verallgemeinerte Ansatz mit drei ähnlichen Arbeiten verglichen. Der Vergleich baut auf der Tatsache auf, dass die Überwachersynthese sowohl auf das Erfüllbarkeits- als auch auf das Modellprüfungsproblem im μ-Kalkül zurückgeführt werden kann. Von diesen ist das Erfüllbarkeitsproblem am schwierigsten zu lösen, was sich auch in der Komplexität der einschlägigen Algorithmen widerspiegelt. Es stellt sich heraus, dass der in dieser Arbeit entstandene Ansatz die Überwachersynthese auf ein Modellprüfungsproblem zurückführt, während die anderen entweder das Erfüllbarkeitsproblem zugrunde legen oder aber Übersetzungsprozeduren ähnlicher Komplexität benötigen. Vor allem können die anderen Ansätze die Vorteile der symbolischen Darstellung nicht nutzen, was dem hier vorgestellten Verfahren einen entscheidenden Vorteil verschafft.

Kapitel 7 fasst die erzielten Ergebnisse noch einmal zusammen und weist auf noch teilweise ungeklärte Aspekte hin, die zum Thema weiterer Forschungsarbeiten werden könnten.

Anhang A enthält einen Beweis des Tarski-Knaster-Theorems. Letzterer entstand durch die Aufarbeitung des Beweises für dieses Theorem in [75]. Obwohl dadurch eine starke Ähnlichkeit zwischen beiden Beweisen entsteht, ist die Darstellung in Anhang A ausführlicher und besser in die restliche Arbeit eingegliedert. Da dieser Beweis der Schlüssel zu einem tiefgründigen Verständnis des μ-Kalküls und somit der gesamten Arbeit ist, wird er hier trotz dieser Ähnlichkeit wiedergegeben.

Anhang B enthält Beweise einiger Theoreme aus Kapitel 3. Sie wurden ausgelagert, um dem Leser, der sich zunächst einen Überblick über die gesamte Arbeit verschaffen möchte, einen schnelleren Weg durch dieses Kapitel zu weisen. Sie bilden jedoch die Grundlagen, die diese Arbeit überhaupt entstehen ließ.

Es folgen ein Literaturverzeichnis und ein Index. In diesem sind die Begriffe in aller Regel nach Substantiven sortiert. Zum Beispiel befindet sich der Eintrag für „steuerbares Ereignis“ unter „Ereignis, steuerbares“. Kursiv gedruckte Seitenzahlen verweisen auf die Stelle, an der der genannte Begriff definiert bzw. am besten erklärt wird.

Teil I

Grundlagen

Kapitel 2

Die Überwachersynthese

In diesem Kapitel wird die Überwachersynthese vorgestellt. Sie bildet den Kern des von Ramadge und Wonham [71, 72, 86] entworfenen Modells zur Handhabung diskreter, ereignisgesteuerter Systeme, das in der einschlägigen Literatur als Ramadge-Wonham-Modell bekannt ist. Außer den Originalquellen geben aktuellere und umfassendere Werke wie das Buch von Cassandras und Lafortune [16] oder Wonhams *Lecture Notes* [85] Einsicht in die theoretischen Grundlagen, die hier nur zum Teil wiedergegeben werden sollen.

Abschnitt 2.1 enthält ein motivierendes Beispiel. Dieses steht stellvertretend für größere Systeme, deren Zustandsraum leicht unübersichtlich wird und in Testverfahren aus Aufwandsgründen nicht vollständig durchlaufen werden kann. Ohne einen formalen Ansatz können dadurch subtile Fehler unentdeckt bleiben, die erst unter ungewöhnlichen Bedingungen zur Laufzeit ans Tageslicht treten. Weitere einfache Beispiele werden z.B. in [90] beschrieben. Abschnitte 2.2 und 2.3 beschreiben das Prinzip des Ramadge-Wonham-Modells und die Modellierung der verschiedenen Komponenten. Das Überwachersyntheseproblem und die gängigen Algorithmen zu dessen Lösung werden in den Abschnitten 2.4 und 2.5 vorgestellt. Abschnitt 2.6 erläutert Strategien für die Erzeugung von Spezifikationen. In Abschnitt 2.7 werden verwandte Arbeiten aufgezählt, und Abschnitt 2.8 erläutert Verbesserungsmöglichkeiten, an denen sich der Beitrag der vorliegenden Arbeit orientiert. Vorausgesetzt werden Grundlagen der Automatentheorie, wie z.B. die aus dem Werk von Hopcroft et al. [40].

2.1 Motivation

Folgendes Beispiel aus Wonhams *Lecture Notes* [85, p. 101] stellt die Problematik vor, um die es in dieser Arbeit geht. Es handelt von einer Fertigungszelle, deren Teilsysteme als Automaten modelliert werden. In der graphischen Darstellung der Automaten werden Initialzustände mit einem kleinen Pfeil, akzeptierende Zustände durch einen doppelten Rand gekennzeichnet. Die Rolle der akzeptierenden Zustände in den Automaten wird erst in Abschnitt 2.3 erklärt. Sie können vorerst ignoriert werden.

Beispiel 2.1 Eine Fertigungszelle besteht aus zwei Maschinen, M_A und M_B, und aus einem automatischen Transportwagen *ATW*, wie in Abbildung 2.1 dargestellt.

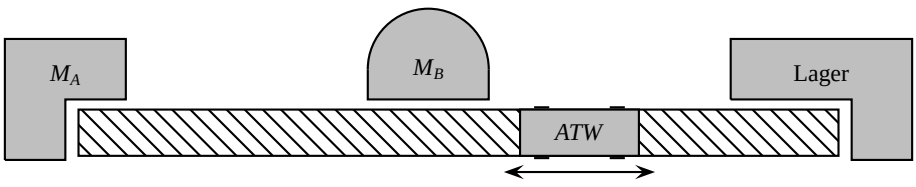


Abbildung 2.1: Die Fertigungszelle mit den Maschinen M_A und M_B und dem *ATW*

Maschine M_A produziert in Folge eines Befehls Start_A ein Teil des Typs *A*. Dieses wird von dem *ATW* abgeholt (Ereignis Lade_A). Danach ist M_A wieder frei und kann ein neues Teil produzieren. Der *ATW* gibt das Teil vom Typ *A* an Maschine M_B weiter (Ereignis Start_B) und ist danach wieder frei. Von M_B wird das Teil vom

Typ *A* zu einem Teil vom Typ *B* weiterverarbeitet. Wenn Maschine M_B fertig ist, wird das Teil vom Typ *B* von dem *ATW* abgeholt (Ereignis Lade_B), wonach M_B wieder frei ist und ein neues Teil vom Typ *A* entgegennehmen kann. Der *ATW* bringt das fertige Teil vom Typ *B* in ein Lager (Ereignis Lager) und ist danach wieder frei. Diese informale Beschreibung der Systemkomponenten lässt sich in die Automaten in Abbildung 2.2 übersetzen. Die Übersetzung ist recht intuitiv und zeigt somit die Umwandlung einer informalen in eine formale Beschreibung des Systems.

Die Ereignisse Start_A, Lade_A, Start_B, Lade_B, Lager lassen sich auf den Automaten verfolgen. Diese nehmen nach und nach die Zustände ein, die in den Abbildungen 2.2 bis 2.7 eingefärbt dargestellt sind.

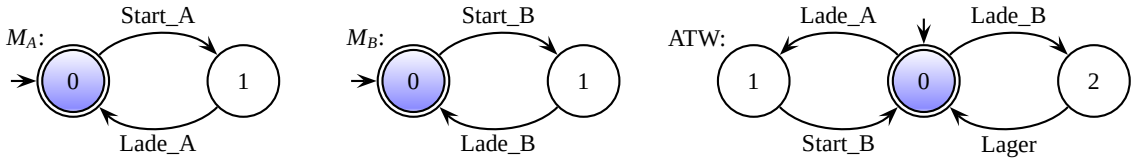


Abbildung 2.2: M_A, M_B und *ATW* im Initialzustand

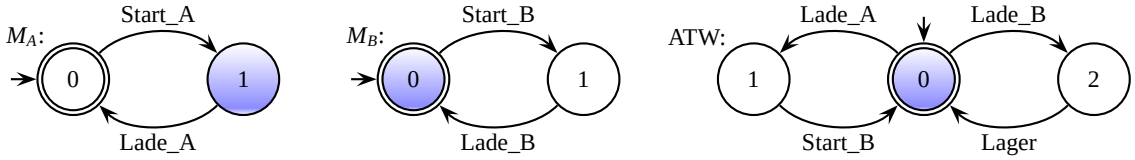


Abbildung 2.3: Nach Start_A

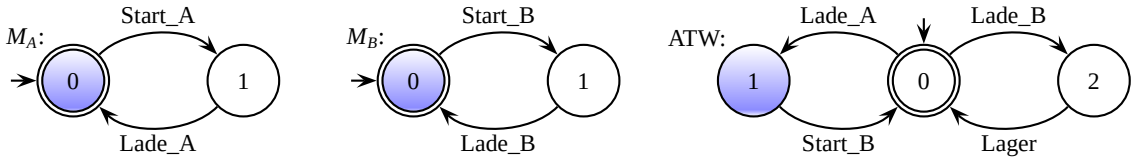


Abbildung 2.4: Nach Start_A, Lade_A

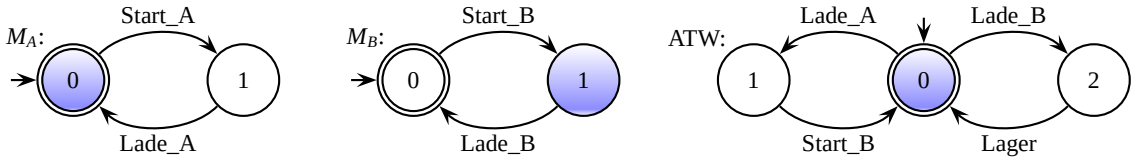


Abbildung 2.5: Nach Start_A, Lade_A, Start_B

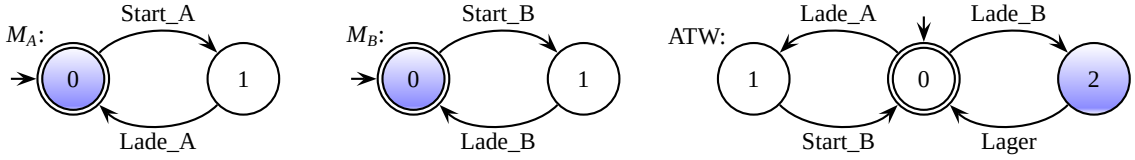


Abbildung 2.6: Nach Start_A, Lade_A, Start_B, Lade_B

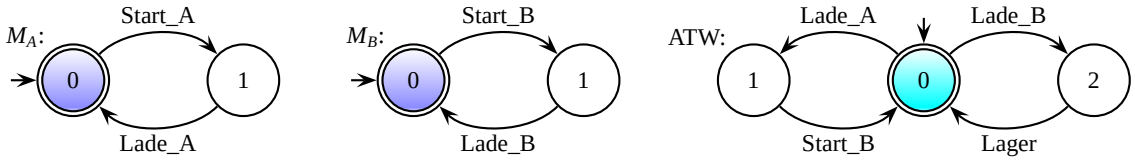


Abbildung 2.7: Nach Start_A, Lade_A, Start_B, Lade_B, Lager

Aus den Abbildungen 2.2 und 2.7 ist ersichtlich, dass das System nach den durchgespielten Ereignissen zu seinem Initialzustand zurückkehrt. Dies bedeutet, dass diese Ereignisreihenfolge beliebig oft wiederholt werden kann, ohne dass Verklemmungen auftreten. Ein hypothetisches Programm, das die Fertigungszelle in dieser Weise steuert, würde einen Test mit der genannten Folge von Ereignissen bestehen.

Allerdings ist diese Folge nicht die einzig mögliche. Falls mehrere Teile hintereinander hergestellt werden sollen, wird der Wunsch entstehen, die Maschine M_A sofort nach dem Ereignis Lade_A erneut zu starten, um sie so effizient wie möglich zu nutzen. Angenommen, der Vorgang auf Maschine M_B dauert auf Grund einer außergewöhnlichen Verzögerung so lange, dass Maschine M_A mit dem zweiten Teil fertig wird, bevor das erste Teil von M_B vollständig bearbeitet werden konnte. Dadurch wird die Ereignisfolge Start_A, Lade_A, Start_B, Start_A, Lade_A möglich. Die eingenommenen Zustände können auf den Abbildungen 2.8 bis 2.12 verfolgt werden.

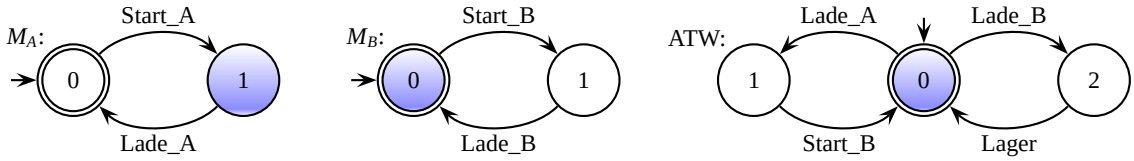


Abbildung 2.8: Nach Start_A

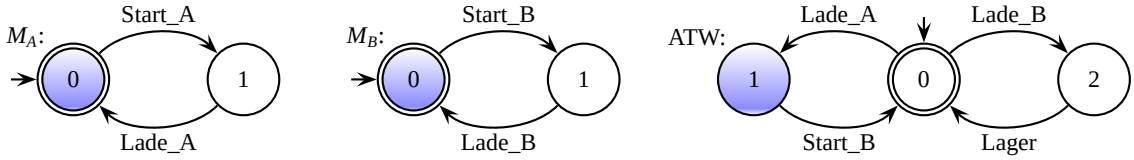


Abbildung 2.9: Nach Start_A, Lade_A

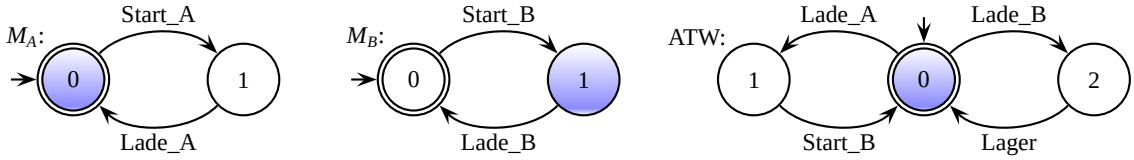


Abbildung 2.10: Nach Start_A, Lade_A, Start_B

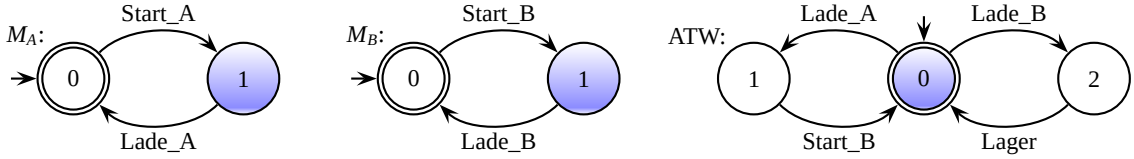


Abbildung 2.11: Nach Start_A, Lade_A, Start_B, Start_A

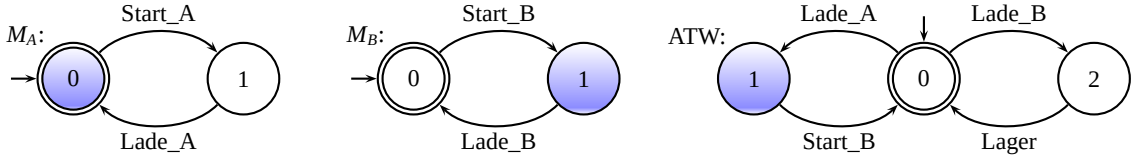


Abbildung 2.12: Nach Start_A, Lade_A, Start_B, Start_A, Lade_A

Nach dem Ablauf dieser Folge von Ereignissen ist ein Zustand erreicht, in dem das System nicht mehr weiterkommt: Der *ATW* hat ein Teil des Typs *A* geladen, das er an Maschine M_B weiter geben müsste. Diese ist aber noch mit der Bearbeitung des ersten Teils beschäftigt und kann das neue Teil vom Typ *A* nicht entgegennehmen. Andererseits kann der *ATW* das Teil vom Typ *B* der Maschine M_B nicht abnehmen wenn es schließlich fertig wird, weil er bereits das Teil vom Typ *A* geladen hat. Diese Verklemmung ist an den Automaten in Abbildung 2.12 erkennbar: Automat *ATW* ist im Zustand 1 und könnte im Prinzip das Ereignis Start_B ausführen. Dies wird jedoch durch den Automaten M_B verhindert, weil dieser sich in einem Zustand befindet, in dem das Ereignis Start_B nicht vorkommt. Andererseits könnte Automat M_B im Prinzip das Ereignis Lade_B ausführen, was aber durch den Automaten *ATW* verhindert wird, denn dort kann Lade_B in Zustand 1 nicht ausgeführt werden. Das einzig noch mögliche Ereignis ist ein weiteres Start_A, das von den Automaten M_B und *ATW* nicht verhindert wird, weil es in deren Alphabeten nicht vorkommt. Die Verklemmung wird aber dadurch nicht aufgehoben, und danach kann auch M_A keine Transitionen mehr ausführen.

In ähnlicher Weise würde auch das hypothetische Programm für die Steuerung des Systems an dieser Stelle blockieren. In so einem kleinen System würde der Fehler wahrscheinlich in der Testphase entdeckt werden. Das Beispiel ist jedoch stellvertretend für Systeme mit Zustandsräumen, die auf Grund ihrer Größe nicht erschöpfend durchlaufen werden können. Gerade in dieser Einschränkung liegt die Ursache vieler unentdeckter Fehler, die sich erst nach der Systemfreigabe bemerkbar machen. □

Das Ziel des Ramadge-Wonham-Modells ist es, Steuerungen für diskrete Systeme durch einen formalen Ansatz zu erzeugen, so dass alle erreichbaren Zustände bestimmte Eigenschaften, darunter die Verklemmungsfreiheit, erfüllen.

2.2 Das Prinzip

Das Ramadge-Wonham-Modell für die Steuerung diskreter Systeme wird seit Beginn der 1980er Jahre in der Arbeitsgruppe von Wonham an der Universität Toronto entwickelt. Die dort tätigen Wissenschaftler stammen überwiegend aus dem Gebiet der Elektrotechnik, und entsprechend ist auch das Modell für den Entwurf diskreter Systeme geprägt. Dies erklärt eine Anlehnung an die Steuer- und Regelungstechnik für Systeme mit kontinuierlichem Zustandsraum. Dort findet, wie in Abbildung 2.13 dargestellt, eine Rückkopplung der Ausgabe des zu steuernden Systems über einen Regler statt. Dadurch wird ein Verhalten erzielt, das einer gegebenen Spezifikation entspricht und das ohne die Rückkopplung nicht gewährleistet wäre.

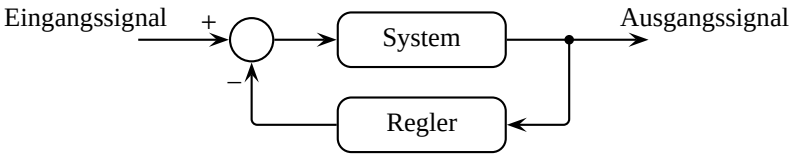


Abbildung 2.13: Steuerung eines kontinuierlichen Systems

Im Fall der diskreten Systeme tritt an die Stelle des Reglers ein *Überwacher* (Engl. *supervisor*). Der Überwacher kann die Zustandsänderungen des Systems über die dort auftretenden Ereignisse verfolgen und das System über eine Rückkopplungsschleife beeinflussen, wie in Abbildung 2.14 dargestellt. Seine Aufgabe ist es, anhand der wahrgenommenen Ereignisse die Steuerungsaktionen so zu wählen, dass das resultierende Systemverhalten eine gegebene Spezifikation erfüllt. Da das unüberwachte System alle physikalisch möglichen Aktionen ausführen kann, besteht die Steuerungsaktion aus einer Einschränkung der Ereignisse, so dass letztendlich eine Untermenge des möglichen Verhaltens übrig bleibt, die im Idealfall mit dem gewünschten Verhalten identisch ist.

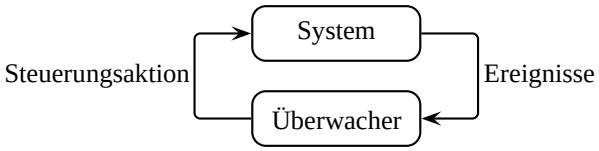


Abbildung 2.14: Das Ramadge-Wonham-Modell für diskrete Systeme

Der Überwacher ist nicht dafür verantwortlich, eine der erlaubten Aktionen auszuführen. Dies ist die Aufgabe einer aktiven Steuerung, die als Teil des Systems gesehen wird. Der Überwacher teilt dem System lediglich mit, welche Aktionen ihm im aktuellen Zustand zur Verfügung stehen. Die Unterschiede zwischen Überwacher und Steuerung wurden von Malik in [60] vertieft.

2.3 Die Modellierung

Im Ramadge-Wonham-Modell werden sowohl das System als auch die Spezifikation und der Überwacher als endliche Automaten dargestellt. Ein endlicher Automat ist ein 5-Tupel $\mathcal{A} = \langle \Sigma, Q, \delta, q^0, M \rangle$. Hier sind Σ das

Alphabet, dessen Elemente die Ereignisse darstellen, Q die Zustandsmenge, $\delta \subseteq Q \times \Sigma \times Q$ die Transitionsrelation, $q^0 \in Q$ der Initialzustand und $M \subseteq Q$ die Menge der akzeptierenden Zustände. Diese werden im Ramadge-Wonham-Modell *markierte* Zustände genannt, weil sie als Meilensteine dienen, die besondere Stellen im Ablauf der Ereignisse kennzeichnen. Sie werden in der Regel so gewählt, dass ein Zustand aus M immer dann erreicht wird, wenn das System eine Aufgabe vollendet hat. Die Mengen Σ und Q sind endlich, und damit sind es auch δ und M.

In der einschlägigen Literatur wird die Transitionsrelation meistens als deterministische Funktion $\delta : Q \times \Sigma \to Q$ definiert. An ihrer Stelle wird hier die o.g. Relation $\delta \subseteq Q \times \Sigma \times Q$ benutzt, weil dies die Schreibweise später vereinfacht. Die Notation $\delta(q, \sigma, q')$ steht für $(q, \sigma, q') \in \delta$. Um dieselbe Funktionalität zu gewährleisten, werden nur deterministische Relationen zugelassen, d.h., $\delta(q, \sigma, q') \wedge \delta(q, \sigma, q'') \Rightarrow q' = q''$. Die Relation δ wird in der üblichen Weise auf Wörter $s \in \Sigma^*$ erweitert, so dass $(q, s, q') \in \delta$ genau dann erfüllt ist, wenn es einen Pfad von q nach q' gibt, dessen Ereignisse das Wort s bilden.

Die Transitionsrelation δ muss nicht, wie in der Automatentheorie üblich, vollständig definiert sein. Sie wird bei der Modellierung eines Systems so gewählt, dass in jedem Zustand genau die Transitionen definiert sind, die das System ausführen kann. Folgende Definition erlaubt es, die jeweils definierten Ereignisse formal auszudrücken:

Definition 2.2 (Aktive Ereignisse) *Gegeben sei ein Automat $\mathcal{A} = \langle \Sigma, Q, \delta, q^0, M \rangle$. Dann ist für $q \in Q$*

$$\mathsf{act}_{\mathcal{A}}(q) := \{\sigma \in \Sigma \mid \exists q' \in Q.\delta(q, \sigma, q')\}$$

die Menge der aktiven Ereignisse von q.

Da die aktiven Ereignisse eines Zustands $q \in Q$ im Allgemeinen eine echte Teilmenge von Σ bilden, ist auch die Menge der Wörter, deren Ereignisse in der gegebenen Reihenfolge als Pfad auf dem Automaten abgebildet werden können, im Allgemeinen eine echte Teilmenge von Σ^*. Diese Teilmenge besteht genau aus den Wörtern, deren Ereignisse in der gegebenen Reihenfolge in dem System vorkommen können. Sie wird *generierte Sprache* des Automaten genannt:

Definition 2.3 (Generierte Sprache eines Automaten) *Die generierte Sprache eines Automaten $\mathcal{A} = \langle \Sigma, Q, \delta, q^0, M \rangle$ ist*

$$\mathsf{L}_Q(\mathcal{A}) := \left\{ s \in \Sigma^* \mid \exists q \in Q.\delta\left(q^0, s, q\right) \right\}.$$

Bei der üblichen Wahl der akzeptierenden Zustände akzeptiert der Automat genau die Wörter, deren Ereignisse, in der gegebenen Reihenfolge ausgeführt, eine vollendete Aufgabe seitens des Systems darstellen. Diese Menge wird wie üblich die *akzeptierte Sprache* des Automaten genannt:

Definition 2.4 (Akzeptierte Sprache eines Automaten) *Die akzeptierte Sprache eines Automaten $\mathcal{A} = \langle \Sigma, Q, \delta, q^0, M \rangle$ ist*

$$\mathsf{L}_M(\mathcal{A}) := \left\{ s \in \Sigma^* \mid \exists q \in M.\delta\left(q^0, s, q\right) \right\}.$$

Es gilt

$$\mathsf{L}_M(\mathcal{A}) \subseteq \mathsf{L}_Q(\mathcal{A}) \subseteq \Sigma^*.$$

Im Verlauf der Arbeit werden noch folgende Begriffe aus der Automatentheorie benötigt:

Definition 2.5 (Erreichbare, co-erreichbare und getrimmte Zustände eines Automaten) *Gegeben sei ein Automat $\mathcal{A} = \langle \Sigma, Q, \delta, q^0, M \rangle$. Es werden folgende Untermengen der Zustandsmenge Q definiert:*

- *erreichbare Zustände: $Q^{Ac} := \{q \in Q \mid \exists s \in \Sigma^*.\delta(q^0, s, q)\}$;*
- *co-erreichbare Zustände: $Q^{Co} := \{q \in Q \mid \exists s \in \Sigma^*.\exists q' \in M.\delta(q, s, q')\}$;*
- *getrimmte Zustände: $Q^{Tr} := Q^{Ac} \cap Q^{Co}$.*

Definition 2.6 (Einschränkung eines Automaten auf eine Zustandsmenge) *Gegeben seien ein Automat* $\mathcal{A} = \langle \Sigma, Q, \delta, q^0, M \rangle$ *und eine Menge* $Q_x \subseteq Q$*. Die Einschränkung von* $\mathcal{A}$ *auf* Q_x*, geschrieben* $\mathcal{A}|_{Q_x}$*, ist:*

1. *falls* $q^0 \in Q_x$*, der Automat* $\{\Sigma, Q_x, \delta|_x, q^0, M \cap Q_x\}$ *, wobei*
$$\delta|_x := \{(q, \sigma, q') \in \delta \mid q \in Q_x \wedge q' \in Q_x\},$$
2. *falls* $q^0 \notin Q_x$*, der leere Automat* $\langle \Sigma, \emptyset, \emptyset, \cdot, \emptyset \rangle$*.*

Definition 2.7 (Erreichbare, co-erreichbare und getrimmte Komponenten eines Automaten) *Die erreichbare Komponente eines Automaten* $\mathcal{A} = \langle \Sigma, Q, \delta, q^0, M \rangle$ *ist* $\mathsf{Ac}(\mathcal{A}) := \mathcal{A}|_{Q^{Ac}}$*. Gleiches gilt für die co-erreichbare Komponente* $\mathsf{Co}(\mathcal{A}) := \mathcal{A}|_{Q^{Co}}$ *und die getrimmte Komponente* $\mathsf{Tr}(\mathcal{A}) := \mathcal{A}|_{Q^{Tr}}$*. Ein Automat ist erreichbar, wenn* $\mathsf{Ac}(\mathcal{A}) = \mathcal{A}$*, co-erreichbar, wenn* $\mathsf{Co}(\mathcal{A}) = \mathcal{A}$ *und getrimmt, wenn* $\mathsf{Tr}(\mathcal{A}) = \mathcal{A}$*.*

Im Folgenden werden die Automaten für das System, die Spezifikation und den Überwacher näher beschrieben.

2.3.1 Das System

Der Automat für die Systembeschreibung wird $\mathcal{A}_\mathcal{P}$ genannt (der Index stammt aus dem Englischen *plant*). Da die Transitionsrelationen in der Praxis oft sehr große Zustandsräume haben, kann $\mathcal{A}_\mathcal{P}$ meistens nicht in einem Guss erstellt werden. Stattdessen wird das System anhand seiner Komponenten erfasst, so dass zunächst für jede Komponente ein überschaubarer Automat erstellt wird. Die einzelnen Automaten werden dann mit Hilfe der *parallelen Komposition* zusammengefügt. Deren Ergebnis ist ein Automat, dessen Transitionsrelation die Ereignisse der einzelnen Komponenten in allen möglichen Reihenfolgen enthält (Engl. *interleaving* der Ereignisse):

Definition 2.8 (Parallele Komposition zweier Automaten) *Die parallele Komposition der Automaten* $\mathcal{A}_\mathcal{M} = \langle \Sigma, Q_\mathcal{M}, \delta_\mathcal{M}, q^0_\mathcal{M}, M_\mathcal{M} \rangle$ *und* $\mathcal{A}_\mathcal{N} = \langle \Sigma, Q_\mathcal{N}, \delta_\mathcal{N}, q^0_\mathcal{N}, M_\mathcal{N} \rangle$ *ist der Automat*

$$\mathcal{A}_\mathcal{M} \| \mathcal{A}_\mathcal{N} := \langle \Sigma_\mathcal{M} \cup \Sigma_\mathcal{N}, Q_\mathcal{M} \times Q_\mathcal{N}, \delta_{\mathcal{M}\|\mathcal{N}}, (q^0_\mathcal{M}, q^0_\mathcal{N}), M_\mathcal{M} \times M_\mathcal{N} \rangle,$$

wobei

$$\begin{aligned} \delta_{\mathcal{M}\|\mathcal{N}} \quad := \quad & \{((p,q), \sigma, (p', q')) \mid \delta_\mathcal{M}(p, \sigma, p') \wedge \delta_\mathcal{N}(q, \sigma, q')\} \\ \cup \quad & \{((p,q), \sigma, (p', q)) \mid \sigma \in (\Sigma_\mathcal{M} \setminus \Sigma_\mathcal{N}) \wedge \delta_\mathcal{M}(p, \sigma, p')\} \\ \cup \quad & \{((p,q), \sigma, (p, q')) \mid \sigma \in (\Sigma_\mathcal{N} \setminus \Sigma_\mathcal{M}) \wedge \delta_\mathcal{N}(q, \sigma, q')\}. \end{aligned}$$

Aus Definition 2.8 folgt, dass das Ergebnis der parallelen Komposition genau dann eine Transition mit dem Ereignis σ im Zustand $(p, q) \in Q_\mathcal{M} \times Q_\mathcal{N}$ aufweist, wenn:

- entweder beide Komponenten in den Zuständen p bzw. q in der Lage sind, eine Transition mit dem Ereignis σ auszuführen; in diesem Fall geschieht der Zustandswechsel auf beiden Komponenten gleichzeitig;
- oder eine Transition für σ nur in einem der Zustände p oder q der Komponenten definiert ist, und dieses Ereignis dem Alphabet der anderen Komponente fremd ist; in diesem Fall macht die Komponente, in der die Transition definiert ist, einen Zustandswechsel, während der Zustand der anderen unverändert bleibt.

Transitionen mit Ereignissen, die in den Alphabeten beider Komponenten vertreten sind, aber nur in einem der Zustände p oder q vorkommen, werden durch die parallele Komposition blockiert. Außerdem ist ein Zustand $(p, q) \in Q_\mathcal{M} \times Q_\mathcal{N}$ nur dann ein markierter Zustand, wenn sowohl p als auch q in den Komponenten markiert sind. Dadurch wird eine Aufgabe nur dann als beendet gewertet, wenn alle Teilsysteme eine Aufgabe vollendet haben. Beispiel 2.1 zeigt, dass gerade diese Eigenschaften für die Modellierung benötigt werden. Somit erzeugt die parallele Komposition der Teilsysteme genau die von dem Ramadge-Wonham-Modell vorausgesetzten Systembeschreibung. Die parallele Komposition ist assoziativ und kommutativ.

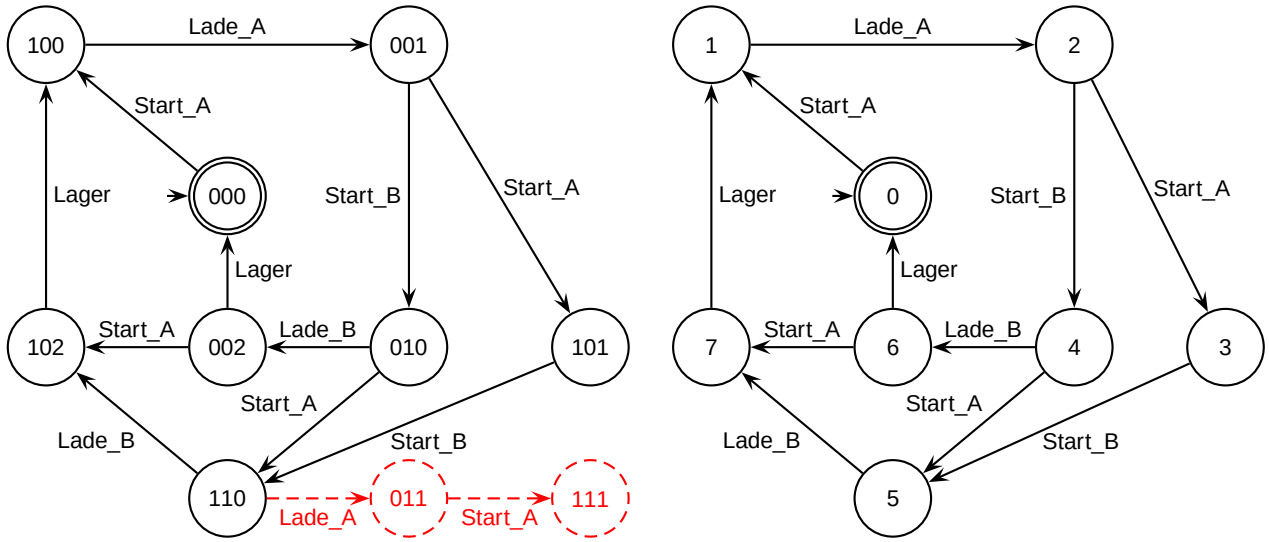


Abbildung 2.15: Die Automaten $\mathcal{A}_\mathcal{P}$ und $\mathcal{A}_\mathcal{E}$ für die Fertigungszelle

Beispiel 2.9 Im Falle des Beispiels 2.1 gilt $\mathcal{A}_\mathcal{P} = M_A||M_B||ATW$ (s. Abbildung 2.2). Der resultierende Automat ist auf der linken Seite der Abbildung 2.15 zu sehen. Die Nummerierung der Zustände gibt die Zustände der Komponenten M_A, M_B bzw. ATW wieder. Die Ereignisfolgen aus Beispiel 2.1 können auf dem Automaten $\mathcal{A}_\mathcal{P}$ verfolgt werden. Die erste davon, nämlich Start_A, Lade_A, Start_B, Lade_B, Lager, führt wie zu erwarten vom Initialzustand 000 wieder zu diesem zurück. Die zweite, Start_A, Lade_A, Start_B, Start_A, Lade_A, führt in die gestrichelt dargestellte Sackgasse. □

2.3.2 Die Spezifikation

Die Spezifikation für das gewünschte Systemverhalten wird durch das direkte Produkt des Automaten $\mathcal{A}_\mathcal{P}$ mit einem Hilfsautomaten $\mathcal{A}_\mathcal{E}$ gegeben[1].

Definition 2.10 (Produkt zweier Automaten) *Das Produkt der Automaten* $\mathcal{A}_\mathcal{M} = \langle \Sigma, Q_\mathcal{M}, \delta_\mathcal{M}, q^0_\mathcal{M}, M_\mathcal{M} \rangle$ *und* $\mathcal{A}_\mathcal{N} = \langle \Sigma, Q_\mathcal{N}, \delta_\mathcal{N}, q^0_\mathcal{N}, M_\mathcal{N} \rangle$ *ist der Automat*

$$\mathcal{A}_\mathcal{M} \times \mathcal{A}_\mathcal{N} := \langle \Sigma, Q_\mathcal{M} \times Q_\mathcal{N}, \delta_{\mathcal{M}\times\mathcal{N}}, (q^0_\mathcal{M}, q^0_\mathcal{N}), M_\mathcal{M} \times M_\mathcal{N} \rangle, \text{ wobei}$$

$$\delta_{\mathcal{M}\times\mathcal{N}}((p,q),\sigma,(p',q')) \Leftrightarrow \delta_\mathcal{M}(p,\sigma,p') \wedge \delta_\mathcal{N}(q,\sigma,q').$$

Im Falle des vorangegangenen Beispiels kann $\mathcal{A}_\mathcal{E}$ so gewählt werden, dass die unerwünschte Sackgasse in dem Produkt $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ nicht mehr vorhanden ist. Dazu genügt es, eine Kopie von $\mathcal{A}_\mathcal{P}$ zu erzeugen und davon die unerwünschten Zustände zu entfernen, wodurch der Automat $\mathcal{A}_\mathcal{E}$ auf der rechten Seite der Abbildung 2.15 entsteht. Die Spezifikation $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ wird in Abbildung 2.16 wiedergegeben. Die Nummerierung der Zustände ergibt sich aus den Zuständen der Automaten $\mathcal{A}_\mathcal{P}$ und $\mathcal{A}_\mathcal{E}$. Sie gibt die Zustände der Automaten M_A, M_B, ATW bzw. $\mathcal{A}_\mathcal{E}$ wieder.

Im vorangegangenen Beispiel wurde der Automat $\mathcal{A}_\mathcal{E}$ durch das Streichen von Zuständen aus einer Kopie von $\mathcal{A}_\mathcal{P}$ erstellt. Bei größeren Automaten ist eine solche Angabe der zu entfernenden Zustände nicht ohne weiteres möglich, so dass andere Lösungen für die Erzeugung der Spezifikation angewendet werden müssen. So kann z.B. der Automat $\mathcal{A}_\mathcal{E}$ aus mehreren kleinen, überschaubaren Automaten zusammengefügt werden, wie es in dem Beispiel in Abschnitt 5.4 der Fall ist. In der Regel hat dann die Spezifikation einen größeren Zustandsraum als das System, weil $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ die Transitionsrelation von $\mathcal{A}_\mathcal{P}$ verfeinert. In anderen Fällen wird der Hilfsautomat $\mathcal{A}_\mathcal{E}$ nicht zur Verfeinerung der Transitionsrelation, sondern lediglich zur Veränderung der Markierung der Zustände benutzt. Diese Technik wird bei der Lösung des Beispiels in Abschnitt 5.5 angewendet.

[1] In der Literatur wird $\mathcal{A}_\mathcal{E}$ oft auch als *specification automaton* bezeichnet, was leicht zu der falschen Annahme führt, $\mathcal{A}_\mathcal{E}$ sei die Spezifikation selbst. Diese kommt jedoch erst durch das Produkt mit $\mathcal{A}_\mathcal{P}$ zustande.

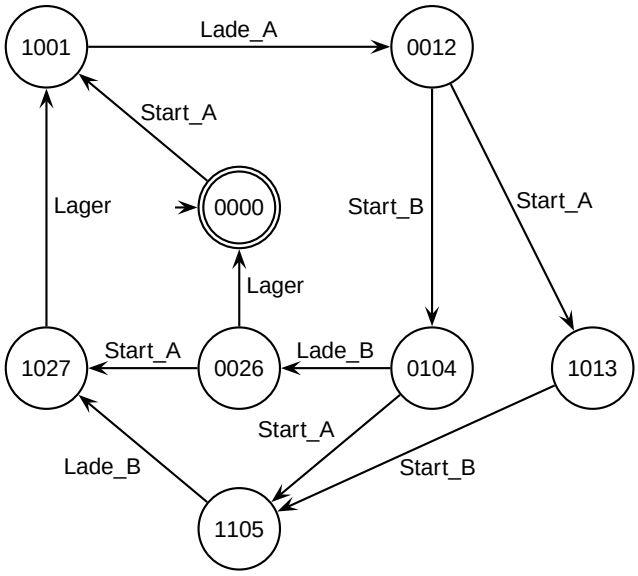


Abbildung 2.16: Die Spezifikation $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ für die Fertigungszelle

2.3.3 Der Überwacher

Der Überwacher in Abbildung 2.14 besteht aus einem Automaten $\mathcal{A}_\mathcal{S}$, der die Ereignisse des Systems wahrnimmt, um gleichzeitig mit diesem den Zustand zu wechseln und so den aktuellen Zustand des Systems zu verfolgen. Die Steuerungsaktion kommt dadurch zustande, dass dem System unmittelbar nach jedem Zustandswechsel die Menge der aktiven Ereignisse des neu betretenen Zustands des Überwachers mitgeteilt wird. Von dem System wird erwartet, dass das nächste Ereignis aus dieser Menge gewählt wird, alle anderen Ereignisse gelten als verboten.

Der Automat $\mathcal{A}_\mathcal{S}$ muss gewährleisten, dass der Zustandsraum der Spezifikation $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ während des Betriebs nicht verlassen wird. Im einfachsten Fall wird zu diesem Zweck $\mathcal{A}_\mathcal{S} := \mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ gewählt.

Beispiel 2.11 Im Falle des Beispiels 2.9 würde das überwachte System bei der Wahl $\mathcal{A}_\mathcal{S} := \mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ mit dem Automat $\mathcal{A}_\mathcal{P}$ aus Abbildung 2.15 und der Spezifikation $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ aus Abbildung 2.16 arbeiten. Immer dann, wenn sich das System in den Zustand 110 begibt, befindet sich der Überwacher im Zustand 1105 und meldet, dass lediglich das Ereignis `Lade_B` erlaubt ist. Dadurch wird der unerwünschte Eingang in die Sackgasse verboten. □

Die Spezifikation wird allerdings nur solange eingehalten, wie das System sich an die vorgegebenen Einschränkungen halten kann. Im nächsten Abschnitt wird deutlich, dass dies nicht immer der Fall ist. Die Bestimmung des Überwachers wird dann komplexer als in dem vorangegangenen Beispiel.

2.4 Formalisierung der Überwachersynthese

Die Überwachersynthese besteht darin, für ein gegebenes System $\mathcal{A}_\mathcal{P}$ und eine gegebene Spezifikation $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ einen passenden Überwacher $\mathcal{A}_\mathcal{S}$ zu berechnen, so dass der Zustandsraum von $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ nicht verlassen und gleichzeitig ein verklemmungsfreier Betrieb gewährleistet wird. Im Folgenden wird deshalb untersucht, wie sich diese Anforderungen auf die verwendeten Automaten auswirken.

2.4.1 Steuerbarkeit

Aus der Beschreibung des Überwachers in Abschnitt 2.3.3 geht hervor, dass die Konstruktion in Abbildung 2.14 nur dann funktionieren kann, wenn das System in der Lage ist, die verbotenen Ereignisse tatsächlich zu unterbinden. Dies wurde in Beispiel 2.11 vorausgesetzt, ist jedoch nicht immer der Fall: Lassen sich z.B. Ereignisse

wie das Einschalten von Maschinen oder das Senden einer Meldung in einem Kommunikationssystem in der Regel bei Bedarf verhindern, ist dies z.B. bei Sensorsignalen, dem Ablaufen eines Zeitglieds oder der Ankunft einer Meldung nicht möglich. Das Ramadge-Wonham-Modell unterscheidet deshalb zwischen *steuerbaren* Ereignissen $\Sigma_c \subseteq \Sigma$ und *nicht steuerbaren* Ereignissen $\Sigma_u := \Sigma \setminus \Sigma_c$ (die Indizes kommen aus dem Englischen *controllable* bzw. *uncontrollable*). Transitionen eines Automaten, die mit steuerbaren bzw. nicht steuerbaren Ereignissen beschriftet sind, werden ebenfalls als *steuerbar* bzw. *nicht steuerbar* bezeichnet.

Beispiel 2.12 Im Fall der Fertigungszelle könnte der Ladevorgang Lade_A automatisch angestoßen werden, sobald M_A mit der Bearbeitung eines Teils fertig ist. Dann würde der Befehl, dieses Ereignis im Zustand 110 des Systems zu verhindern, keinen Sinn ergeben, und der Automat $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ in Abbildung 2.16 wäre nicht wie in Beispiel 2.11 als Überwacher geeignet. □

Beispiel 2.12 zeigt, dass die Berechnung des Überwachers die Steuerbarkeit der Ereignisse berücksichtigen muss. Die nicht steuerbaren Ereignisse stellen eine Bedingung an die Existenz des Überwachers: Befindet sich der Überwacher im Zustand $(p, q) \in Q_\mathcal{S}$, so sind aus dessen Sicht genau die aktiven Ereignisse $\mathsf{act}_{\mathcal{A}_\mathcal{S}}((p, q))$ erlaubt. Das System befindet sich zu diesem Zeitpunkt in dem Zustand p, und die nicht steuerbaren Ereignisse, die in diesem Zustand physikalisch möglich sind, bilden die Menge $\mathsf{act}_{\mathcal{A}_\mathcal{P}}(p) \cap \Sigma_u$. Wie oben verdeutlicht, darf der Überwacher keine physikalisch möglichen, nicht steuerbaren Ereignisse verbieten, d.h., für jeden Zustand $(p, q) \in Q_\mathcal{S}$ muss gelten:

$$\mathsf{act}_{\mathcal{A}_\mathcal{P}}(p) \cap \Sigma_u \subseteq \mathsf{act}_{\mathcal{A}_\mathcal{S}}((p, q)) .$$

Die in Beispiel 2.11 verwendete Lösung $\mathcal{A}_\mathcal{S} := \mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ ist also nur dann zulässig, wenn jeder Zustand $(p, q) \in Q_\mathcal{P} \times Q_\mathcal{E}$ die Bedingung

$$\mathsf{act}_{\mathcal{A}_\mathcal{P}}(p) \cap \Sigma_u \subseteq \mathsf{act}_{\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}}((p, q)) \tag{2.1}$$

erfüllt[2]. Zustände der Spezifikation $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$, für die diese Bedingung nicht zutrifft, dürfen nicht betreten werden. Sie werden deshalb *verbotene Zustände* genannt (s.a. Definition 2.17).

Ramadge und Wonham haben in [72] die Steuerbarkeit einer Spezifikation mit Hilfe der Sprachen aus den Definitionen 2.3 und 2.4 beschrieben und zu diesem Zweck folgende Definition eingeführt:

Definition 2.13 (Steuerbare Sprache) *Sei $\mathcal{A}_\mathcal{P} = \langle \Sigma, Q_\mathcal{P}, \delta_\mathcal{P}, q^0_\mathcal{P}, M_\mathcal{P} \rangle$ ein Automat mit steuerbaren Ereignissen $\Sigma_c \subseteq \Sigma$ und nicht steuerbaren Ereignissen $\Sigma_u := \Sigma \setminus \Sigma_c$, der die Sprache $\mathsf{L}_Q(\mathcal{A}_\mathcal{P})$ generiert. Sei ferner $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ ein Automat, der die Sprache $K := \mathsf{L}_M(\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E})$ akzeptiert. Sei $\overline{K}$ die Menge aller Präfixe der Wörter in K. Die Sprache K ist genau dann mit Bezug auf $\mathcal{A}_\mathcal{P}$ steuerbar, wenn*

$$\overline{K}\Sigma_u \cap \mathsf{L}_Q(\mathcal{A}_\mathcal{P}) \subseteq \overline{K}.$$

Definition 2.13 ist folgendermaßen zu lesen: Die Sprache K ist genau dann steuerbar, wenn für jedes Präfix s eines Wortes in K, das in dem System $\mathcal{A}_\mathcal{P}$ von einem nicht steuerbaren Ereignis $\sigma \in \Sigma_u$ gefolgt werden kann, das Wort $s' := s\sigma$ wiederum in K enthalten ist.

Spezifikationen, deren akzeptierte Sprache steuerbar ist, werden *steuerbare Spezifikationen* genannt. Sie versuchen nicht, physikalisch mögliche, nicht steuerbare Ereignisse zu verbieten. Die Steuerbarkeit von $K = \mathsf{L}_M(\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E})$ kann anhand der Automaten $\mathcal{A}_\mathcal{P}$ und $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ erkannt werden. Allerdings müssen dazu die nicht co-erreichbaren Zustände des letzteren entfernt werden, wie folgendes Beispiel zeigt.

Beispiel 2.14 Gegeben seien die Automaten $\mathcal{A}_\mathcal{P}$ und $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ in Abbildung 2.17. Es sei $\Sigma_u = \{\beta\}$. Der Automat $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ akzeptiert die Sprache $K = (\alpha\beta\gamma)^*$. K ist nicht steuerbar, weil in Zustand 2 von $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ auf das Präfix $\alpha\beta \in \overline{K}$ das Ereignis β folgen kann, das Wort $\alpha\beta\beta$ jedoch nicht in $\overline{K}$ enthalten ist. Trotzdem hat $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ keine verbotenen Zustände. Wird jedoch der nicht co-erreichbaren Zustand 3 (zusammen mit den Transitionen $(2, \beta, 3)$ und $(3, \beta, 3)$) von $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ entfernt, so verletzt Zustand 2 die Bedingung 2.1 und wird zum verbotenen Zustand. □

Beispiel 2.14 ist allgemein gültig und lässt folgende Aussage zu:

[2] Es genügt, dass alle erreichbaren Zustände von $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ diese Bedingung erfüllen. Für dieser Arbeit ist diese Tatsache jedoch nicht von Bedeutung.

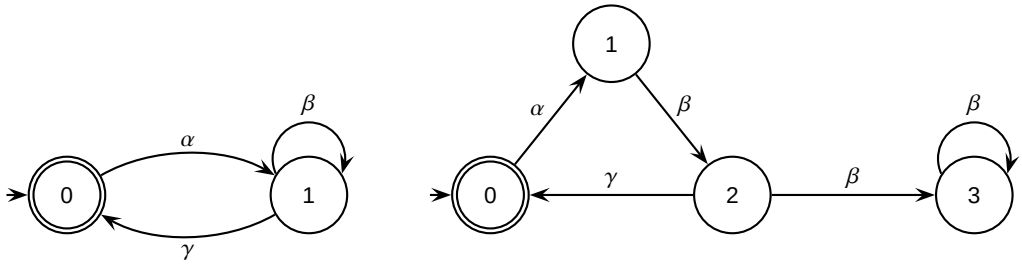


Abbildung 2.17: Die Automaten $\mathcal{A}_\mathcal{P}$ und $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ für Beispiel 2.14

Lemma 2.15 (Steuerbarkeit und verbotene Zustände) *Eine co-erreichbare Spezifikation ist genau dann steuerbar, wenn sie keine verbotenen Zustände enthält.*

2.4.2 Verklemmungsfreiheit

Außer der Steuerbarkeit ist die Verklemmungsfreiheit zu berücksichtigen. Diese verlangt, dass das System stets in der Lage sein muss, eine Aufgabe zu vollenden. Dies ist nicht der Fall, wenn das System entweder einen Zustand erreicht, in dem keine weiteren Ereignisse möglich sind und noch keine Aufgabe vollendet ist (*deadlock*), oder sich nur noch innerhalb einer Zustandsmenge bewegen kann, in der keine Aufgabe vollendet wird (*livelock*). Da die markierten Zustände im Ramadge-Wonham-Modell vollendete Aufgaben kennzeichnen, kommt die Verklemmungsfreiheit der Co-Erreichbarkeit des Überwachers gleich (s. Definition 2.7).

Es ist deshalb zweckmäßig, zu Beginn der Überwachersynthese die nicht co-erreichbaren Zustände aus $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ zu entfernen. Damit kann auch die Steuerbarkeit, wie in Abschnitt 2.4.1 gezeigt, anhand der verbotenen Zustände ermittelt werden. Diese Erkenntnis wird von den Algorithmen in Abschnitt 2.5 genutzt.

2.4.3 Das Überwachersyntheseproblem

Auf den Konzepten von Steuerbarkeit und Verklemmungsfreiheit aufbauend konnten Ramadge und Wonham auch für den Fall, dass die Spezifikation $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ verbotene Zustände enthält, eine Berechnungsmethode für einen Überwacher finden. Dieser kann dann zwar nicht die gewünschte Spezifikation einhalten, wohl aber einen Teil davon. Es gelang ihnen in [72] zu zeigen, dass jede nicht steuerbare Sprache $K = \mathsf{L}_M(\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E})$ stets steuerbare Teilmengen enthält, und dass diese einen Verband bilden. Folglich enthält jede nicht steuerbare Sprache eine *größte steuerbare Teilsprache* (Engl. *supremal controllable sublanguage*) $\mathsf{supC}(K)$.

Sollte die Sprache $K = \mathsf{L}_M(\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E})$ nicht steuerbar sein, muss es also einen Automaten $\mathcal{A}_\mathcal{S}$ geben, der das Verhalten des Systems auf $\mathsf{supC}(K)$ einschränkt. Allerdings können die Einschränkungen auch zu weit gehen – $\mathsf{supC}(K)$ kann im Extremfall die leere Menge sein –, so dass das Ergebnis in der Praxis nicht immer brauchbar ist. Das Überwachersyntheseproblem sieht deshalb die Möglichkeit vor, ein *kleinstes akzeptables Verhalten* anzugeben.

Definition 2.16 (Überwachersyntheseproblem) *Gegeben seien ein System $\mathcal{A}_\mathcal{P}$ und ein Automat $\mathcal{A}_\mathcal{E}$ zur Erstellung einer Spezifikation, so dass $K := \mathsf{L}_M(\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}) \subseteq \mathsf{L}_M(\mathcal{A}_\mathcal{P})$ das gewünschte Verhalten des überwachten Systems ausdrückt. Gegeben sei ferner die Sprache $A_{min} \subseteq K$, die das kleinste akzeptable Verhalten des überwachten Systems darstellt. Gesucht ist ein verklemmungsfreier Überwacher $\mathcal{A}_\mathcal{S}$, so dass $A_{min} \subseteq \mathsf{L}_M(\mathcal{A}_\mathcal{S}) \subseteq K$.*

Ramadge und Wonham haben in [72] gezeigt, dass das Überwachersyntheseproblem genau dann lösbar ist, wenn $\mathsf{supC}(K) \supseteq A_{min}$. Gleichzeitig ist $\mathsf{supC}(K)$ die Lösung, die das Systemverhalten am wenigsten einschränkt. Die Überwachersynthese besteht also darin, aus $\mathcal{A}_\mathcal{P}$ und $\mathcal{A}_\mathcal{E}$ einen verklemmungsfreien Automaten $\mathcal{A}_\mathcal{S}$ zu berechnen, so dass $\mathsf{L}_M(\mathcal{A}_\mathcal{S}) = \mathsf{supC}(K)$. Solange die Einschränkungen, die sich daraus ergeben, tolerierbar sind, kann dieser Automat als Überwacher dienen.

2.5 Synthesealgorithmen

Ein verklemmungsfreier Automat $\mathcal{A}_\mathcal{S}$ mit der Eigenschaft $\mathsf{L}_M(\mathcal{A}_\mathcal{S}) = \mathsf{supC}(K)$ kann aus den Automaten $\mathcal{A}_\mathcal{P}$ und $\mathcal{A}_\mathcal{E}$ abgeleitet werden. Wonham und Ramadge haben zu diesem Zweck in [86] einen Algorithmus veröffentlicht. Eine effizientere Version wurde von Kumar und Garg nachgeliefert [51]. Diese Algorithmen werden in den nächsten Abschnitten beschrieben. Aus der Lösung von Kumar und Garg wird hier ein zusätzlicher Algorithmus abgeleitet, der in Kapitel 4 weiterverwendet wird. Des Weiteren gibt es Ansätze von Jiang und Kumar [43], de Alfaro et al. [24] und Arnold et al. [2], die das Überwachersyntheseproblem mit Hilfe von Baumautomaten lösen. Diese werden erst in Kapitel 6 angesprochen, da sie Grundlagen benötigen, die erst später eingebracht werden.

Die folgenden Algorithmen gehen alle von den Automaten $\mathcal{A}_\mathcal{P}$ und $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ aus und entfernen aus letzterem verbotene und nicht co-erreichbare Zustände. Zustände aus $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ zu entfernen bedeutet, diesen Automaten gemäß Definition 2.6 auf eine bestimmte Menge $Q \subseteq Q_\mathcal{P} \times Q_\mathcal{E}$ einzuschränken. Mit dem Entfernen eines Zustands fallen auch alle Transitionen weg, die von ihm ausgehen oder zu ihm führen. Sei $(p, q) \in Q_\mathcal{P} \times Q_\mathcal{E}$ ein Zustand, der in einem gegebenen Syntheseschritt entfernt wird, und $(p', q') \in Q_\mathcal{P} \times Q_\mathcal{E}$ ein Zustand, der in diesem Schritt erhalten bleibt. Wenn es eine nicht steuerbare Transition $(p', q'), \sigma, (p, q)$ in $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ gibt, dann wird diese zusammen mit (p, q) entfernt, und so wird auch (p', q') zum verbotenen Zustand. Dies wird von der folgenden Definition erfasst:

Definition 2.17 (Verbotene Zustände) *Gegeben seien eine Systembeschreibung $\mathcal{A}_\mathcal{P}$ und eine Spezifikation $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$, die durch das Entfernen von Zuständen auf die Menge $Q \subseteq Q_\mathcal{P} \times Q_\mathcal{E}$ eingeschränkt ist. Sei Σ_u die Menge der nicht steuerbaren Ereignisse der gegebenen Automaten. Zustände aus $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$, welche die Bedingung*

$$\mathsf{act}_{\mathcal{A}_\mathcal{P}}(p) \cap \Sigma_u \subseteq \mathsf{act}_{(\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E})|_Q}(p, q)$$

verletzen, werden verbotene Zustände genannt.

Bedingung 2.1 ist ein Sonderfall von Definition 2.17. Dort ist $Q = Q_\mathcal{P} \times Q_\mathcal{E}$.

2.5.1 Der Ansatz von Wonham und Ramadge

Algorithmus 2.18 *Gegeben seien die Automaten $\mathcal{A}_\mathcal{P}$ und $\mathcal{A}_\mathcal{E}$ für das System bzw. für die Erstellung der Spezifikation und eine Partitionierung der Ereignismenge Σ in $\Sigma_c \subseteq \Sigma$ und $\Sigma_u := \Sigma \setminus \Sigma_c$.*

1. *Initialisiere $\mathcal{A}_\mathcal{S} := \mathsf{Tr}(\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E})$. Es gilt $\mathcal{A}_\mathcal{S} = \langle \Sigma, Q_\mathcal{S}, \delta_\mathcal{S}, q^0_\mathcal{S}, M_\mathcal{S} \rangle$.*
2. *Berechne die verbotenen Zustände $Q_b := \left\{(p, q) \in Q_\mathcal{S} \mid \mathsf{act}_{\mathcal{A}_\mathcal{P}}(p) \cap \Sigma_u \not\subseteq \mathsf{act}_{\mathcal{A}_\mathcal{S}}((p, q))\right\}$.*
3. *Falls $Q_b = \emptyset$, gebe $\mathcal{A}_\mathcal{S}$ zurück; sonst:*
   - *3.1 Berechne das Komplement $\bar{Q}_b$.*
   - *3.2 Ersetze $\mathcal{A}_\mathcal{S}$ durch $\mathsf{Tr}\left(\mathcal{A}_\mathcal{S}|_{\bar{Q}_b}\right)$.*
   - *3.3 Gehe zu Schritt 2.*

Der Algorithmus besteht im Wesentlichen aus einer Schleife, in der abwechselnd verbotene und nicht getrimmte (und damit nicht co-erreichbare) Zustände von dem Automaten $\mathcal{A}_\mathcal{S}$ entfernt werden (s. Definitionen 2.6 und 2.7). Diese Abwechslung entsteht, weil einerseits das Entfernen von nicht co-erreichbaren Zuständen neue verbotene Zustände aufdecken, andererseits das Entfernen von verbotenen Zuständen die Co-Erreichbarkeit verletzen kann. Gibt es keine verbotenen Zustände mehr, ist die Synthese beendet.

Die Komplexität von Algorithmus 2.18 lässt sich wie folgt bestimmen: Sowohl das Produkt der Automaten als auch das Trimmen in Schritt 1 sind durch die obere Zeitschranke $O\left(\left|Q_{\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}}\right| |\Sigma|\right)$ begrenzt. Für die Ermittlung der verbotenen Zustände in Schritt 2.1 müssen alle Transitionen angeschaut werden. Dies geschieht innerhalb der gleichen oberen Zeitschranke. Die Berechnung von $\bar{Q}_b$ in Schritt 3.1 ist durch $O\left(\left|Q_{\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}}\right|\right)$ begrenzt, die

Einschränkung auf $\bar{Q}_b$ und das Trimmen in Schritt 3.2 wiederum durch $O\left(\left|Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}\right||\Sigma|\right)$. Es können maximal $\left|Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}\right|$ Wiederholungen stattfinden, weil im schlimmsten Fall genau ein Zustand pro Durchlauf entfernt wird. Insgesamt ergibt dies eine obere Zeitgrenze von $O\left(\left|Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}\right|^2|\Sigma|\right)$.

Beispiel 2.19 Zur Veranschaulichung wird ein passender Überwacher für die Fertigungszelle aus den vorangegangenen Beispielen berechnet. Die Automaten für das System und die Spezifikation sind $\mathcal{A}_\mathcal{P}$ aus Abbildung 2.15 bzw. $\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}$ aus Abbildung 2.16. Die Ereignisse werden wie folgt in steuerbar bzw. nicht steuerbar eingeteilt: $\Sigma_c = \{\mathsf{Start\_A}\}$, $\Sigma_u = \{\mathsf{Start\_B}, \mathsf{Lade\_A}, \mathsf{Lade\_B}, \mathsf{Lager}\}$. Algorithmus 2.18 durchläuft dann folgende Schritte:

1. $\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}$ ist bereits getrimmt, also beginnt die erste Iteration mit den Automaten in Abbildung 2.18.

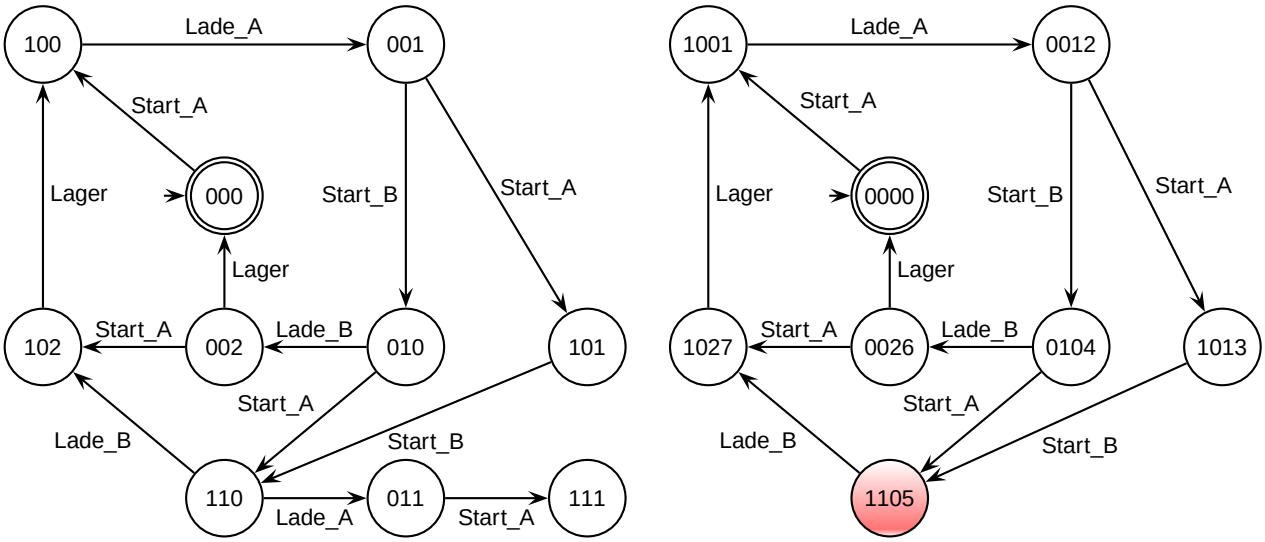


Abbildung 2.18: Die Automaten $\mathcal{A}_\mathcal{P}$ und $\mathcal{A}_\mathcal{S}$ zu Beginn der 1. Iteration

2. Die Ermittlung der verbotenen Zustände nach Bedingung 2.1 ergibt $Q_b = \{1105\}$. Der verbotene Zustand ist in Abbildung 2.18 eingefärbt dargestellt.

3. Es gilt $Q_b \neq \emptyset$, also folgen:

   3.1 $\bar{Q}_b = \{0000, 1001, 0012, 1013, 0104, 0026, 1027\}$

   3.2 Bei der Berechnung von $\mathcal{A}_\mathcal{S}|_{\bar{Q}_b}$ fällt der Zustand 1105 weg; dadurch ist der Zustand 1013 nicht mehr co-erreichbar, also wird er bei der Berechnung von $\mathsf{Tr}\left(\mathcal{A}_\mathcal{S}|_{\bar{Q}_b}\right)$ entfernt. Die Automaten für die 2. Iteration sind in Abbildung 2.19 dargestellt.

   3.3 Es wird zu Schritt 2 zurückgesprungen.

2. Die Ermittlung der verbotenen Zustände ergibt dieses Mal $Q_b = \emptyset$.

3. Die Berechnung ist beendet. Das Ergebnis ist der Automat $\mathcal{A}_\mathcal{S}$ in Abbildung 2.19.

Wenn die Automaten $\mathcal{A}_\mathcal{P}$ und $\mathcal{A}_\mathcal{S}$ wie in Abbildung 2.14 verknüpft werden, wird das System nun so gesteuert, dass bereits in den Zuständen 0012 und 0104 das (steuerbare) Ereignis Start_A verhindert wird. Dadurch kann das System nie in den verbotenen Zustand 1105 kommen, in dem sich das (nicht steuerbare) Ereignis Lade_A nicht mehr verhindern ließe. □

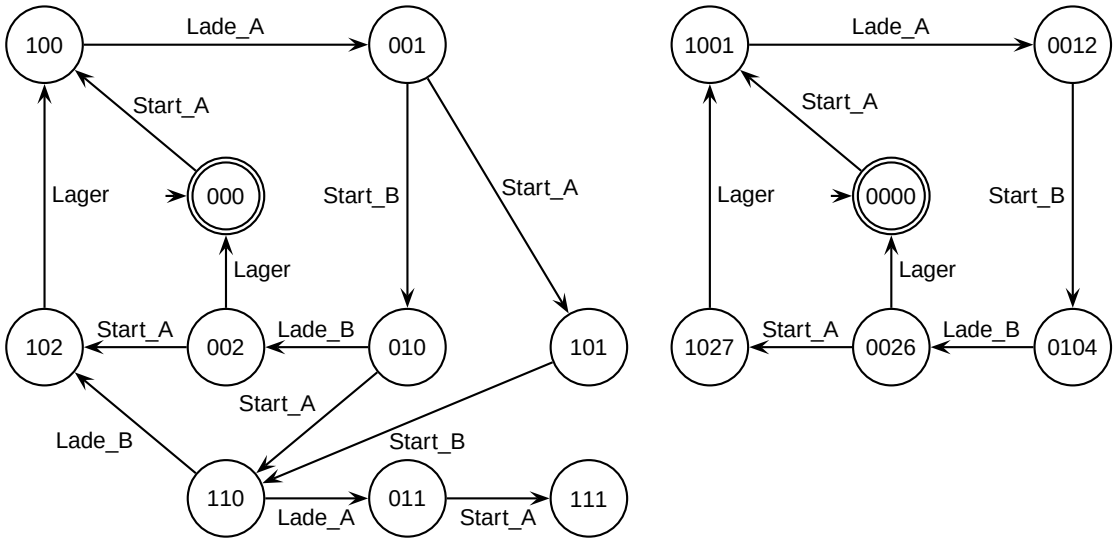


Abbildung 2.19: Die Automaten $\mathcal{A}_\mathcal{P}$ und $\mathcal{A}_\mathcal{S}$ zu Beginn der 2. Iteration

2.5.2 Der Ansatz von Kumar und Garg

Kumar und Garg haben in [51] gezeigt, dass die Synthese auch mit folgendem Algorithmus möglich ist:

Algorithmus 2.20 *Gegeben seien die Automaten $\mathcal{A}_\mathcal{P}$ und $\mathcal{A}_\mathcal{E}$ für das System bzw. für die Erstellung der Spezifikation und eine Partitionierung der Ereignismenge Σ in $\Sigma_c \subseteq \Sigma$ und $\Sigma_u := \Sigma \setminus \Sigma_c$.*

1. *Initialisiere $\mathcal{A}_\mathcal{S} := \mathsf{Tr}(\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E})$. Es gilt $\mathcal{A}_\mathcal{S} = \langle \Sigma, Q_\mathcal{S}, \delta_\mathcal{S}, q^0_\mathcal{S}, M_\mathcal{S} \rangle$.*
2. *Initialisiere die Menge der verbotenen Zustände Q_b:*
   - 2.1 $Q_b := \left\{(p,q) \in Q_\mathcal{S} \mid \mathsf{act}_{\mathcal{A}_\mathcal{P}}(p) \cap \Sigma_u \not\subseteq \mathsf{act}_{\mathcal{A}_\mathcal{S}}((p,q))\right\}$
   - 2.2 *Falls $Q_b = \emptyset$, gebe $\mathcal{A}_\mathcal{S}$ zurück; anderenfalls gehe zu Schritt 3.*
3. *Suche nach Vorgängern verbotener Zustände:*
   - 3.1 $Q'_b := Q_b \cup \{(p',q') \in Q_\mathcal{S} \setminus Q_b \mid \exists (p,q) \in Q_b . \exists \sigma \in \Sigma_u . \delta_\mathcal{S}((p',q'),\sigma,(p,q))\}$.
   - 3.2 *Falls $Q'_b = Q_b$, gehe zu Schritt 4; anderenfalls setze $Q_b := Q'_b$ und gehe zu Schritt 3.1.*
4. *Suche nach nicht co-erreichbaren Zuständen:*
   - 4.1 *Berechne das Komplement $\overline{Q}_b$.*
   - 4.2 *Berechne den Automaten $\mathcal{A}_\mathcal{S}|_{\overline{Q}_b}$. Die Zustandsmenge dieses Automaten ist $\overline{Q}_b$.*
   - 4.3 *Berechne den Automaten $\mathsf{Co}\left(\mathcal{A}_\mathcal{S}|_{\overline{Q}_b}\right)$ und bezeichne dessen Zustandsmenge als Q^{Co}.*
   - 4.4 *Berechne die Menge $Q''_b := \overline{Q}_b \setminus Q^{Co}$.*
5. *Falls $Q''_b = \emptyset$, gebe $\mathcal{A}_\mathcal{S}|_{\overline{Q}_b}$ zurück, ansonsten ersetze Q_b durch $Q_b \cup Q''_b$ und gehe zurück zu Schritt 3.*

Schritte 1 und 2 sind identisch mit denen in Algorithmus 2.18. In Schritt 3 unterscheidet sich jedoch der Ansatz von Kumar und Garg von dem von Wonham und Ramadge: Während letzterer in jeder Iteration den Automaten $\mathcal{A}_\mathcal{S}$ durch das Entfernen der verbotenen Zustände neu berechnet (siehe Schritt 3.2 auf Seite 23), bleibt der Automat $\mathcal{A}_\mathcal{S}$ bei Kumar und Garg die ganze Zeit unverändert (in Schritt 4.2 wird zwar der Automat $\mathcal{A}_\mathcal{S}|_{\overline{Q}_b}$ berechnet, $\mathcal{A}_\mathcal{S}$ bleibt aber trotzdem erhalten). Die Menge der verbotenen Zustände wird in Schritt 3 mit den Zuständen ergänzt, die über nicht steuerbare Ereignisse bereits als verboten erkannte Zustände erreichen können. Durch die Einschränkung auf $\overline{Q}_b$ in Schritt 4.2 können nicht co-erreichbare Zustände entstehen, die in den Schritten 4.3 und 4.4 berechnet und in Schritt 5 zu den verbotenen Zuständen hinzugefügt werden. Dadurch können wiederum neue verbotene Zustände entstehen, was die Wiederholung ab Schritt 3 notwendig macht.

Die Komplexität von Algorithmus 2.20 lässt sich wie folgt bestimmen: Sowohl das Produkt der Automaten als auch das Trimmen in Schritt 1 sind durch die obere Zeitschranke $O\left(\left|Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}\right||\Sigma|\right)$ begrenzt. Für die Ermittlung der verbotenen Zustände in Schritt 2.1 müssen alle Transitionen angeschaut werden. Dies geschieht innerhalb der gleichen oberen Zeitschranke. Die Vorgänger verbotener Zustände in Schritt 3 lassen sich über die Gesamtheit der Iterationen ebenfalls innerhalb der Zeit $O\left(\left|Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}\right||\Sigma|\right)$ berechnen. In Schritt 4.1 ist die Komplementbildung der Menge Q_b durch $O\left(\left|Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}\right|\right)$ beschränkt, die Einschränkung des Automaten auf die Menge $\bar{Q}_b$ wieder durch $O\left(\left|Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}\right||\Sigma|\right)$. Dies gilt auch für die Berechnung der co-erreichbaren Komponente von $\mathcal{A}_\mathcal{S}|_{\bar{Q}_b}$. Die Berechnung der Menge Q_b'' ist wiederum durch $O\left(\left|Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}\right|\right)$ begrenzt. Die Wiederholung kann höchstens $\left\lceil\left|Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}\right|/2\right\rceil$-mal stattfinden, zumal im schlimmsten Fall genau ein Zustand in Schritt 3 und einer in Schritt 4 pro Wiederholung gesammelt wird.

Insgesamt ergibt dies wie bei dem Ansatz von Wonham und Ramadge eine obere Zeitgrenze von $O\left(\left|Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}\right|^2|\Sigma|\right)$. Der Sonderfall $M_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}} = Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}$ wird jedoch von Kumar und Garg effizienter behandelt. Dann sind nämlich alle Zustände co-erreichbar, und alle verbotenen Zustände werden von Algorithmus 2.20 innerhalb einer Iteration durch die Schleife in Schritt 3 gefunden, was die Komplexität auf $O\left(\left|Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}\right||\Sigma|\right)$ reduziert.

Beispiel 2.21 Zur Veranschaulichung wird der Überwacher für die Fertigungszelle mit Algorithmus 2.20 berechnet. Die Automaten für das System und die Spezifikation sind wieder $\mathcal{A}_\mathcal{P}$ aus Abbildung 2.15 und $\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}$ aus Abbildung 2.16. Für die Ereignisse gilt weiterhin Σ_c ={Start_A}, Σ_u ={Start_B, Lade_A, Lade_B, Lager}. Der Algorithmus durchläuft dann folgende Schritte:

1. Der Automat $\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}$ ist bereits getrimmt. Die erste Iteration beginnt also mit den Automaten in Abbildung 2.18.
2. Die Ermittlung der verbotenen Zustände nach Bedingung 2.1 ergibt $Q_b = \{1105\}$. Der verbotene Zustand ist in Abbildung 2.18 eingefärbt dargestellt.
3. Die Suche nach weiteren verbotenen Zuständen ergibt:
   3.1 $Q_b' = \{1013, 1105\}$ (weil Start_B $\in \Sigma_u$).
   3.2 Da $Q_b' \neq Q_b$, wird $Q_b := \{1013, 1105\}$ gesetzt und auf 3.1 zurückgesprungen.
   3.1 $Q_b' = \{1013, 1105\}$.
   3.2 Da nun $Q_b' = Q_b$, geht es mit Schritt 4 weiter.
4. Die Suche nach nicht co-erreichbaren Zuständen ergibt:
   4.1 $\bar{Q}_b = \{0000, 1001, 0012, 0104, 0026, 1027\}$.
   4.2 Der Automat $\mathcal{A}_\mathcal{S}|_{\bar{Q}_b}$ ist identisch mit dem Automaten $\mathcal{A}_\mathcal{S}$ auf Abbildung 2.19.
   4.3 Auch $\mathsf{Co}\left(\mathcal{A}_\mathcal{S}|_{\bar{Q}_b}\right)$ ist identisch mit dem Automaten $\mathcal{A}_\mathcal{S}$ auf Abbildung 2.19.
   4.4 Da es keine neuen nicht co-erreichbaren Zustände gibt, ist $Q_b'' = \emptyset$.
5. Es wird der Automat auf der rechten Seite der Abbildung 2.19 zurückgegeben. □

2.5.3 Eine Variante des Ansatzes von Kumar und Garg

In diesem Abschnitt wird Algorithmus 2.20 leicht modifiziert. Wie sich in späteren Kapiteln zeigen wird, ergeben sich daraus Vorteile bei der Übersetzung der Überwachersynthese in den μ-Kalkül.

Die wesentliche Änderung betrifft Schritt 1 in Algorithmus 2.20. Dort werden durch das Trimmen sowohl die nicht erreichbaren als auch die nicht co-erreichbaren Zustände von $\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}$ entfernt. Unerreichbare Zustände haben jedoch keinen Einfluss auf die Funktionalität des Überwachers. Insofern kann das Entfernen dieser Zustände am Anfang des Algorithmus wegfallen. Im Prinzip könnte der neue Algorithmus mit einem zusätzlichen

Schritt versehen werden, der die erreichbare Komponente des Überwachers berechnet und als Ergebnis zurückgibt. Dann würde dieses wie bei Algorithmus 2.18 getrimmt sein. Unerreichbare Zustände können jedoch u.U. zu einer kompakteren Darstellung der Transitionsrelation beitragen, wenn zu diesem Zweck wie in den folgenden Kapiteln eine symbolische Darstellung gewählt wird. Deshalb wird auf die Berechnung der erreichbaren Zustände verzichtet.

Auch die nicht co-erreichbaren Zustände müssen nicht in Schritt 1 entfernt werden, solange sichergestellt ist, dass dies innerhalb der noch verbleibenden Schritte passiert. Damit beschränkt sich Schritt 1 in dem neuen Algorithmus darauf, das Produkt $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ zu bilden:

Algorithmus 2.22 *Gegeben seien die Automaten $\mathcal{A}_\mathcal{P}$ und $\mathcal{A}_\mathcal{E}$ für das System bzw. für die Erstellung der Spezifikation und eine Partitionierung der Ereignismenge Σ in $\Sigma_c \subseteq \Sigma$ und $\Sigma_u := \Sigma \setminus \Sigma_c$.*

1. *Initialisiere $\mathcal{A}_\mathcal{S} := \mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$. Es gilt $\mathcal{A}_\mathcal{S} = \langle \Sigma, Q_\mathcal{S}, \delta_\mathcal{S}, q^0_\mathcal{S}, M_\mathcal{S} \rangle$.*
2. *Suche nach nicht co-erreichbaren Zuständen:*
   - 2.1 *Berechne den Automaten $\mathsf{Co}(\mathcal{A}_\mathcal{S})$ und bezeichne dessen Zustandsmenge als Q^{Co}.*
   - 2.2 *Initialisiere $Q_b := Q_\mathcal{S} \setminus Q^{Co}$.*
3. *Ergänze die Menge Q_b mit den anfangs verbotenen Zuständen:*
   - 3.1 $Q_b := Q_b \cup \left\{(p,q) \in Q_\mathcal{S} \mid \mathsf{act}_{\mathcal{A}_\mathcal{P}}(p) \cap \Sigma_u \not\subseteq \mathsf{act}_{\mathcal{A}_\mathcal{S}}((p,q))\right\}$
   - 3.2 *Falls $Q_b = \emptyset$, gebe $\mathcal{A}_\mathcal{S}$ zurück; anderenfalls gehe zu Schritt 4.*
4. *Suche nach Vorgängern verbotener Zustände:*
   - 4.1 $Q'_b := Q_b \cup \{(p',q') \in Q_\mathcal{S} \setminus Q_b \mid \exists (p,q) \in Q_b . \exists \sigma \in \Sigma_u . \delta_\mathcal{S}((p',q'),\sigma,(p,q))\}$.
   - 4.2 *Falls $Q'_b = Q_b$, gehe zu Schritt 5; anderenfalls setze $Q_b := Q'_b$ und gehe zu Schritt 4.1.*
5. *Suche nach nicht co-erreichbaren Zuständen:*
   - 5.1 *Berechne das Komplement $\overline{Q}_b$.*
   - 5.2 *Berechne den Automaten $\mathcal{A}_\mathcal{S}|_{\overline{Q}_b}$. Die Zustandsmenge dieses Automaten ist $\overline{Q}_b$.*
   - 5.3 *Berechne den Automaten $\mathsf{Co}\left(\mathcal{A}_\mathcal{S}|_{\overline{Q}_b}\right)$ und bezeichne dessen Zustandsmenge als Q^{Co}.*
   - 5.4 *Berechne die Menge $Q''_b := \overline{Q}_b \setminus Q^{Co}$.*
6. *Falls $Q''_b = \emptyset$, gebe $\mathcal{A}_\mathcal{S}|_{\overline{Q}_b}$ zurück, ansonsten ersetze Q_b durch $Q_b \cup Q''_b$ und gehe zurück zu Schritt 4.*

Um die Äquivalenz zwischen den Algorithmen 2.22 und 2.20 in Bezug auf die co-erreichbaren Zustände zu zeigen, wird zwischen folgenden Fällen unterschieden:

1. *Der Automat $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ ist co-erreichbar.* Der Automat $\mathcal{A}_\mathcal{S}$ ist in Schritt 1 beider Algorithmen co-erreichbar, und die Menge Q_b, die Algorithmus 2.22 in Schritt 2.2 berechnet, ist leer. In Schritt 3.1 erkennt dieser Algorithmus dann alle verbotenen Zustände, die Algorithmus 2.20 in Schritt 2.1 erkennt. Ab diesem Punkt verlaufen beide Algorithmen (von den unerreichbaren Zuständen abgesehen) gleich.

2. *Der Automat $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ ist nicht co-erreichbar.* Algorithmus 2.20 erkennt in Schritt 2.1 einen Zustand $(p',q') \in Q_\mathcal{S}$ als verboten, wenn er die Bedingung $\mathsf{act}_{\mathcal{A}_\mathcal{P}}(p') \cap \Sigma_u \subseteq \mathsf{act}_{\mathcal{A}_\mathcal{S}}((p',q'))$ verletzt. Da Algorithmus 2.22 die nicht co-erreichbaren Zustände von $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ in Schritt 1 nicht entfernt, kann es vorkommen, dass (p',q') diese Bedingung in Schritt 3.1 noch erfüllt und zunächst unentdeckt bleibt. Dies ist genau dann der Fall, wenn von (p',q') ein Ereignis $\sigma \in \Sigma_u$ zu einem der nicht co-erreichbaren Zustände führt. Da diese aber in Schritt 2.2 zur Initialisierung der Menge Q_b verwendet werden, wird (p',q') zu einem Vorgänger eines Zustands $(p,q) \in Q_b$, den er über das nicht steuerbare Ereignis σ erreicht. Damit wird er in der ersten Iteration in Schritt 4.1 als verboten erkannt, und von diesem Punkt an verläuft der Algorithmus wie im Fall 1.

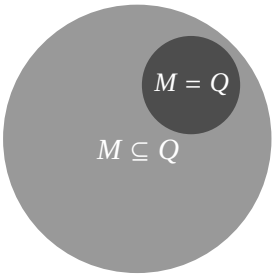
Die Komplexität der Berechnungen ist wie bei Algorithmus 2.22 $O\left(\left|Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}\right|^2 |\Sigma|\right)$ bzw. $O\left(\left|Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}\right| |\Sigma|\right)$ im Sonderfall $M_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}} = Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}$.

2.5.4 Die Synthesealgorithmen im Vergleich

Im Folgenden wird der Ansatz von Wonham und Ramadge mit dem von Kumar und Garg verglichen. Mit dem ersten ist Algorithmus 2.18 gemeint, letzterer fasst die Algorithmen 2.20 und 2.22 zusammen.

Obwohl alle Algorithmen im allgemeinen Fall dieselbe Komplexität aufweisen, haben die unterschiedlichen Vorgehensweisen Auswirkungen auf die Laufzeit. Wenn in $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ Pfade vorkommen, die in einem verbotenen Zustand enden und nur aus nicht steuerbaren Ereignissen bestehen, dann werden deren Zustände in den Algorithmen 2.20 und 2.22 in einer Schleife in den Schritten 3 bzw. 4 gesammelt. Dagegen benötigt Algorithmus 2.18 für jeden Zustand eines solchen Pfades eine ganze Iteration, die $\mathcal{A}_\mathcal{S}$ durch einen neuen Automaten ersetzt, in dem der genannte Pfad um den letzten Zustand gekürzt ist. Deshalb ist der Ansatz von Kumar und Garg der effizientere.

Wie in Abschnitt 2.5.2 erläutert, schlägt sich dieser Unterschied in dem Sonderfall $M_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}} = Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}$ sogar auf die Komplexität nieder. Dieser Zusammenhang ist in Abbildung 2.20 dargestellt. Aus Platzgründen stehen hier M und Q für $M_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}$ bzw. $Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}$.



	Wonham und Ramadge	Kumar und Garg
$M = Q$	$O\left(\lvert Q\rvert^2 \lvert\Sigma\rvert\right)$	$O\left(\lvert Q\rvert \lvert\Sigma\rvert\right)$
$M \subset Q$	$O\left(\lvert Q\rvert^2 \lvert\Sigma\rvert\right)$	$O\left(\lvert Q\rvert^2 \lvert\Sigma\rvert\right)$

Abbildung 2.20: Die Komplexitäten der Algorithmen zur Lösung des Überwachersyntheseproblems

Im Gegensatz zu Algorithmus 2.18, der immer einen getrimmten Automaten liefert, kann das Ergebnis der Algorithmen 2.20 und 2.22 unerreichbare Zustände enthalten, die durch das Entfernen der verbotenen Zustände entstehen. Diese behindern den Einsatz des Automaten als Überwacher allerdings nicht. Sollen jedoch beide Algorithmen dasselbe Ergebnis erzielen – z.B. um die Ergebnisse für Testzwecke zu vergleichen, oder um Speicherplatz zu sparen –, müssen eventuell nicht erreichbare Zustände nach dem Ablauf dieser Algorithmen entfernt werden. Die Komplexität wird dadurch nicht beeinflusst.

Ein weiterer Unterschied zwischen den beiden Ansätzen ist, dass in den Algorithmen 2.20 und 2.22 die Berechnung der verbotenen Zustände anhand der Automaten $\mathcal{A}_\mathcal{P}$ und $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ nur einmal (in den Schritten 2 bzw. 3) vorkommt, während in Algorithmus 2.18 diese in jeder Iteration wiederholt wird (Schritt 2). Dieser Unterschied wird in Kapitel 4 von Bedeutung sein.

2.6 Spezifikationsstrategien

Die Überwachersynthese muss immer dann eingesetzt werden, wenn eine gegebene Spezifikation verbotene Zustände enthält oder Verklemmungen zulässt. Es sollte jedoch nicht der Eindruck entstehen, dies seien fehlerhafte oder nachlässig erstellte Spezifikationen. Verbotene Zustände und Verklemmungen können auch dann in Spezifikationen entstehen, wenn die Anforderungen, die an das System gestellt werden, auf den ersten Blick harmlos erscheinen. Dies kommt z.B. in der Steuerung einer Telefonnummernauskunftzentrale vor, die in Abschnitt 5.4 im Detail analysiert wird.

In anderen Fällen sind verbotene Zustände in der Spezifikation sogar beabsichtigt. Oft ist es leicht, die Zustände anzugeben, die ein System während des Betriebs nicht erreichen sollte, ohne jedoch vorhersehen zu können, welche Bedingungen dafür eingehalten werden müssen. In diesem Fall genügt es, diese Zustände aus der Spezifikation zu entfernen (und dabei u.U. verbotene Zustände zu erzeugen) und die Berechnung des dann noch

möglichen Verhaltens der Überwachersynthese zu überlassen. Dies ist bereits in Beispiel 2.12 der Fall gewesen. Insbesondere in großen Systemen ist diese Vorgehensweise viel einfacher, als zu versuchen, eine korrekte Lösung nach gesundem Menschenverstand zu finden.

In anderen Fällen werden mit Absicht Verklemmungen eingebaut, die von der Syntheseprozedur entfernt werden. Diese Möglichkeit wird in dem Beispiel in Abschnitt 5.5 angewendet.

2.7 Varianten des Ramadge-Wonham-Modells

Das ursprüngliche Ramadge-Wonham-Modell in Abbildung 2.14 wurde im Laufe der Zeit in verschiedene Richtungen weiterentwickelt:

- In dem *modularen* Modell von Ramadge und Wonham [70, 87] wird die Steuerungsaktion von zwei (im Prinzip auch mehreren) Überwachern ausgeübt, die parallel auf das System einwirken. Es werden Bedingungen aufgestellt, unter denen sich ein System modular überwachen lässt. Da das Automatenprodukt der Überwacher wegfällt, kann die modulare Verteilung den Zustandsraum der Überwacher gegenüber dem ursprünglichen monolithischen Modell signifikant verkleinern.

- Lin und Wonham untersuchten in [57] das Überwachersyntheseproblem mit nicht beobachtbaren Ereignissen. Damit werden Systeme erfasst, deren Struktur es nicht zulässt, dass alle Ereignisse von dem Überwacher wahrgenommen werden. Der Ansatz ist grundlegend für die danach entwickelten verteilten und hierarchischen Modelle.

- Mit den Beiträgen von Lin bzw. Rudie und Wonham [57, 73] entstand das *verteilte* Modell, in dem mehrere Überwacher dezentral auf ein System einwirken. Dabei kann jeder Überwacher nur eine Untermenge der Ereignisse wahrnehmen und steuern. Es werden dadurch Systeme erfasst, die entkoppelte Teilsysteme haben, wie z.B. Sender und Empfänger in einem Kommunikationssystem.

- Das *hierarchische* Modell basiert auf Arbeiten mehrerer Autoren, darunter Lawford et al. [53] und Zhong bzw. Wong und Wonham [84, 89]. Ziel ist es, das System in hierarchische Ebenen zu zerlegen, die verschiedene Einsichten ermöglichen, indem jede Ebene die Aktionen der darunter liegenden Ebene als Makroschritte sieht.

- Das *vektorielle* Modell von Li und Wonham [55] erweitert die Ausdrucksfähigkeit des Ramadge-Wonham-Modells auf die von Petri-Netzen.

- Das *zeitbehaftete* Modell, basierend auf Arbeiten von Brandin und Wonham [9] ist eine Erweiterung des Modells für die Behandlung von Echtzeitsystemen.

- In dem Grundmodell liefert die Lösung des Überwachersyntheseproblems einen co-erreichbaren und steuerbaren Automaten. Mit diesen Merkmalen lassen sich Sicherheits- und zum Teil auch Lebendigkeitseigenschaften (siehe Kapitel 1) beschreiben, nicht aber Fairnesseigenschaften. Um auch letztere abzudecken, haben Thistle und Wonham [80, 81] ω-Automaten in die Überwachersynthese einbezogen. Dieser Ansatz stellt allerdings keine Verallgemeinerung des Grundmodells dar, so dass beide nicht miteinander verknüpft werden können.

Das Ziel dieser Arbeit, die Überwachersynthese mit Hilfe der temporallogischen Modellprüfung zu beschreiben und zu verallgemeinern, könnte sich im Prinzip auf all die o.g. Varianten des Ramadge-Wonham-Modells erstrecken. Dies würde jedoch den Rahmen bei weitem sprengen. Die folgenden Kapitel konzentrieren sich deshalb ausschließlich auf das Grundmodell, das in den vorherigen Abschnitten beschrieben wurde.

2.8 Verbesserungsmöglichkeiten

Die Überwachersynthese kennt auch verschiedene Grenzen. Ziel dieser Arbeit ist es, neue Ansätze für die folgenden Probleme zu bieten:

- Die Zustandsexplosion. Da die parallele Komposition und das Produkt von Automaten das Produkt deren Zustandsräume benötigen, entstehen bei den Modellen rasch Zustandsräume, deren Größe zehn- bis tausendstellige Zahlen beträgt. Eine Auflistung der entsprechenden Transitionsrelationen im Speicher jedes noch so großen Rechners ist deshalb in vielen Fällen unmöglich. In Abschnitt 2.7 wurde darauf hingewiesen, dass das modulare Modell diesem Problem entgegenwirken kann. Es ist aber nicht auf jedes beliebige Problem anwendbar. Andererseits tritt das gleiche Problem auch in der temporallogischen Modellprüfung auf. Dort hat sich inzwischen die symbolische Darstellung der Transitionsrelationen etabliert. Mit ihrer Hilfe wurden Systeme in der genannten Größenordnung in vielen Fällen erfolgreich erfasst. Da in dem hier vorgestellten Ansatz das Überwachersyntheseproblem auf die Modellprüfung zurückgeführt wird, profitiert die Überwachersynthese automatisch von der symbolischen Darstellung. Es werden dadurch viele Probleme lösbar, die vorher nicht behandelt werden konnten. Auch in der Arbeitsgruppe von Wonham wird inzwischen der Weg zur symbolischen Darstellung eingeschlagen, wie die Arbeit von Zhang [88] und andere laufende Arbeiten belegen.

- Ein weiteres Problem ist in der Formulierung des Überwachersyntheseproblems selbst zu finden. Dort wird vorausgesetzt, dass ein kleinstes akzeptables Verhalten A_{min} angegeben wird. Damit soll sichergestellt werden, dass die Einschränkungen, welche die Synthese der Spezifikation zusätzlich auferlegt, nicht zu weit gehen. Dies ist im Prinzip sinnvoll, denn bei größeren Automaten als dem im Beispiel der Fertigungszelle wird das Ergebnis schnell unübersichtlich. Wäre A_{min} jedoch bekannt, würde es die Bedingung des Überwachersyntheseproblems per Definition erfüllen und könnte bereits als Lösung dienen. Folglich müsste das Syntheseproblem gar nicht gelöst werden. In der Praxis gibt es deshalb in dem Ramadge-Wonham-Modell keine Antwort auf die Frage, ob das überwachte Verhalten eines Systems immer noch allen Anforderungen genügt. In der vorliegenden Arbeit werden zwei Möglichkeiten zur Beantwortung dieser Frage gegeben: Erstens ermöglicht die Übersetzung der Überwachersynthese in den μ-Kalkül die Anwendung der Modellprüfung auf das Ergebnis der Synthese, zumal dieses als Kripke-Struktur vorliegt (s. Kapitel 4). Zweitens ermöglicht es die Verallgemeinerung, die in Kapitel 6 erzielt wird, die zu verifizierenden Eigenschaften bereits in den Syntheseprozess zu integrieren, so dass das Ergebnis entweder die Anforderungen erfüllt oder die leere Menge ist. Eine nachträgliche Verifikation wird somit überflüssig.

- Wie in Abschnitt 2.7 erwähnt, können mit dem Grundmodell keine Fairnesseigenschaften behandelt werden. Diese spielen jedoch in reaktiven Systemen eine wichtige Rolle. Die Verallgemeinerung in Kapitel 6 behandelt Sicherheits-, Lebendigkeits-, Fortdauer- und Fairnesseigenschaften einheitlich und umfasst auch die Möglichkeiten des o.g. Ansatzes von Thistle und Wonham.

- Obwohl bereits wichtige Probleme mit Hilfe der Überwachersynthese gelöst wurden (siehe z.B. die Anwendungen von Balemi et al. [3] oder Seidl et al. [77]), ist diese nicht besonders stark verbreitet. Dies ist auf mehrere Ursachen zurückzuführen. Das Problem der Zustandsexplosion beschränkte bisher die Anwendung auf Systeme mit Zustandsräumen, die eine explizite Auflistung der Transitionsrelation zulassen. Ferner verlangt die dahinter stehende Theorie eine lange Einarbeitungszeit, die sich auch deswegen den Entwicklern aufdrängt, weil es noch keine Werkzeuge gibt, die ohne tiefere Kenntnisse der Materie zu bedienen wären. Dies erschwert die Erkennung von Überwachersyntheseproblemen, denn einerseits ist die Methode vielen Entwicklern unbekannt, andererseits wird einem fremden Experten oft nicht die nötige Einsicht in Projekte gewährt, um interessante Probleme zu erkennen. Durch die Anwendung der symbolischen Darstellung sollten jedoch die Vorteile der Überwachersynthese im Laufe der Zeit mehr und mehr an Bedeutung gewinnen und eine breitere Anwendung begünstigen.

Kapitel 3

Der μ-Kalkül

In diesem Kapitel wird der μ-Kalkül vorgestellt. Es handelt sich dabei um eine Logik, mit deren Hilfe Zustandsmengen einer Transitionsrelation beschrieben bzw. berechnet werden können. Das Verfahren wurde anfangs der 1980er Jahren von Pratt [67] und Kozen [47, 48] eingeführt und in Arbeiten von Emerson und Lei [32], Dicky [26], Arnold und Crubille [1], sowie von Cleaveland et al. [21, 22] und Schneider [75] weiterentwickelt. Die Verbindung zur automatischen Verifikation von Systemen mit endlich großem Zustandsraum ergab sich u.a. aus Erkenntnissen von Emerson und Clarke [27], wonach sich die Korrektheit diskreter Systeme mit Hilfe von Fixpunkten mathematischer Funktionen nachweisen lässt. Clarke und Emerson haben auch den Begriff der Modellprüfung (engl. *model checking*) geprägt [17], der in ähnlicher Form auch in Arbeiten von Quielle und Sifakis aus der gleichen Zeit [68, 69] zu finden ist.

Die Berechnung einer Zustandsmenge erfolgt anhand eines μ-Kalkül-Ausdrucks. Dieser kann so gestaltet werden, dass die berechnete Zustandsmenge genau die Zustände enthält, die eine gegebene, zu prüfende Eigenschaft haben. Da übliche Temporallogiken wie CTL, LTL und CTL* Untermengen des μ-Kalküls bilden, können insbesondere die in der Einleitung angesprochenen Sicherheits-, Lebendigkeits-, Fortdauer- und Fairnesseigenschaften mit Hilfe von μ-Kalkül-Ausdrücken erfasst werden. Die Modellprüfung besteht also darin, anhand eines gegebenen Systemmodells festzustellen, welche Zustände eine gegebene Eigenschaft besitzen.

Anders als bei der Überwachersynthese, die auf endlichen Automaten basiert, setzt der μ-Kalkül ein Modell des zu verifizierenden Systems in Form einer *Kripke-Struktur* voraus. Durch die *symbolische Darstellung* deren Transitionsrelation wird die Handhabung von Systemen möglich, deren Zustandsräume ansonsten die Speicherkapazität jedes Rechners übersteigen würde. Kripke-Strukturen werden in Abschnitt 3.1 vorgestellt.

Die μ-Kalkül-Ausdrücke lassen sich in Formeln und Gleichungssysteme unterteilen. Entsprechend werden in Abschnitt 3.2 der *flache* und in Abschnitt 3.3 der *vektorielle* μ-Kalkül aufgearbeitet. Es werden die Vorteile des vektoriellen μ-Kalküls gegenüber dem flachen μ-Kalkül erläutert und Algorithmen für die Lösung von Gleichungssystemen vorgestellt. Mit diesen Grundlagen kann das Ziel dieser Arbeit, die Überwachersynthese und die Modellprüfung zu vereinigen, umgesetzt werden.

3.1 Kripke-Strukturen

Kripke-Strukturen haben ihren Ursprung in Arbeiten von Kripke [49] und Hughes und Cresswell [41] über modale Logiken. Obwohl nicht speziell für die Verifikation entworfen, eignen sie sich für die Modellierung nebenläufiger Programme und diskreter Systeme. Sie werden deshalb in der Modellprüfung eingesetzt.

Kripke-Strukturen bestehen aus Zuständen, unbeschrifteten Zustandsübergängen und booleschen Variablen. In einer graphischen Darstellung werden die Zustände des Systems durch Kreise und die Transitionen durch gerichtete Kanten zwischen den Zuständen wie in Abbildung 3.1 dargestellt. Initialzustände werden durch kleine ankommende Pfeile gekennzeichnet. In Abbildung 3.1 besteht die Menge der Initialzustände lediglich aus Zustand 0.

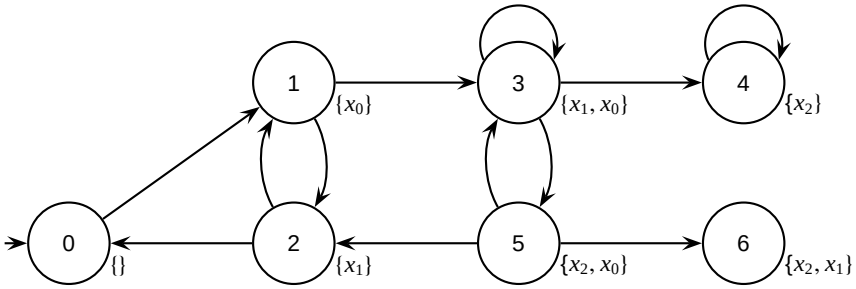


Abbildung 3.1: Eine Kripke-Struktur über den Variablen $\{x_0, x_1, x_2\}$

Außerdem werden die Zustände der Kripke-Struktur mit booleschen Variablen beschriftet. Diese Beschriftung ermöglicht es, eine Verknüpfung zwischen einem Zustand der Kripke-Struktur und einem Zustand des zu modellierenden Systems herzustellen. Um dies zu erreichen, werden die Zustände des Systems anhand einer endlichen Menge von Eigenschaften erfasst. Es sind immer so viele Eigenschaften zu wählen, dass alle modellierten Zustände des Systems eindeutig beschrieben werden können. Diese Eigenschaften werden als logische Aussagen formuliert, die entweder wahr oder falsch sein können und sich nicht weiter zerlegen lassen.

Beispiel 3.1 Eine Verkehrsampel hat die Eigenschaften „rotes Licht leuchtet“, „gelbes Licht leuchtet“ und „grünes Licht leuchtet“. Mit diesen drei Eigenschaften lassen sich insgesamt acht verschiedene Zustände beschreiben, von der ausgeschalteten Ampel bis zu dem Zustand, in dem alle Lichter an sind. Soll zusätzlich unterschieden werden, ob ein Summer bei grün dazugeschaltet ist, kommt eine weitere Eigenschaft „Summer eingeschaltet“ hinzu. □

Die genannten Aussagen sind atomare Propositionen, die durch boolesche Variablen repräsentiert werden können. Die Beschriftung eines Zustands der Kripke-Struktur erfolgt nun mit genau den booleschen Variablen, die in dem damit modellierten Zustand des Systems den Wert 1 annehmen.

Beispiel 3.2 Die Beschriftung der Zustände der Kripke-Struktur ist ebenfalls Teil der graphischen Darstellung in Abbildung 3.1, in der die Zustände mit Variablen aus der Menge $\{x_2, x_1, x_0\}$ beschriftet sind. Dort kennzeichnet z.B. Zustand 0 den Systemzustand, in dem alle drei Variablen den Wert 0 haben, in Zustand 1 gilt „$x_2 = 0, x_1 = 0, x_0 = 1$“, usw. □

Damit kann an jedem Zustand der Kripke-Struktur eindeutig abgelesen werden, in welchem Zustand sich das System befindet. Theoretisch wäre es zulässig, dass mehrere Zustände der Kripke-Struktur dieselbe Beschriftung bekommen. Dies würde jedoch lediglich bedeuten, dass die Kripke-Struktur durch Zusammenlegung der gleich beschrifteten Zustände zu minimieren wäre. Diese Möglichkeit erweitert also nicht die Modellierungsmöglichkeiten. Im Gegenteil, es ist für diese Arbeit wünschenswert, dass auch die Zustände der Kripke-Struktur eineindeutig beschriftet werden, weil dies deren Kodierung in der *symbolischen Darstellung* (s. Abschnitt 3.1.1) vereinfacht.

Folgende Definition fasst diese Erläuterungen zusammen:

Definition 3.3 (Kripke-Struktur) *Eine Kripke-Struktur über den booleschen Variablen $\mathcal{V}$ ist ein 4-Tupel $\mathcal{K} = \langle \mathcal{S}, \mathcal{I}, \mathcal{R}, \mathcal{L} \rangle$. Dabei sind:*

- *$\mathcal{S}$ die Menge der Zustände,*
- *$\mathcal{I} \subseteq \mathcal{S}$ die Menge der Initialzustände,*
- *$\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$ die Transitionsrelation,*
- *$\mathcal{L} : \mathcal{S} \to 2^{\mathcal{V}}$ eine Beschriftung der Zustände.*

In dieser Arbeit wird wie oben erklärt davon ausgegangen, dass $\mathcal{L}$ eineindeutig ist.

Da mit $|\mathcal{V}|$ booleschen Variablen $2^{|\mathcal{V}|}$ verschiedene Zustände beschriftet werden können, ist $|\mathcal{S}|$ stets eine Zweierpotenz. Allerdings werden nicht immer alle Zustände für die Modellierung gebraucht. Im Falle der Kripke-Struktur in Abbildung 3.1 sind z.B. nur sieben der insgesamt acht möglichen Zustände erreichbar. Der nicht verwendete Zustand existiert trotzdem, ist aber nicht erreichbar. Nicht erreichbare Zustände haben keinen Einfluss auf die Modellprüfung und werden deshalb in der Regel nicht in der graphischen Darstellung berücksichtigt.

3.1.1 Enumerative und symbolische Darstellung von Zustandsmengen

Die Zustände einer Kripke-Struktur $\mathcal{K} = \langle \mathcal{S}, \mathcal{I}, \mathcal{R}, \mathcal{L} \rangle$ über den Variablen $\mathcal{V}$ können sowohl über ihre Nummerierung als auch über ihre Beschriftung angesprochen werden. Daraus ergeben sich zwei Möglichkeiten, Zustandsmengen zu beschreiben.

In der *enumerativen* Darstellung werden Zustandsmengen explizit aufgelistet. In Abbildung 3.1 ist eine solche Menge z.B. $\{1, 3\}$.

In der *symbolischen* Darstellung werden Zustandsmengen durch boolesche Ausdrücke über den Variablen $\mathcal{V}$ angegeben. Zu diesem Zweck wird jeder Untermenge $Q \subseteq \mathcal{S}$ ein boolescher Ausdruck zugewiesen, der nur durch die Variablenbelegung der Zustände in Q erfüllt wird.

Beispiel 3.4 In Abbildung 3.1 wird die Zustandsmenge $\{1, 3\}$ eindeutig durch den Ausdruck $x_0 \wedge \neg x_2$ gekennzeichnet, da dieser nur durch die Variablenbelegungen der Zustände 1 und 3 erfüllt wird. □

Folgende Definition legt die Notation fest, die im Verlauf dieser Arbeit für die symbolische Darstellung verwendet wird:

Definition 3.5 (Notation für symbolische Darstellung) *Seien $\mathcal{K} = \langle \mathcal{S}, \mathcal{I}, \mathcal{R}, \mathcal{L} \rangle$ eine Kripke-Struktur und φ ein boolescher Ausdruck über den Variablen $\mathcal{V}$. Dann bezeichnet $[\![\varphi]\!]_{\mathcal{K}}$ die Zustandsmenge, deren Zustände eine Variablenbelegung haben, die den Ausdruck φ erfüllt.*

Sei darüber hinaus $Q \subseteq \mathcal{S}$ eine Zustandsmenge von $\mathcal{K}$. Dann bezeichnet φ_Q jeden booleschen Ausdruck so dass $[\![\varphi_Q]\!]_{\mathcal{K}} = Q$.

Die Notation $[\![\varphi]\!]_{\mathcal{K}}$ erlaubt es, von der symbolischen Darstellung in die enumerative zurückzukehren.

Beispiel 3.6 Aus dem vorangegangenen Beispiel folgt $[\![x_0 \wedge \neg x_2]\!]_{\mathcal{K}} = \{1, 3\}$. □

Die Notation φ_Q steht für jeden booleschen Ausdruck, der die Menge Q symbolisch darstellt. φ_Q ist nicht eindeutig, zumal es im Allgemeinen viele syntaktisch unterschiedliche Formeln gibt, die zur selben Zustandsmenge unter $[\![\cdot]\!]$ führen.

Mit dieser Notation können nun die Ausdrücke für Zustände und Zustandsmengen angegeben werden. Gegeben seien $\mathcal{K} = \langle \mathcal{S}, \mathcal{I}, \mathcal{R}, \mathcal{L} \rangle$ und $\mathcal{V} := \{x_0, x_1, \ldots, x_{k-1}\}$. Dann ist der eindeutige Ausdruck für Zustand $s_i \in \mathcal{S}$

$$\varphi_{\{s_i\}} := \bigwedge_{j=0}^{k-1} x_j^*, \tag{3.1}$$

wobei

$$x_j^* := \begin{cases} x_j & \text{falls } x_j \in \mathcal{L}(s_i) \\ \neg x_j & \text{falls } x_j \notin \mathcal{L}(s_i). \end{cases}$$

Um die Lesbarkeit der booleschen Ausdrücke zu erhöhen, wird im Folgenden $\overline{x}$ statt $\neg x$ geschrieben. Außerdem wird der Operator $\wedge$ ausgelassen, wenn er sich zwischen zwei Variablen befindet.

Beispiel 3.7 In Abbildung 3.1 sind die booleschen Ausdrücke für die Zustände 1 und 3 $\varphi_{\{1\}} = \bar{x}_2\bar{x}_1x_0$ bzw. $\varphi_{\{3\}} = \bar{x}_2x_1x_0$. □

Werden die Ausdrücke, die den Zuständen einer Menge $Q \subseteq \mathcal{S}$ entsprechen, in disjunktiver Normalform aneinandergereiht, entsteht eine Formel, die die Menge Q eindeutig kennzeichnet. Der Ausdruck für die Menge Q ist also

$$\varphi_Q := \bigvee_{q \in Q} \varphi_{\{q\}}. \tag{3.2}$$

Beispiel 3.8 Für die Menge $\{1, 3\}$ gilt $\varphi_{\{1,3\}} = \bar{x}_2\bar{x}_1x_0 \vee \bar{x}_2x_1x_0 = \bar{x}_2x_0$. □

Diese Form der Darstellung von Zustandsmengen kann auch für andere endliche Transitionsrelationen verwendet werden. Insbesondere wird sie in Kapitel 4 auf endliche Automaten übertragen.

Die symbolische Darstellung ist besonders wichtig, weil sich die Mengenoperationen Vereinigung, Schnitt und Komplement auf symbolischer Ebene durchführen lassen. Es gelten:

$$\begin{aligned} \varphi_{P\cup Q} &:= \varphi_P \vee \varphi_Q \\ \varphi_{P\cap Q} &:= \varphi_P \wedge \varphi_Q \\ \varphi_{\mathcal{S}\setminus Q} &:= \neg\varphi_Q. \end{aligned}$$

Beispiel 3.9 Seien auf Abbildung 3.1 die Mengen $M_1 = \{0, 1, 3, 4\}$, $M_2 = \{1, 2, 3, 5\}$ und $M_3 = \{0, 1, 5\}$. Zu berechnen sei $(M_1 \cup M_2) \cap \bar{M}_3$. Die Formeln für die einzelnen Mengen können in disjunktiver Normalform erstellt und dann zu folgenden Ausdrücken vereinfacht werden: $\varphi_{M_1} = \bar{x}_2x_0 \vee \bar{x}_1\bar{x}_0$, $\varphi_{M_2} = \bar{x}_2x_1 \vee \bar{x}_1x_0$ und $\varphi_{M_3} = \bar{x}_2\bar{x}_1 \vee x_2\bar{x}_1x_0$. Es folgt $(\varphi_{M_1} \vee \varphi_{M_2}) \wedge \neg\varphi_{M_3} = \bar{x}_2x_1 \vee x_2\bar{x}_1\bar{x}_0$. Wie zu erwarten, ist $[\![\bar{x}_2x_1 \vee x_2\bar{x}_1\bar{x}_0]\!]_{\mathcal{K}} = \{2, 3, 4\}$. □

3.1.2 Symbolische Darstellung der Transitionsrelation

Damit es möglich ist, Systemeigenschaften auf der Transitionsrelation einer Kripke-Struktur $\mathcal{K} = \langle \mathcal{S}, \mathcal{I}, \mathcal{R}, \mathcal{L} \rangle$ symbolisch zu verifizieren, muss auch $\mathcal{R}$ symbolisch dargestellt werden. Die Aufgabe besteht zunächst darin, jede Transition $(s, s') \in \mathcal{R}$ mit Hilfe der Formeln $\varphi_{\{s\}}$ und $\varphi_{\{s'\}}$ darzustellen. Dazu wird folgender Hilfemechanismus eingeführt: Die Menge der booleschen Variablen wird dupliziert, und jeder Zustand wird sowohl mit den Variablen $\mathcal{V}^x = \{x_0, \ldots, x_{k-1}\}$ als auch mit $\mathcal{V}^y = \{y_0, \ldots, y_{k-1}\}$ in identischer Weise beschriftet. Entsprechend wird die Notation $\varphi^x_{\{s\}}$ bzw. $\varphi^y_{\{s\}}$ verwendet, um die Variablenmenge anzugeben, mit der der Zustand s gerade kodiert wird. Die Variablen in $\mathcal{V}^x$ werden für die Kodierung des Ursprungszustands s der Transition (s, s') verwendet, die in $\mathcal{V}^y$ für den Zielzustand s'. Dann kann die Transition (s, s') symbolisch als $\varphi^x_{\{s\}} \wedge \varphi^y_{\{s'\}}$ angegeben werden, was folgende Darstellung für die Transitionsrelation $\mathcal{R}$ ermöglicht:

$$\varphi_{\mathcal{R}} := \bigvee_{(s,s') \in \mathcal{R}} \left(\varphi^x_{\{s\}} \wedge \varphi^y_{\{s'\}}\right). \tag{3.3}$$

Beispiel 3.10 Um die Transitionsrelation der Kripke-Struktur in Abbildung 3.1 symbolisch darzustellen werden die Zustände mit den Variablen $\mathcal{V}^y = \{y_0, y_1, y_2\}$ zusätzlich kodiert. Die symbolische Transitionsrelation ist dann:

$$\begin{aligned} \varphi_R = \; & \bar{x}_2\bar{x}_1\bar{x}_0\bar{y}_2\bar{y}_1y_0 \vee \bar{x}_2\bar{x}_1x_0\bar{y}_2y_1\bar{y}_0 \vee \bar{x}_2\bar{x}_1x_0\bar{y}_2y_1y_0 \vee \bar{x}_2x_1\bar{x}_0\bar{y}_2\bar{y}_1\bar{y}_0 \vee \\ & \bar{x}_2x_1\bar{x}_0\bar{y}_2\bar{y}_1y_0 \vee \bar{x}_2x_1x_0\bar{y}_2y_1y_0 \vee \bar{x}_2x_1x_0y_2\bar{y}_1\bar{y}_0 \vee \bar{x}_2x_1x_0y_2\bar{y}_1y_0 \vee \\ & x_2\bar{x}_1\bar{x}_0y_2\bar{y}_1\bar{y}_0 \vee x_2\bar{x}_1x_0\bar{y}_2y_1\bar{y}_0 \vee x_2\bar{x}_1x_0\bar{y}_2y_1y_0 \vee x_2\bar{x}_1x_0y_2y_1\bar{y}_0. \end{aligned} \tag{3.4}$$

□

Diese Darstellung ermöglicht unterschiedliche Berechnungen, zum Beispiel lassen sich die Transitionen, die von den Zuständen der Menge φ_Q ausgehen, durch $\varphi_{\mathcal{R}} \wedge \varphi_Q$ angeben.

Beispiel 3.11 Die Transitionen, die aus den Zuständen 1 und 3 der Kripke-Struktur in Abbildung 3.1 ausgehen sind

$$\begin{aligned}\varphi_{\mathcal{R}} \wedge \varphi_{\{1,3\}} = \varphi_{\mathcal{R}} \wedge \overline{x}_2 x_0 \quad = \quad & \overline{x}_2\overline{x}_1 x_0 \overline{y}_2 y_1 \overline{y}_0 \vee \overline{x}_2\overline{x}_1 x_0 \overline{y}_2 y_1 y_0 \vee \\ & \overline{x}_2 x_1 x_0 \overline{y}_2 y_1 y_0 \vee \overline{x}_2 x_1 x_0 y_2 \overline{y}_1 \overline{y}_0 \vee \\ & \overline{x}_2 x_1 x_0 y_2 \overline{y}_1 y_0.\end{aligned}$$

□

Um im vorangegangenen Beispiel die Zustände zu berechnen, die von den Zuständen in $\{1, 3\}$ innerhalb einer Transition erreicht werden können, genügt es, aus dem obigen Ergebnis die Variablen $\{x_0, x_1, x_2\}$ zu entfernen. Dies gelingt mit Hilfe der *existentiellen Quantifizierung*, die mit Hilfe der folgenden Definitionen eingeführt wird:

Definition 3.12 (Variablensubstitution) *Gegeben seien eine Kripke-Struktur $\mathcal{K} = \langle \mathcal{S}, \mathcal{I}, \mathcal{R}, \mathcal{L} \rangle$ und eine Formel φ über den Variablen $\mathcal{V}$, so dass $v_0, \ldots, v_m \in \mathcal{V}$. Seien ferner $\psi_0, \ldots, \psi_m$ Formeln über den Variablen $\mathcal{V} \cup \{0, 1\}$. Dann bezeichnet*

$$[\varphi]^{\psi_0,\ldots,\psi_m}_{v_0,\ldots,v_m}$$

die Formel, die durch das Ersetzen der Variablen v_i in φ durch die Ausdrücke ψ_i für $i = 0, \ldots, m$ entsteht. Die Ausdrücke ψ_i werden ggf. in Klammern gesetzt, um die Priorität der Operatoren nicht zu beeinflussen.

Definition 3.13 (Existentielle Quantifizierung) *Gegeben sei eine aussagenlogische Formel φ über den booleschen Variablen $\mathcal{V}$. Die existentielle Quantifizierung der Variablen $x \in \mathcal{V}$ ist der Ausdruck*

$$\exists x.\varphi := [\varphi]^0_x \vee [\varphi]^1_x .$$

Beispiel 3.14 Die Nachfolgerzustände der Menge $\{1, 3\}$ aus dem letzten Beispiel lassen sich durch die existentielle Quantifizierung der Variablen x_2, x_1 und x_0 berechnen. Das Ergebnis ist

$$\overline{y}_2 y_1 \overline{y}_0 \vee \overline{y}_2 y_1 y_0 \vee y_2 \overline{y}_1 \overline{y}_0 \vee y_2 \overline{y}_1 y_0 = \overline{y}_2 y_1 \vee y_2 \overline{y}_1.$$

Wie zu erwarten, ist $[\![\overline{y}_2 y_1 \vee y_2 \overline{y}_1]\!]_{\mathcal{K}} = \{2, 3, 4, 5\}$. □

In der Regel ist es wünschenswert, ein solches Ergebnis wieder mit den Variablen aus $\mathcal{V}^x$ zu kodieren. Dann können die berechneten Zielzustände für weitere Berechnungen wieder als Ursprungszustände dienen.

Beispiel 3.15 Die Substitution aus Definition 3.12 wird angewendet, wenn die Nachfolgerzustände der Menge $\{1, 3\}$ aus dem vorangegangenen Beispiel in Ursprungszustände zurückzuverwandeln sind. In diesem Fall werden die Variablen aus $\mathcal{V}^y$ durch jene aus $\mathcal{V}^x$ ersetzt:

$$[\overline{y}_2 y_1 \vee y_2 \overline{y}_1]^{x_1,x_2}_{y_1,y_2} = \overline{x}_2 x_1 \vee x_2 \overline{x}_1.$$

□

3.1.3 Handhabung der symbolischen Darstellung

Die Stärke der symbolischen Darstellung kommt vor allem bei großen Mengen zum Tragen, weil die Anzahl der Belegungen einer booleschen Formel exponentiell mit ihrer Länge anwächst. Dadurch kann ein kompakter Ausdruck u.U. eine sehr große Menge darstellen, zum Beispiel gilt $\varphi_{\mathcal{S}} = 1$, unabhängig von der Größe der Menge $\mathcal{S}$. Die symbolische Darstellung löst deshalb in vielen praktischen Fällen das Problem der Zustandsexplosion (s. Abschnitt 2.8).

Um die symbolischen Berechnungen durchzuführen, sind Datenstrukturen und Algorithmen notwendig, die folgenden Anforderungen genügen:

- Es muss möglich sein, mit den booleschen Variablen die Grundoperationen $\neg$ (Komplement), $\wedge$ (und-Verknüpfung) und $\vee$ (oder-Verknüpfung), sowie die Variablensubstitution durchzuführen.
- Es muss feststellbar sein, ob zwei Datenstrukturen dieselbe Funktion darstellen (z.B. um zu prüfen, ob bei einer Iteration ein Fixpunkt erreicht ist).

Die ersten erfolgreichen Ansätze stammen von Burch, Clarke und McMillan [12, 13, 14] und von Berthet, Coudert und Madre [4]. Sie bauen auf der von Bryant entwickelten kanonischen Form von binären Entscheidungsdiagrammen (kurz BDDs aus dem Englischen *binary decision diagrams*) [11] auf. Die kanonische Form begünstigt insbesondere den Vergleich zweier Diagramme, der sich innerhalb eines Programms auf den Vergleich von deren Speicheradressen reduziert. BDDs haben allerdings auch den Nachteil, dass ihre Größe stark von der Variablenordnung innerhalb des Diagramms abhängig ist. So wächst z.B. die Darstellung der booleschen Funktion eines Addierers im günstigsten Fall linear, im ungünstigsten Fall aber exponentiell mit der Anzahl der Variablen. Für andere Funktionen, wie z.B. den Multiplizierer, gibt es nur Anordnungen, die exponentiell wachsen. Darüber hinaus ist im Allgemeinen die Variablenordnung nur über heuristische Verfahren zu optimieren. Deshalb lässt sich der Rechenaufwand für ein gegebenes Problem in der Regel schwer abschätzen. Trotz dieser Nachteile konnten aber seit Beginn der 1990er Jahre mehrere Protokolle und Schaltungen verifiziert werden, wobei auch in bereits veröffentlichten Richtlinien noch subtile Fehler gefunden wurden, wie Clarke et al. in [20] berichten.

Ein neuerer Ansatz zur Handhabung der symbolischen Formeln beruht auf SAT-Lösern[1], wie erstmals von Biere et al. vorgeschlagen [6]. Obwohl der Vergleich zweier Funktionen wegen der nicht kanonischen Darstellung nun aufwändiger wird, ist der Ansatz zumindest in bestimmten Fällen effizienter als bei der Verwendung BDDs.

Die in dieser Arbeit vorgestellten Ergebnisse lassen jegliche Berechnungsmethode zu, die den o.g. Anforderungen genügt. Auf weitere Details der Implementierung wird daher nur dann eingegangen, wenn es darum geht, experimentelle Ergebnisse zu präsentieren.

3.2 Der flache μ-Kalkül

In diesem Abschnitt wird ein Teil des μ-Kalküls vorgestellt, der von den ursprünglichen Arbeiten von Pratt [67] und Kozen [47, 48] abgeleitet ist, in denen keine Gleichungssysteme vorkommen. Letztere wurden erst durch die Arbeiten von Arnold und Crubille [1] über den *vektoriellen* μ-Kalkül eingeführt. Im Gegensatz zu diesem hat der μ-Kalkül aus diesem Abschnitt keine übliche Bezeichnung. Er wird in dieser Arbeit *flacher* μ-Kalkül genannt, um die Unterscheidung von dem vektoriellen zu erleichtern. Gleichermaßen werden Formeln des flachen μ-Kalküls auch als *flache Formeln* bezeichnet. Der flache μ-Kalkül zählt zu den Grundlagen der Informatik und ist deshalb bekannter als der vektorielle. Leser, die mit diesen Grundlagen bereits vertraut sind, können sich direkt zu Abschnitt 3.3 auf Seite 44 begeben.

Im Wesentlichen handelt es sich bei dem μ-Kalkül um einen Formalismus, mit dessen Hilfe Zustandsmengen von Kripke-Strukturen beschrieben und berechnet werden können. Dazu wird die symbolische Darstellung aus Abschnitt 3.1.1 benutzt und erweitert. Insbesondere können dann auch Zustandsmengen in Formeln gefasst werden, deren Bestimmung das Durchforsten der Transitionsrelation erfordert. Solche Mengen können mit den bisher vorgestellten Mitteln nicht durch Formeln, sondern nur durch die Angabe von Algorithmen beschrieben werden.

Beispiel 3.16 Soll für die Kripke-Struktur in Abbildung 3.1 eine Formel für „alle Zustände, in denen die atomare Proposition x_0 erfüllt ist“ angegeben werden, ist die Beschriftung eines jeden Zustands bereits ausreichend um zu entscheiden, ob er zu der gesuchten Menge gehört oder nicht. Laut Abschnitt 3.1.1 lautet die Formel $\varphi = x_0$, und es folgt $[\![\varphi]\!]_{\mathcal{K}} = \{1, 3, 5\}$. In anderen Fällen aber muss die Transitionsrelation $\mathcal{R}$ mit einbezogen werden, um eine gesuchte Menge zu bestimmen. Z.B. könnte die Menge aller Zustände gesucht werden, von denen aus es möglich ist, einen Zustand zu erreichen, in dem $x_0 = 1$ gilt. Diese Menge kann berechnet werden,

[1] Abgeleitet aus dem Englischen *satisfiability problem*, das Erfüllbarkeitsproblem logischer Ausdrücke.

indem die Transitionsrelation von den Zuständen $[\![x_0]\!]_{\mathcal{K}}$ aus rückwärts durchlaufen wird. Die dabei erreichten Zustände werden zu der Initialmenge wiederholt hinzufügt, bis keine neuen Zustände mehr hinzukommen. Obwohl sich diese Berechnung mit Hilfe der in Abschnitten 3.1.1 und 3.1.2 vorgestellten Operationen auch auf symbolischer Ebene durchführen lässt, bleibt die Beschreibung der gesuchten Menge auf den angedeuteten Algorithmus beschränkt. □

Der μ-Kalkül wird durch die Syntax und die Semantik seiner Formeln definiert. Diese werden in den folgenden Definitionen festgelegt und anschließend erläutert.

Definition 3.17 (Syntax des flachen μ-Kalküls) *Gegeben sei eine Menge boolescher Variablen $\mathcal{V}$. Die Menge der μ-Kalkül-Formeln über $\mathcal{V}$ ist dann die kleinste Menge $\mathcal{L}_\mu$, die folgende Bedingungen erfüllt:*

- $\mathcal{V} \cup \{0, 1\} \subseteq \mathcal{L}_\mu$
- $\neg\varphi, \varphi \vee \psi, \varphi \wedge \psi \in \mathcal{L}_\mu$,
- $\Diamond\varphi, \boxdot\varphi \in \mathcal{L}_\mu$,
- $\kappa_\pi(\varphi) \in \mathcal{L}_\mu$,
- $\mu u.\varphi \in \mathcal{L}_\mu$,

vorausgesetzt, $\varphi, \psi \in \mathcal{L}_\mu$ und $u \in \mathcal{V}$.

Definition 3.18 (Semantik des flachen μ-Kalküls) *Gegeben sei die Kripke-Struktur $\mathcal{K} = \langle \mathcal{S}, \mathcal{I}, \mathcal{R}, \mathcal{L} \rangle$ über den booleschen Variablen $\mathcal{V}$. Jeder Formel $\Phi \in \mathcal{L}_\mu$ wird die Zustandsmenge $[\![\Phi]\!]_{\mathcal{K}} \subseteq \mathcal{S}$ wie folgt zugewiesen:*

- $[\![x]\!]_{\mathcal{K}} := \{s \in \mathcal{S} \mid x \in \mathcal{L}(s)\}$ *für alle Variablen* $x \in \mathcal{V}$
- $[\![\neg\varphi]\!]_{\mathcal{K}} := \mathcal{S} \setminus [\![\varphi]\!]_{\mathcal{K}}$
- $[\![\varphi \wedge \psi]\!]_{\mathcal{K}} := [\![\varphi]\!]_{\mathcal{K}} \cap [\![\psi]\!]_{\mathcal{K}}$
- $[\![\varphi \vee \psi]\!]_{\mathcal{K}} := [\![\varphi]\!]_{\mathcal{K}} \cup [\![\psi]\!]_{\mathcal{K}}$
- $[\![\kappa_\pi(\varphi)]\!]_{\mathcal{K}} := \pi\left([\![\varphi]\!]_{\mathcal{K}}\right)$ *für monotone Funktionen* $\pi : 2^S \to 2^S$
- $[\![\Diamond\varphi]\!]_{\mathcal{K}} := \left\{s \in \mathcal{S} \mid \exists s' \in [\![\varphi]\!]_{\mathcal{K}} . \mathcal{R}(s, s')\right\}$
- $[\![\boxdot\varphi]\!]_{\mathcal{K}} := \left\{s' \in \mathcal{S} \mid \exists s \in [\![\varphi]\!]_{\mathcal{K}} . \mathcal{R}(s, s')\right\}$
- $[\![\mu u.\varphi]\!]_{\mathcal{K}} := \bigcap \left\{Q \subseteq \mathcal{S} \mid f_\varphi(Q) \subseteq Q\right\}$, *wobei* $f_\varphi(Q) := \left[\!\left[[\varphi]_u^{\varphi_Q}\right]\!\right]_{\mathcal{K}}$.

Der Zusammenhang zwischen einer Formel φ und einem Zustand $s \in [\![\varphi]\!]_{\mathcal{K}}$ wird durch die Redewendungen φ *gilt in s* oder, was das gleiche bedeutet, *s erfüllt* φ, geschrieben $s \models \varphi$, ausgedrückt.

Um die Lesbarkeit der Formeln zu erhöhen, wird im Folgenden der Index $\mathcal{K}$ in $[\![\varphi]\!]_{\mathcal{K}}$ ausgelassen, sofern keine Verwechslungsgefahr besteht. Die nächsten Abschnitte enthalten weitere Erläuterungen zu den einzelnen Formeln aus Definition 3.18.

3.2.1 Aussagenlogische Ausdrücke

Die ersten vier Punkte in Definition 3.18 lassen aussagenlogische Ausdrücke über den Variablen einer Kripke-Struktur als μ-Kalkül-Formeln zu. Sie sind eine formale Fassung der symbolischen Darstellung aus Abschnitt 3.1.1 und bringen lediglich die dort bereits angesprochenen Mengenoperationen in den μ-Kalkül ein. Damit werden Berechnungen, die auf diesen Operationen beruhen, auf symbolischer Ebene möglich, ohne dass irgendein Zwischenresultat als Menge dargestellt werden muss.

3.2.2 Zustandstransformationsfunktionen

Die Formel $\kappa_\pi(\varphi)$ in den Definitionen 3.17 und 3.18 ist eine Ergänzung der ansonsten üblichen Definitionen des μ-Kalküls. Sie dient der Einführung benutzerdefinierter Zustandstransformationsfunktionen der Form $\pi : 2^{\mathcal{S}} \to 2^{\mathcal{S}}$ in die μ-Kalkül-Formeln. Unter Berücksichtigung der Notation aus Definition 3.5 gilt, für eine gegebene Menge $Q \subseteq \mathcal{S}$, $[\![\kappa_\pi(\varphi_Q)]\!] := \pi(Q)$. Dies bedeutet, dass die Anwendung von π auf die Menge Q ebenfalls auf symbolischer Ebene durch κ_π berechnet werden kann.

Beispiel 3.19 Für die Kripke-Struktur in Abbildung 3.1 sei $\pi(Q) := Q \cup \{0\}$. Dann ist $\kappa_\pi(\varphi_Q) = \varphi_Q \vee \overline{x}_2\overline{x}_1\overline{x}_0$. □

3.2.3 Vorgänger und Nachfolger

Von besonderem Interesse in der Modellprüfung ist die Berechnung der Vorgänger- und Nachfolgerzustände der Zustände einer gegebenen Menge, weil diese das schrittweise Durchlaufen der Transitionsrelation ermöglichen. Dazu dienen die *Modaloperatoren* $\Diamond$ und $\diamondsuit$ aus Definition 3.18. Außerdem sind folgende Abkürzungen zur Vereinfachung oft auftretender Ausdrücke nützlich:

Definition 3.20 (Modaloperatoren □ und ⊡) *Mit Hinblick auf Definition 3.17 gelten:*

$$\begin{aligned} \Box\varphi &:= \neg\Diamond\neg\varphi \\ \boxdot\varphi &:= \neg\diamondsuit\neg\varphi. \end{aligned}$$

Zur Beschreibung dieser Operatoren muss wie folgt zwischen *existentiellen* und *universellen* Vorgänger- bzw. Nachfolgerzuständen unterschieden werden:

Definition 3.21 (Vorgänger und Nachfolger) *Gegeben seien eine Kripke-Struktur $\mathcal{K} = \langle \mathcal{S}, \mathcal{I}, \mathcal{R}, \mathcal{L} \rangle$ und eine Menge $Q \subseteq \mathcal{S}$. Dann gilt:*

- $\mathsf{pre}_\exists(Q)$ *steht für die Menge der existentiellen Vorgängerzustände der Menge Q, kurz existentielle Vorgänger von Q. Das sind die Zustände, die mindestens eine ausgehende Transition haben, die zu einem Zustand in Q führt.*
- $\mathsf{suc}_\exists(Q)$ *steht für die Menge der existentiellen Nachfolgerzustände der Menge Q, kurz existentielle Nachfolger von Q. Das sind die Zustände, die mindestens eine ankommende Transition haben, die von einem Zustand in Q kommt.*
- $\mathsf{pre}_\forall(Q)$ *steht für die Menge der universellen Vorgängerzustände der Menge Q, kurz universelle Vorgänger von Q. Das sind die Zustände, die keine ausgehende Transition haben, die zu einem Zustand außerhalb Q führt.*
- $\mathsf{suc}_\forall(Q)$ *steht für die Menge der universellen Nachfolgerzustände der Menge Q, kurz universelle Nachfolger von Q. Das sind die Zustände, die keine ankommende Transition haben, die von einem Zustand außerhalb Q kommt.*

Infolge dieser Definition gelten folgende Behauptungen:

- Es ist unwichtig, ob ein existentieller Vorgänger zusätzliche ausgehende Transitionen hat, die zu Zuständen außerhalb Q führen.
- Es ist unwichtig, ob ein existentieller Nachfolger zusätzliche ankommende Transitionen hat, die von Zuständen außerhalb Q kommen.
- Bei universellen Vorgängern müssen alle ausgehenden Transitionen zu Zuständen in Q führen. Zustände ohne ausgehende Transitionen erfüllen diese Bedingung trivialerweise.
- Bei universellen Nachfolgern müssen alle ankommenden Transitionen von Zuständen in Q kommen. Zustände ohne ankommende Transitionen erfüllen diese Bedingung trivialerweise.

Beispiel 3.22 Die Bestimmung existentieller bzw. universeller Vorgänger und Nachfolger kann mit Hilfe der Kripke-Struktur in Abbildung 3.1 veranschaulicht werden. Sei $Q = \{3, 4\}$. Dann gelten: $\mathsf{pre}_\exists(Q) = \{1, 3, 4, 5\}$, $\mathsf{suc}_\exists(Q) = \{3, 4, 5\}$, $\mathsf{pre}_\forall(Q) = \{4, 6\}$ und $\mathsf{suc}_\forall(Q) = \{4, 5\}$. □

In diesem Beispiel wurden die Vorgänger und Nachfolger anhand der enumerativen Darstellung der Zustände und der Transitionen in Abbildung 3.1 bestimmt. Mit Hilfe der Definitionen 3.5, 3.18, 3.20 und 3.21 lässt sich zeigen, dass sich Vorgänger und Nachfolger symbolisch wie folgt ausdrücken lassen:

Lemma 3.23 (Vorgänger und Nachfolger) *Sei $\mathcal{K} = \langle \mathcal{S}, \mathcal{I}, \mathcal{R}, \mathcal{L} \rangle$ eine Kripke-Struktur und φ_Q eine aussagenlogische Formel über deren Variablen, so dass $[\![\varphi_Q]\!]_\mathcal{K} = Q$. Dann gelten:*

$$\mathsf{pre}_\exists(Q) = [\![\Diamond\varphi_Q]\!]_\mathcal{K} \tag{3.5}$$
$$\mathsf{suc}_\exists(Q) = [\![\diamond\varphi_Q]\!]_\mathcal{K} \tag{3.6}$$
$$\mathsf{pre}_\forall(Q) = [\![\Box\varphi_Q]\!]_\mathcal{K} \tag{3.7}$$
$$\mathsf{suc}_\forall(Q) = [\![\boxdot\varphi_Q]\!]_\mathcal{K} \tag{3.8}$$

Damit lässt sich anhand der Definitionen und Beispiele in Abschnitt 3.1.2 die Berechnung der Modaloperatoren ableiten:

Lemma 3.24 (Symbolische Berechnung der Modaloperatoren) *Gegeben seien eine Kripke-Struktur $\mathcal{K} = \langle \mathcal{S}, \mathcal{I}, \mathcal{R}, \mathcal{L} \rangle$ und eine symbolische Darstellung $\varphi_\mathcal{R}$ ihrer Transitionsrelation über den Variablen $\mathcal{V}^x = \{x_0, \ldots, x_{k-1}\}$ bzw. $\mathcal{V}^y = \{y_0, \ldots, y_{k-1}\}$. Für die Zustandsmenge $Q \subseteq \mathcal{S}$ bzw. ihre symbolische Darstellung φ_Q über $\mathcal{V}^x$ gelten dann:*

$$\Diamond\varphi_Q = \exists y_{k-1} \ldots \exists y_0 . \varphi_R \wedge [\varphi_Q]^{y_0,\ldots,y_{k-1}}_{x_0,\ldots,x_{k-1}} \tag{3.9}$$
$$\diamond\varphi_Q = [\exists x_{k-1} \ldots \exists x_0 . \varphi_R \wedge \varphi_Q]^{x_0,\ldots,x_{k-1}}_{y_0,\ldots,y_{k-1}} . \tag{3.10}$$

Die Berechnung der Modaloperatoren □ und ⊡ ergibt sich aus Definition 3.20. Folgendes Lemma fasst noch einige nützliche Terme zusammen, die aus den Definitionen 3.18 und 3.20 folgen:

Lemma 3.25 (Besondere Vorgänger- und Nachfolgerterme) *Für die Kripke-Struktur $\mathcal{K} = \langle \mathcal{S}, \mathcal{I}, \mathcal{R}, \mathcal{L} \rangle$ seien die Formeln φ_{suc} und φ_{pre} so definiert, dass Zustand $s \in [\![\varphi_{suc}]\!]$ genau dann, wenn s einen Nachfolgerzustand hat, und $s \in [\![\varphi_{pre}]\!]$ genau dann, wenn s einen Vorgängerzustand hat. Dann gelten:*

$$\Diamond 0 = 0 \tag{3.11}$$
$$\Diamond 1 = \varphi_{suc} \tag{3.12}$$
$$\Box 0 = \neg\varphi_{suc} \tag{3.13}$$
$$\Box 1 = 1 \tag{3.14}$$
$$\diamond 0 = 0 \tag{3.15}$$
$$\diamond 1 = \varphi_{pre} \tag{3.16}$$
$$\boxdot 0 = \neg\varphi_{pre} \tag{3.17}$$
$$\boxdot 1 = 1 \tag{3.18}$$

Die Operatoren ◇, ⋄, □ und ⊡ beziehen sich auf Vorgänger und Nachfolger unabhängig davon, ob diese sich auf endlichen oder unendlichen Pfaden der Kripke-Struktur befinden. Dies ist der gebräuchliche Ansatz in der Literatur über den μ-Kalkül und auch der, der in den weiteren Kapiteln verfolgt wird. Der Vollständigkeit halber sei erwähnt, dass in der formalen Verifikation reaktiver Systeme die Modaloperatoren üblicherweise so definiert werden, dass ausschließlich Zustände, die auf unendlichen Pfaden liegen, berücksichtigt werden. Die Operatoren ◇, ⋄, □ bzw. ⊡ werden dann EX, EY, AX bzw. AY geschrieben. Durch den Ausschluss der endlichen Pfade ergeben sich z.T. subtile Unterschiede in der Semantik der Operatoren. So enthält z.B. die Menge $[\![\mathsf{AX}\,\varphi]\!]$ im Gegensatz zu $[\![\Box\varphi]\!]$ nicht die Zustände, die keine ausgehenden Transitionen haben, und damit gilt $[\![\mathsf{AX}\,\varphi]\!] \subseteq [\![\mathsf{EX}\,\varphi]\!]$, während die entsprechende Ungleichung unter Berücksichtigung endlicher Pfade $[\![\Box\varphi]\!] \subseteq [\![\Diamond\varphi \vee \Box 0]\!]$ ist.

3.2.4 Fixpunkte

Der letzte Punkt in Definition 3.18 lautet $[\![\mu u.\varphi]\!]_{\mathcal{K}} := \bigcap \{Q \subseteq \mathcal{S} \mid f_\varphi(Q) \subseteq Q\}$, wobei $f_\varphi(Q) := \left[\!\left[[\varphi]_u^{\varphi_Q}\right]\!\right]_{\mathcal{K}}$. Die Schreibweise $[\varphi]_u^{\varphi_Q}$ ist eine Anwendung von Definition 3.12 und bedeutet, dass die Variable u in der Formel $\varphi(u)$ durch die Formel φ_Q zu ersetzen ist. Variablen, die – wie u – unmittelbar nach einem Fixpunktoperator stehen und als Platzhalter innerhalb der Funktion φ dienen, werden als *gebundene Variablen* bezeichnet und dürfen nicht zur Beschriftung von Zuständen herangezogen werden. Dies ist jedoch keine Einschränkung, zumal die Menge $\mathcal{V}$ immer mit solchen Platzhaltervariablen ergänzt werden kann.[2] Variablen, die nicht durch einen Fixpunktoperator gebunden sind, werden als *freie* Variablen bezeichnet. Formeln der Form $\mu u.\varphi$ werden in dieser Arbeit *Fixpunktformeln* genannt.

Um die Definition von $[\![\mu u.\varphi]\!]_{\mathcal{K}}$ zu interpretieren, wird ein Ergebnis benötigt, das aus dem Tarski-Knaster-Theorem für Fixpunkte monotoner Funktionen auf Verbänden folgt.

Definition 3.26 (Monotonie) *Gegeben sei eine endliche Menge $\mathcal{S}$. Die Funktion $f := 2^{\mathcal{S}} \to 2^{\mathcal{S}}$ ist monoton genau dann, wenn*

$$\forall Q_1, Q_2 \subseteq \mathcal{S}. Q_1 \subseteq Q_2 \to f(Q_1) \subseteq f(Q_2).$$

Das Tarski-Knaster-Theorem wird in Anhang A ausführlich erläutert. Für die Modellprüfung ist es ausreichend, das Theorem wie folgt auf endliche Potenzmengen einzuschränken[3]:

Theorem 3.27 (Tarski-Knaster-Theorem) *Seien $\mathcal{P}$ eine endliche Menge und $f : 2^{\mathcal{P}} \to 2^{\mathcal{P}}$ eine monotone Funktion auf dem Verband $(2^{\mathcal{P}}, \subseteq)$. Dann hat f Fixpunkte, und diese bilden einen vollständigen Verband mit folgendem Minimum bzw. Maximum:*

$$\begin{array}{rcccl} \check{x} & = & \bigcap\{X \subseteq \mathcal{P} | f(X) = X\} & = & \bigcap\{X \subseteq \mathcal{P} | f(X) \subseteq X\} \\ \hat{x} & = & \bigcup\{X \subseteq \mathcal{P} | f(X) = X\} & = & \bigcup\{X \subseteq \mathcal{P} | X \subseteq f(X)\}. \end{array}$$

Seien ferner $P, Q \in \mathcal{P}$ Elemente, die folgende Bedingungen erfüllen:

$$\begin{array}{llll} (\mu_1) & P \sqsubseteq f(P) & (\nu_1) & f(Q) \sqsubseteq Q \\ (\mu_2) & P \sqsubseteq \check{x} & (\nu_2) & \hat{x} \sqsubseteq Q. \end{array}$$

Dann können $\check{x}$ bzw. $\hat{x}$ iterativ berechnet werden, indem $X_0 := P$ bzw. $X_0 := Q$ als Ausgangspunkt gewählt wird. Die Folge $X_{i+1} := f(X_i)$ erreicht dann den gesuchten Fixpunkt.

Obwohl Definition 3.18 nicht verlangt, dass f_φ monoton ist, ist ihre Anwendung nur in diesem Fall sinnvoll. Deshalb wird stets davon ausgegangen, dass f_φ diese Eigenschaft hat. Somit ist laut Theorem 3.27 $[\![\mu u.\varphi]\!]_{\mathcal{K}}$ als der kleinste Fixpunkt der Funktion f_φ zu interpretieren. Um die Monotonie von f_φ zu gewährleisten genügt es, dass die gebundenen Variablen in φ stets unter einer geraden Anzahl von Negationen vorkommen. Insbesondere gilt:

Lemma 3.28 (Monotonie von f_φ) *Sei φ eine Formel über den Variablen einer Kripke-Struktur, die die gebundene Variable u enthält, und $f_\varphi(Q) = \left[\!\left[[\varphi]_u^{\varphi_Q}\right]\!\right]$ die dazugehörige Zustandstransformationsfunktion. Falls u in φ nur unter einer geraden Anzahl von Negationen vorkommt, ist f_φ monoton. Umgekehrt gilt: Falls f_φ monoton ist, dann gibt es eine Schreibweise für φ, in der u nur unter einer geraden Anzahl von Negationen vorkommt.*

Ein vollständiger Beweis dieses Lemmas würde den Rahmen dieser Arbeit sprengen. Deshalb wird die Vorgehensweise nur wie folgt angedeutet: Wird φ in die disjunktive Normalform (DNF) gebracht, dann kommt u in jedem Minterm höchstens einmal vor. Ist u stets positiv (d.h., nicht negiert), so muss f_φ monoton sein. Weicht

[2] Es ist ebenfalls möglich, die gleiche Variable sowohl für die Beschriftung als auch als Platzhalter zuzulassen. Um beide Bedeutungen auseinanderzuhalten werden dann, ähnlich wie bei lokalen Variablen in Programmiersprachen, Gültigkeitsbereiche eingeführt. Beide Ansätze sind äquivalent.

[3] Dieses Theorem entspricht Korollar A.34.

nun die Darstellung von der DNF ab, so kann u immer nur unter einer geraden Anzahl von Negationen stehen. Umgekehrt gilt: ist f_φ monoton, so gibt es für φ mindestens die DNF, in der u ausschließlich in positiver Form (und somit unter einer geraden Anzahl von Negationen) vorkommt.

Im Folgenden werden Formeln φ, so dass f_φ monoton ist, ebenfalls monoton genannt.

Der größte Fixpunkt von f_φ wird mit $[\![\nu u.\varphi]\!]_\mathcal{K}$ bezeichnet. Es lässt sich zeigen, dass er mit Hilfe des kleinsten Fixpunktes wie folgt definiert werden kann:

Lemma 3.29 (Symbolische Darstellung des größten Fixpunktes) *Sei $\mathcal{K} = \langle \mathcal{S}, \mathcal{I}, \mathcal{R}, \mathcal{L} \rangle$ eine Kripke-Struktur und $f_\varphi : 2^\mathcal{S} \to 2^\mathcal{S}$ die monotone Funktion $f_\varphi(Q) := \left[\!\left[[\varphi]_u^{\varphi_Q} \right]\!\right]_\mathcal{K}$. Sei auch*

$$\nu u.\varphi := \neg \mu u. \left[\neg \varphi\right]_u^{\neg u}.$$

Dann ist $[\![\nu u.\varphi]\!]$ der größte Fixpunkt von f_φ. Außerdem gilt

$$\mu u.\varphi := \neg \nu u. \left[\neg \varphi\right]_u^{\neg u}.$$

Fixpunkte monotoner Zustandstransformationsfunktionen sind für die Modellprüfung interessant, weil sich damit wichtige Zustandsmengen repräsentieren lassen.

Beispiel 3.30 Mit Hilfe der Fixpunktberechnung kann Beispiel 3.16 neu behandelt werden. Es geht darum, auf der Kripke-Struktur in Abbildung 3.1 die Menge der Zustände zu bestimmen, von denen aus es möglich ist, in einen Zustand zu gelangen, auf dem die atomare Proposition x_0 erfüllt ist. Wie aus der Abbildung hervorgeht, ist dies die Menge $\{0, 1, 2, 3, 5\}$. Die Vorgehensweise zur Berechnung dieser Menge in Beispiel 3.16 entspricht der Berechnung des kleinsten Fixpunktes der Funktion

$$f_\varphi(Q) := \mathsf{pre}_\exists(Q) \cup \{1, 3, 5\}.$$

Der Anfangswert für die Iteration muss die Bedingungen μ_1 und μ_2 in Theorem 3.27 erfüllen, was z.B. bei $Q^0 = \emptyset$ der Fall ist. Die Berechnung liefert nach folgenden Schritten das erwartete Ergebnis:

$$\begin{aligned} Q^1 &= f_\varphi\left(Q^0\right) \\ &= \mathsf{pre}_\exists(\emptyset) \cup \{1, 3, 5\} \\ &= \emptyset \cup \{1, 3, 5\} \\ &= \{1, 3, 5\} \end{aligned}$$

$$\begin{aligned} Q^2 &= f_\varphi\left(Q^1\right) \\ &= \mathsf{pre}_\exists(\{1, 3, 5\}) \cup \{1, 3, 5\} \\ &= \{0, 1, 2, 3, 5\} \cup \{1, 3, 5\} \\ &= \{0, 1, 2, 3, 5\} \end{aligned}$$

$$\begin{aligned} Q^3 &= f_\varphi\left(Q^2\right) \\ &= \mathsf{pre}_\exists(\{0, 1, 2, 3, 5\}) \cup \{1, 3, 5\} \\ &= \{0, 1, 2, 3, 5\} \cup \{1, 3, 5\} \\ &= \{0, 1, 2, 3, 5\} \\ &= Q^2. \end{aligned}$$

Die gleiche Iteration kann auch symbolisch durchgeführt werden. Dazu muss zunächst aus $f_\varphi(Q)$ durch Anwendung von Lemma 3.23 bzw. Definition 3.18 die Formel φ abgeleitet werden.

Beispiel 3.31 Sei $f_\varphi(Q)$ der Ausdruck aus dem vorangegangenen Beispiel. Dann gilt

$$\begin{aligned} f_\varphi(Q) &= \mathsf{pre}_\exists(Q) \cup \{1,3,5\} \\ &= [\![\Diamond\varphi_Q]\!] \cup [\![\varphi_{\{1,3,5\}}]\!] \\ &= [\![\Diamond\varphi_Q]\!] \cup [\![x_0]\!] \\ &= [\![\Diamond\varphi_Q \vee x_0]\!] \\ &= \left[\!\!\left[[\Diamond u \vee x_0]_u^{\varphi_Q}\right]\!\!\right], \end{aligned}$$

und aus Definition 3.18 folgt

$$\varphi = \Diamond u \vee x_0. \tag{3.19}$$

□

Folgendes Lemma unterstützt dann die symbolische Berechnung der Fixpunkte:

Lemma 3.32 (Symbolische Berechnung von Fixpunkten) *Gegeben seien eine Kripke-Struktur $\mathcal{K} = \langle \mathcal{S}, \mathcal{I}, \mathcal{R}, \mathcal{L}\rangle$ und eine μ-Kalkül-Formel φ, sowie die in Definition 3.18 festgelegte monotone Funktion $f_\varphi : 2^\mathcal{S} \to 2^\mathcal{S}$. Sei $u^0 := 0$ bzw. $u^0 := 1$. Dann konvergiert die Folge*

$$u^{i+1} := [\varphi]_u^{u^i}$$

zu $\mu u.\varphi$ bzw. zu $\nu u.\varphi$.

Beweis: Laut Theorem 3.27 konvergiert die Folge $Q^{i+1} := f_\varphi\left(Q^i\right)$ zu dem kleinsten bzw. größten Fixpunkt von f_φ je nachdem, ob $Q^0 := \emptyset$ bzw. $Q^0 := \mathcal{S}$ gewählt wird. Es genügt also zu zeigen, dass $\left[\!\left[u^i\right]\!\right] = Q^i$ für jedes i. Für den Ausgangspunkt u^0 ist diese Bedingung erfüllt, zumal $u^0 = 0 = \varphi_\emptyset$ bzw. $u^0 = 1 = \varphi_\mathcal{S}$. Angenommen, es gilt $\left[\!\left[u^{i-1}\right]\!\right] = Q^{i-1}$ und somit $u^{i-1} = \varphi_{Q^{i-1}}$, folgt:

$$\left[\!\left[u^i\right]\!\right] = \left[\!\left[[\varphi]_u^{u^{i-1}}\right]\!\right] = \left[\!\left[[\varphi]_u^{\varphi_{Q^{i-1}}}\right]\!\right] = f_\varphi\left(Q^{i-1}\right) = Q^i.$$

□

Lemma 3.32 besagt, dass die einzelnen Iterationsschritte auf symbolischer Ebene genau den Schritten der Berechnung mit enumerativ dargestellten Mengen entsprechen. Da die Zwischenresultate in der Regel nicht gebraucht werden, muss die Umwandlung in die enumerative Mengendarstellung – wenn überhaupt – nur am Ende der Iteration stattfinden. Somit können auch Fixpunkte rein symbolisch berechnet werden. Im Folgenden wird, wenn vorteilhaft, auch die Notation $\varphi\left(u^i\right)$ an Stelle von $[\varphi]_u^{u^i}$ verwendet. Der Fixpunkt wird im Folgenden mit u^∞ bezeichnet.

Beispiel 3.33 Der kleinste Fixpunkt der Funktion f_φ der zwei letzten Beispiele lässt sich mit Hilfe von Gleichung 3.19 und $u^0 = 0$ sowie der Transitionsrelation $\varphi_\mathcal{R}$ aus Gleichung 3.4 wie folgt symbolisch berechnen:

$$\begin{aligned} u^1 &= \varphi\left(u^0\right) \\ &= \Diamond 0 \vee x_0 \\ &\overset{(3.11)}{=} 0 \vee x_0 \\ &= x_0 \end{aligned}$$

$$\begin{aligned} u^2 &= \varphi\left(u^1\right) \\ &= \Diamond x_0 \vee x_0 \\ &\overset{(3.9)}{=} \exists y_2.\exists y_1.\exists y_0.\varphi_R \wedge [x_0]_{x_0,x_1,x_2}^{y_0,y_1,y_2} \vee x_0 \\ &= \overline{x_2} \vee \overline{x_1} x_0 \vee x_0 \\ &= \overline{x_2} \vee x_0 \end{aligned}$$

$$
\begin{aligned}
u^3 &= \varphi\left(u^2\right) \\
&= \Diamond(\overline{x_2} \vee x_0) \vee x_0 \\
&\overset{(3.9)}{=} \exists y_2.\exists y_1.\exists y_0.\varphi_R \wedge [\overline{x_2} \vee x_0]^{y_0,y_1,y_2}_{x_0,x_1,x_2} \vee x_0 \\
&= \overline{x_2} \vee \overline{x_1} x_0 \vee x_0 \\
&= \overline{x_2} \vee x_0 \\
&= u^\infty.
\end{aligned}
$$

Wie zu erwarten, ist $[\![u^\infty]\!] = \{0, 1, 2, 3, 5\}$. □

3.2.5 Formeln mit verschachtelten Fixpunkten

Im vorangegangenen Beispiel hatte die Formel φ nur eine gebundene Variable. Definition 3.17 lässt jedoch Fixpunktformeln der Form $\mu u.\varphi$ zu, wobei φ eine beliebige μ-Kalkül-Formel sein darf und deshalb selbst Fixpunktformeln enthalten kann. Dadurch können Ausdrücke wie z.B.

$$\mu u.(\Diamond u \vee x_1 \vee \mu u.\Box u \vee x_2)$$

entstehen. Um einen solchen Ausdruck auszuwerten, müssen die Gültigkeitsbereiche der beiden Erscheinungen von μu auseinander gehalten werden. Dieser Aufwand kann jedoch durch die Verwendung unterschiedlicher Variablen vermieden werden, wie in

$$\mu u_1.(\Diamond u_1 \vee x_1 \vee \mu u_2.\Box u_2 \vee x_2). \tag{3.20}$$

Diese Unterscheidung ist besonders dann nützlich, wenn die verschachtelten Fixpunkte voneinander abhängen, wie in dem Ausdruck

$$\mu u_1.(\Diamond u_1 \vee x_1 \vee \mu u_2.\Box u_2 \vee x_2 \wedge u_1). \tag{3.21}$$

Hier kommt die Variable u_1 aus dem äußeren Fixpunkt μu_1 innerhalb des inneren Fixpunktes μu_2 vor, und die Verwendung unterschiedlicher Variablennamen erspart weitere Erklärungen. Im Folgenden wird bei derart verschachtelten Formeln deshalb stets die letzte Form vorausgesetzt, so dass keine Gültigkeitsbereiche explizit angegeben werden müssen. Die Ausdrucksfähigkeit der Formeln wird dadurch nicht beeinträchtigt. Sind die Gültigkeitsbereiche jedoch klar voneinander getrennt, wie z.B. in

$$\mu u_1.\Diamond u_1 \vee x_1 \wedge ((\mu u_2.\Diamond u_2 \vee x_2 \wedge \nu u_3.\Box u_3 \wedge x_3 \vee u_1) \vee \nu u_3.\Box u_3 \wedge x_3 \vee u_1), \tag{3.22}$$

ist die Wiederverwendung gebundener Variablen (wie in diesem Fall u_3) erlaubt.

Bei verschachtelten Fixpunkten, z.B. $\sigma_1 u_1.\varphi_1(u_1, \sigma_2 u_2.\varphi_2(u_1, u_2))$, mit $\sigma_1, \sigma_2 \in \{\mu, \nu\}$, verhält sich der innere Fixpunkt $\sigma_2 u_2.\varphi_2(u_1, u_2)$ wie ein monotoner Ausdruck in u_1, der für jeden Wert $u_1^{i_1}$ den Wert $u_2^{i_1,\infty}$ ergibt. Letzterer wird für die Berechnung des Schrittes $u_1^{i_1+1}$ der Iteration für φ_1 benötigt, was zu einer Verschachtelung der Berechnungen führt.

Beispiel 3.34 Die verschachtelte Fixpunktformel $\sigma_1 u_1.\varphi_1(u_1, \sigma_2 u_2.\varphi_2(u_1, u_2))$ kann mit Hilfe folgender iterativen Formeln berechnet werden:

$$\left[\begin{array}{lcl} u_1^{i_1+1} & := & \varphi_1(u_1^{i_1}, u_2^{i_1,\infty}) \\ u_2^{i_1,i_2+1} & := & \varphi_2(u_1^{i_1}, u_2^{i_1,i_2}). \end{array}\right. \tag{3.23}$$

Die Initialwerte sind $u_1^0 = 0$ bzw. $u_1^0 = 1$ und, für alle i_1, $u_2^{i_1,0} := 0$ bzw. $u_2^{i_1,0} := 1$, je nachdem, ob σ_1 und σ_2 kleinste oder größte Fixpunkte bedeuten. □

Bei einer solchen Berechnung entstehen für eine Formel mit k verschachtelten Fixpunkten k verschachtelte Iterationsschleifen, die zu einer Zeitkomplexität $O\left(|\mathcal{R}|^k\right)$ führen. Wie folgendes Beispiel zeigt, ist dies aber oft ineffizient.

Beispiel 3.35 Zur Auswertung des Ausdrucks (3.20) ist folgende Iteration mit $u_1^0 = u_2^{i_1,0} := 0$ durchzuführen:

$$\left[\begin{array}{lcl} u_1^{i_1+1} & := & \Diamond u_1^{i_1} \vee x_1 \vee u_2^{i_1,\infty} \\ u_2^{i_1,i_2+1} & := & \Box u_2^{i_1,i_2} \vee x_2. \end{array}\right.$$

Da der zweite Ausdruck in diesem Fall nicht von $u_1^{i_1}$ abhängt, ergibt sich für jedes i_1 der gleiche Fixpunkt $u_2^{i_1,\infty} = u_2^{\infty}$. Es würde also genügen, letzteren ein Mal zu berechnen und dann als konstanten Faktor in den Ausdruck für u_1^{i+1} einzusetzen:

$$u_1^{i+1} := \Diamond u_1^{i_1} \vee x_1 \vee u_2^{\infty}.$$

□

Das Beispiel zeigt, dass die Verschachtelung von Fixpunkten, die nicht gegenseitig voneinander abhängen, die Zeitkomplexität der Berechnungen nicht erhöhen muss. Davon kann die Berechnungsweise nach Definition 3.18 aber keinen Nutzen ziehen, weil diese stets zu Formeln wie 3.23 führt. Eine Verringerung der Komplexität und andere Optimierungen werden jedoch durch den vektoriellen μ-Kalkül ermöglicht.

3.3 Der vektorielle μ-Kalkül

Der vektorielle μ-Kalkül beruht auf Arbeiten von Arnold und Crubille [1] sowie auf Beiträgen von Cleaveland et al. [21] und Schneider [75], in denen die hier benutzte Darstellungsform eingeführt wird. Insbesondere enthält [75] eine lückenlose Folge formaler Ergebnisse, die zur Definition von Syntax und Semantik des vektoriellen μ-Kalküls und zu den Berechnungsmöglichkeiten von dessen Ausdrücken führen. Dieser Ansatz ist zu umfangreich, als dass er hier wiedergegeben werden könnte. Die hier gewählte Darstellung verfolgt deshalb den gleichen Gedankengang, ersetzt aber nach Möglichkeit formale Ergebnisse durch Beispiele und informale Argumente. Der vektorielle μ-Kalkül ist das Hauptwerkzeug für die Herleitung aller Ergebnisse in dieser Arbeit.

3.3.1 Syntax und Semantik

Im vektoriellen μ-Kalkül werden die Formeln aus dem flachen μ-Kalkül mit Hilfe von Gleichungssystemen dargestellt, um neue Möglichkeiten für die Berechnungen der Fixpunkte zu eröffnen. Die Semantik einer gegebenen flachen Formel soll dabei unberührt bleiben, was zur Folge hat, dass die Fixpunkte eines Gleichungssystems, das aus einer Formel abgeleitet wurde, mit denen der Formel übereinstimmen müssen. Syntax und Semantik des vektoriellen μ-Kalküls müssen deshalb so gewählt werden, dass aus einem Gleichungssystem ein Algorithmus abgeleitet werden kann, der das gleiche Ergebnis liefert wie die entsprechende flache Formel.

Ist die Syntax einmal festgelegt, können Gleichungssysteme auf zwei Weisen entstehen: Einerseits kann ein Algorithmus angegeben werden, der eine gegebene flache Formel in ein Gleichungssystem umwandelt; andererseits kann ein Gleichungssystem aber auch auf direktem Wege formuliert werden, also ohne dass eine entsprechende Formel vorher bekannt wäre. Folgende Definition erlaubt, wie sich in den Algorithmen 3.37, 3.39, 3.41 und 3.45 zeigen wird, beide Möglichkeiten:

Definition 3.36 (Syntax eines μ-Kalkül-Gleichungssystems) *Seien $\varphi_1, \ldots, \varphi_n \in \mathcal{L}_\mu$ Formeln des flachen μ-Kalküls über den Variablen $\mathcal{V}$. Seien ferner $u_1, \ldots, u_n \in \mathcal{V}$ Variablen, die für $1 \leq i \leq n$ in den Formeln φ_i nur unter einer geraden Anzahl Negationen vorkommen. Dann ist für $\sigma_i \in \{\mu, \nu\}$*

$$\left\{\begin{array}{lcl} u_n & \overset{\sigma_n}{=} & \varphi_n(u_1, \ldots, u_n) \\ & \vdots & \\ u_1 & \overset{\sigma_1}{=} & \varphi_1(u_1, \ldots, u_n) \end{array}\right.$$

ein μ-Kalkül-Gleichungssystem.

Als nächstes werden Algorithmen angegeben, mit denen Formeln und Gleichungssysteme ineinander umgewandelt werden können. Zunächst werden nur Fixpunktformeln der Form $\mu u.\varphi$ bzw. $\nu u.\varphi$ berücksichtigt, wobei φ beliebig viele verschachtelte Fixpunktausdrücke enthalten darf. Bei der Umwandlung einer solchen Formel in ein Gleichungssystem werden die am tiefsten verschachtelten Teilformeln zuerst behandelt. Dadurch wird gewährleistet, dass für jede dieser Variablen eine Gleichung entsteht.

Algorithmus 3.37 (Umwandlung einer Fixpunktformel in ein Gleichungssystem) *Gegeben sei eine Fixpunktformel $\Phi = \sigma u.\varphi \in \mathcal{L}_\mu$ mit $\sigma \in \{\mu, \nu\}$. Das entsprechende Gleichungssystem E wird in den folgenden Schritten erstellt:*

1. *Initialisiere E mit dem leeren Gleichungssystem.*
2. *Bestimme die Verschachtelungstiefe k der Formel Φ.*
3. *Solange $k > 0$:*
   (a) *Für jede Teilformel der Form $\sigma' v.\psi$ der Verschachtelungsebene k in Φ:*
      i. *Ersetze jede Instanz von $\sigma' v.\psi$ in Φ durch die Variable v.*
      ii. *Erweitere E um die Gleichung $v \overset{\sigma'}{=} \psi$; neue Gleichungen werden unter bereits angelegte geschrieben.*
   (b) *Ersetze k durch $k - 1$ und gehe zurück zu Schritt 3.*
4. *Gebe E zurück.*

Beispiel 3.38 Sei $\Phi = \mu u_1.\Diamond u_1 \vee x_1 \wedge ((\mu u_2.\Diamond u_2 \vee x_2 \wedge \nu u_3.\Box u_3 \wedge x_3 \vee u_1) \vee \nu u_3.\Box u_3 \wedge x_3 \vee u_1)$. Nach Algorithmus 3.37 wird das Gleichungssystem zu Φ in folgenden Schritten erstellt:

1. Zu Beginn ist das Gleichungssystem E leer.
2. Φ hat die Verschachtelungstiefe 3.
3. Für $k = 3$:
   (a) Φ besitzt in der Verschachtelungsebene 3 die Teilformel $\nu u_3.\Box u_3 \wedge x_3 \vee u_1$.
      i. Diese Teilformel kommt auch in der zweiten Verschachtelungsebene vor. Beide Instanzen werden durch die Variable u_3 ersetzt (wenn es um das Ersetzen geht ist es egal, in welcher Verschachtelungsebene die Teilformeln vorkommen). Aus
      $\Phi = \mu u_1.\Diamond u_1 \vee x_1 \wedge ((\mu u_2.\Diamond u_2 \vee x_2 \wedge \underbrace{\nu u_3.\Box u_3 \wedge x_3 \vee u_1}_{=u_3}) \vee \underbrace{\nu u_3.\Box u_3 \wedge x_3 \vee u_1}_{=u_3})$ wird
      $\Phi = \mu u_1.\Diamond u_1 \vee x_1 \wedge ((\mu u_2.\Diamond u_2 \vee x_2 \wedge u_3) \vee u_3)$.
      ii. Es wird eine Gleichung für u_3 angelegt:
      $$E := \left\{ u_3 \overset{\nu}{=} \Box u_3 \wedge x_3 \vee u_1. \right.$$
   (b) $k = 2$.
3. Für $k = 2$:
   (a) Φ besitzt in der Verschachtelungsebene 2 die Teilformel $\mu u_2.\Diamond u_2 \vee x_2 \wedge u_3$.
      i. Diese Teilformel wird durch die Variable u_2 ersetzt. Aus
      $\Phi = \mu u_1.\Diamond u_1 \vee x_1 \wedge ((\underbrace{\mu u_2.\Diamond u_2 \vee x_2 \wedge u_3}_{=u_2}) \vee u_3)$ wird
      $\Phi = \mu u_1.\Diamond u_1 \vee x_1 \wedge (u_2 \vee u_3)$.

ii. Es wird eine Gleichung für u_2 angelegt:

$$E := \begin{cases} u_3 & \overset{\nu}{=} & \Box u_3 \wedge x_3 \vee u_1 \\ u_2 & \overset{\mu}{=} & \Diamond u_2 \vee x_2 \wedge u_3. \end{cases}$$

(b) $k = 1$.

3. Für $k = 1$:

(a) Φ besitzt in der Verschachtelungsebene 1 die Teilformel $\mu u_1.\Diamond u_1 \vee x_1 \wedge (u_2 \vee u_3)$.

i. Diese Teilformel wird durch die Variable u_1 ersetzt. Aus
$\Phi = \underbrace{\mu u_1.\Diamond u_1 \vee x_1 \wedge (u_2 \vee u_3)}_{=u_1}$ wird
$\Phi = u_1$.

ii. Es wird eine Gleichung für u_1 angelegt:

$$E := \begin{cases} u_3 & \overset{\nu}{=} & \Box u_3 \wedge x_3 \vee u_1 \\ u_2 & \overset{\mu}{=} & \Diamond u_2 \vee x_2 \wedge u_3 \\ u_1 & \overset{\mu}{=} & \Diamond u_1 \vee x_1 \wedge (u_2 \vee u_3). \end{cases}$$

(b) $k = 0$.

4. Es wird E zurückgegeben. □

Umgekehrt ist es ebenfalls möglich, aus einem Gleichungssystem die entsprechende Fixpunktformel abzuleiten. Dabei werden die Schritte aus Algorithmus 3.37 rückgängig gemacht, indem mit der letzten Gleichung begonnen wird und die weiter verschachtelten Teilformeln nach und nach wieder eingebaut werden:

Algorithmus 3.39 (Umwandlung eines Gleichungssystems in eine Fixpunktformel) *Gegeben sei ein Gleichungssystem der Form*

$$E := \begin{cases} u_n & \overset{\sigma_n}{=} & \varphi_n(u_1, \ldots, u_n) \\ & \vdots & \\ u_1 & \overset{\sigma_1}{=} & \varphi_1(u_1, \ldots, u_n), \end{cases}$$

wobei $\sigma_i \in \{\mu, \nu\}$. Entspricht E einer Fixpunktformel Φ der Form $\sigma_1 u_1.\varphi_1$ des flachen μ-Kalküls, kann Φ wie folgt erstellt werden:

1. *Setze $\Phi := \sigma_1 u_1.\varphi_1$ und $k := 2$.*
2. *Solange $k \leq n$*
   (a) *Ersetze u_k in Φ durch $\sigma_k u_k.\varphi_k$, ggf. in Klammern geschrieben, um die Priorität der Operatoren nicht zu beeinflussen.*
   (b) *Ersetze k durch $k + 1$ und gehe zurück zu Schritt 2.*
3. *Gebe Φ zurück.*

Beispiel 3.40 Die Umwandlung der Formel aus dem letzten Beispiel in ein Gleichungssystem lässt sich mit diesem Algorithmus schrittweise rückgängig machen. □

Die Algorithmen 3.37 und 3.39 erlauben es, zwischen Fixpunktformeln und Gleichungssystemen hin- und herzuschalten. Im Allgemeinen ist aber eine μ-Kalkül-Formel keine einfache Fixpunktformel, sondern ein boolescher Ausdruck, der Fixpunktformeln enthält, z.B. $\Phi = x_1 \wedge \mu u_1.\varphi_1 \vee x_2 \wedge \nu u_2.\varphi_2$. Soll eine solche Formel in ein Gleichungssystem übersetzt werden, wird für jede Teilformel von Φ, die eine Fixpunktformel ist, ein Block Gleichungen wie in Algorithmus 3.37 angelegt. Die Information aus Φ, die besagt, wie die einzelnen Blöcke zu

verknüpfen sind, muss zusätzlich erhalten bleiben. Dies gelingt, indem jede Fixpunktformel der Form $\sigma u.\varphi$ in Φ durch die Variable u ersetzt wird. Das Ergebnis der Umwandlung ist ein Paar (E, ϕ), wobei E das Gleichungssystem ist und ϕ die Formel Φ mit den genannten Ersetzungen. Somit entsteht folgende Verallgemeinerung von Algorithmus 3.37:

Algorithmus 3.41 (Umwandlung einer flachen μ-Kalkül-Formel in ein Gleichungssystem) *Sei $\Phi \in \mathcal{L}_\mu$ eine flache μ-Kalkül-Formel. Ihre Darstellung als Paar (E, ϕ) wird nach folgenden Regeln erstellt:*

1. *Initialisiere E mit dem leeren Gleichungssystem und setze $\phi := \Phi$.*
2. *Für jede Fixpunktformel $\phi' = \sigma u.\varphi$ von ϕ, mit $\sigma \in \{\mu, \nu\}$:*
   (a) *Ersetze ϕ' durch u in ϕ.*
   (b) *Bestimme die Verschachtelungstiefe k der Formel ϕ'.*
   (c) *Solange $k > 0$:*
      i. *Für jede Teilformel der Form $\sigma' v.\psi$ der Verschachtelungsebene k in ϕ':*
         · *Ersetze jede Instanz von $\sigma' v.\psi$ in ϕ' durch die Variable v.*
         · *Erweitere E um die Gleichung $v \overset{\sigma'}{=} \psi$; neue Gleichungen werden unter bereits angelegte geschrieben.*
      ii. *Ersetze k durch $k - 1$ und gehe zurück zu Schritt c.*
3. *Gebe (E, ϕ) zurück.*

Beispiel 3.42 Sei
$\Phi = x_1 \wedge \mu u_1.\Diamond u_1 \vee x_1 \wedge ((\mu u_2.\Diamond u_2 \vee x_2 \wedge \nu u_3.\Box u_3 \wedge x_3 \vee u_1) \vee \nu u_3.\Box u_3 \wedge x_3 \vee u_1) \vee x_4 \wedge \nu u_4.\Box u_4 \wedge x_4$.
Algorithmus 3.41 durchläuft dann folgende Schritte:

1. Zu Beginn ist das Gleichungssystem E leer und es gilt
   $\phi = x_1 \wedge \mu u_1.\Diamond u_1 \vee x_1 \wedge ((\mu u_2.\Diamond u_2 \vee x_2 \wedge \nu u_3.\Box u_3 \wedge x_3 \vee u_1) \vee \nu u_3.\Box u_3 \wedge x_3 \vee u_1) \vee x_4 \wedge \nu u_4.\Box u_4 \wedge x_4$.
2. Für die Teilformel $\phi' = \mu u_1.\Diamond u_1 \vee x_1 \wedge ((\mu u_2.\Diamond u_2 \vee x_2 \wedge \nu u_3.\Box u_3 \wedge x_3 \vee u_1) \vee \nu u_3.\Box u_3 \wedge x_3 \vee u_1)$:
   (a) Es gilt $\phi = x_1 \wedge u_1 \vee x_4 \wedge \nu u_4.\Box u_4 \wedge x_4$.
   (b) In diesem Schritt und in Schritt (c) wird ϕ' wie in Beispiel 3.38 in ein Gleichungssystem umgewandelt. Danach gilt
   $$E := \begin{cases} u_3 \overset{\nu}{=} \Box u_3 \wedge x_3 \vee u_1 \\ u_2 \overset{\mu}{=} \Diamond u_2 \vee x_2 \wedge u_3 \\ u_1 \overset{\mu}{=} \Diamond u_1 \vee x_1 \wedge (u_2 \vee u_3). \end{cases}$$
2. Für die Teilformel $\phi' = \nu u_4.\Box u_4 \wedge x_4$:
   (a) Es gilt $\phi = x_1 \wedge u_1 \vee x_4 \wedge u_4$.
   (b) ϕ' hat die Verschachtelungstiefe $k = 1$.
   (c) Für $k = 1$:
      i. ϕ' besitzt in der Verschachtelungsebene 1 die Teilformel $\nu u_4.\Box u_4 \wedge x_4$.
         · Diese Teilformel wird durch die Variable u_4 ersetzt. Aus
         $\phi' = \underbrace{\nu u_4.\Box u_4 \wedge x_4}_{=u_4}$ wird
         $\phi' = u_4$.

· Es wird eine Gleichung für u_4 angelegt:

$$E := \begin{cases} u_3 \overset{\nu}{=} \Box u_3 \wedge x_3 \vee u_1 \\ u_2 \overset{\mu}{=} \Diamond u_2 \vee x_2 \wedge u_3 \\ u_1 \overset{\mu}{=} \Diamond u_1 \vee x_1 \wedge (u_2 \vee u_3) \\ u_4 \overset{\nu}{=} \Box u_4 \wedge x_4. \end{cases}$$

ii. $k = 0$.

3. Es wird das Paar (E, ϕ) mit $\phi = x_1 \wedge u_1 \vee x_4 \wedge u_4$ zurückgegeben. □

Am aufwändigsten ist die Umkehrung eines gegebenen Paares (E, ϕ) in eine flache Formel. Zunächst ist eine syntaktische Einschränkung der Formel ϕ notwendig, denn die Umwandlung ergibt nur dann einen Sinn, wenn die Variablen $\{u_1, \ldots, u_n\}$, die auf der linken Seite der Gleichungen im Gleichungssystem E stehen, in der Formel ϕ frei sind (ansonsten könnte das Paar (E, ϕ) nicht aus einer flachen Formel mit Algorithmus 3.41 zurückgewonnen werden). Es wird fortan davon ausgegangen, das ϕ dieser Einschränkung genügt. Das Paar (E, ϕ) wird dann als *gültiges Paar* bezeichnet.

Weiterhin spielt es eine Rolle, ob die Variablen aus der Menge $\{u_1, \ldots, u_n\}$, die in ϕ vorkommen, jeweils der äußersten Verschachtelungsebene einer Teilformel entsprechen. Wie das folgende Beispiel zeigt, besteht dann die Umkehrung lediglich in einer Anwendung von Algorithmus 3.39 für jede freie Variable aus der linken Seite des Gleichungssystems, die in ϕ vorkommt.

Beispiel 3.43 Soll das Paar (E, ϕ) mit

$$E := \begin{cases} u_3 \overset{\nu}{=} \Box u_3 \wedge x_3 \vee u_1 \\ u_2 \overset{\mu}{=} \Diamond u_2 \vee x_2 \wedge u_3 \\ u_1 \overset{\mu}{=} \Diamond u_1 \vee x_1 \wedge (u_2 \vee u_3) \\ u_4 \overset{\nu}{=} \Box u_4 \wedge x_4 \end{cases}$$

und $\phi = x_1 \wedge u_1 \vee x_4 \wedge u_4$ aus dem letzten Beispiel wieder in die ursprüngliche Formel umgewandelt werden, genügt es, den Algorithmus 3.39 auf die Blöcke

$$\begin{cases} u_3 \overset{\nu}{=} \Box u_3 \wedge x_3 \vee u_1 \\ u_2 \overset{\mu}{=} \Diamond u_2 \vee x_2 \wedge u_3 \\ u_1 \overset{\mu}{=} \Diamond u_1 \vee x_1 \wedge (u_2 \vee u_3) \end{cases} \quad \text{und} \quad \left\{ u_4 \overset{\nu}{=} \Box u_4 \wedge x_4 \right.$$

anzuwenden und die Variablen u_1 und u_4 durch die daraus resultierenden Formeln in ϕ zu ersetzen. □

Das Paar (E, ϕ) kann aber auch so erstellt werden, dass eine Variable u_k in ϕ vorkommt, wobei φ_k von Variablen u_i mit $i < k$ abhängt. Wenn eine solche Variable nach Algorithmus 3.39 durch eine Fixpunktformel $\sigma_k u_k.\varphi_k$ ersetzt wird, treten in ϕ Variablen u_i mit $i < k$ auf. Da die entsprechenden Gleichungen von diesem Algorithmus nicht bearbeitet werden (sie stehen weiter unten), bleiben die Variablen u_i fälschlicherweise frei.

Beispiel 3.44 Sei E wie im vorangegangenen Beispiel und $\phi = x_2 \wedge u_2$. In diesem Fall ergibt die Anwendung von Algorithmus 3.39 auf den Block

$$\begin{cases} u_3 \overset{\nu}{=} \Box u_3 \wedge x_3 \vee u_1 \\ u_2 \overset{\mu}{=} \Diamond u_2 \vee x_2 \wedge u_3 \end{cases}$$

$\phi = x_2 \wedge \mu u_2.\Diamond u_2 \vee x_2 \wedge \nu u_3 \Box u_3 \wedge x_3 \vee u_1$. Dies ergibt keinen Sinn, weil u_1 noch frei ist. Die Umkehrung muss also für die Variable u_1 erneut angestoßen werden, so dass diese durch den entsprechenden Fixpunkt $\mu u_1.\Diamond u_1 \vee x_1 \wedge ((\mu u_2.\Diamond u_2 \vee x_2 \wedge \nu u_3.\Box u_3 \wedge x_3 \vee u_1) \vee \nu u_3.\Box u_3 \wedge x_3 \vee u_1)$ ersetzt werden kann. □

Folgender Algorithmus ist eine Erweiterung von Algorithmus 3.39, die diese Problematik berücksichtigt und für jedes gültige Paar (E, ϕ) die entsprechende flache Formel Φ erzeugt:

Algorithmus 3.45 (Umwandlung eines Paares (E, ϕ) in eine μ-Kalkül-Formel) *Gegeben sei ein gültiges Paar (E, ϕ), bestehend aus einem Gleichungssystem*

$$E := \begin{cases} u_n & \overset{\sigma_n}{=} & \varphi_n(u_1, \ldots, u_n) \\ & \vdots & \\ u_1 & \overset{\sigma_1}{=} & \varphi_1(u_1, \ldots, u_n), \end{cases}$$

mit $\sigma_i \in \{\mu, \nu\}$, und einer flachen Formel $\phi \in \mathcal{L}_\mu$, in der die Variablen $u_1, \ldots, u_n$ frei sind. Die entsprechende Formel Φ im flachen μ-Kalkül wird wie folgt erstellt:

1. *Setze $\Phi := \phi$ und initialisiere $\mathsf{FV}((E, \Phi))$ mit der Menge der Variablen $\{u_1, \ldots, u_n\}$, die in Φ frei sind.*
2. *Wenn $\mathsf{FV}((E, \Phi)) = \emptyset$, gebe Φ zurück.*
3. *Für jedes $u_i \in \mathsf{FV}((E, \Phi))$:*
   (a) *Finde die Gleichung $u_i \overset{\sigma_i}{=} \varphi_i$ in E und setze $k = i + 1$.*
   (b) *Solange $k \leq n$*
      i. *Ersetze u_k in φ_i durch $\sigma_k u_k.\varphi_k$, ggf. in Klammern geschrieben, um die Priorität der Operatoren nicht zu beeinflussen.*
      ii. *Ersetze k durch $k + 1$ und gehe zurück zu Schritt b.*
   (c) *Ersetze u_i in Φ durch $\sigma_i u_i.\varphi_i$.*
4. *Aktualisiere $\mathsf{FV}((E, \Phi))$ und gehe zurück zu Schritt 2.*

Beispiel 3.46 Gegeben sei das Paar (E, ϕ) mit

$$E := \begin{cases} u_3 & \overset{\nu}{=} & \Box u_3 \wedge x_3 \vee u_1 \\ u_2 & \overset{\mu}{=} & \Diamond u_2 \vee x_2 \wedge u_3 \\ u_1 & \overset{\mu}{=} & \Diamond u_1 \vee x_1 \wedge (u_2 \vee u_3) \end{cases}$$

und $\phi = x_2 \wedge u_2$. Dann durchläuft Algorithmus 3.45 folgende Schritte:

1. Es gelten $\Phi = x_2 \wedge u_2$ und $\mathsf{FV}((E, \Phi)) = \{u_2\}$. Die Anzahl der Gleichungen ist $n = 3$.
2. $\mathsf{FV}((E, \Phi)) \neq \emptyset$, also folgt:
3. Für u_2:
   (a) Die gesuchte Gleichung $u_i \overset{\sigma_i}{=} \varphi_i$ ist $u_2 \overset{\mu}{=} \Diamond u_2 \vee x_2 \wedge u_3$. Es gelten $i = 2$ und $k = 3$.
   (b) Für $k = 3$:
      i. Die Variable u_3 wird in φ_2 durch $\nu u_3.\Box u_3 \wedge x_3 \vee u_1$ ersetzt. Es ergibt sich $\varphi_2 = \Diamond u_2 \vee x_2 \wedge \nu u_3.\Box u_3 \wedge x_3 \vee u_1$.
      ii. $k = 4$.
   (c) In Φ wird u_2 durch $\mu u_2.\varphi_2$ ersetzt. Es gilt $\Phi = x_1 \wedge \mu u_2.\Diamond u_2 \vee x_2 \wedge \nu u_3.\Box u_3 \wedge x_3 \vee u_1$.
4. Es wird mit $\mathsf{FV}((E, \Phi)) = \{u_1\}$ zu Schritt 2 zurückgesprungen.
2. $\mathsf{FV}((E, \Phi)) \neq \emptyset$, also folgt:
3. Für u_1:
   (a) Die gesuchte Gleichung $u_i \overset{\sigma_i}{=} \varphi_i$ ist $u_1 \overset{\mu}{=} \Diamond u_1 \vee x_1 \wedge (u_2 \vee u_3)$. Es gelten $i = 1$ und $k = 2$.
   (b) Für $k = 2$:

i. Die Variable u_2 wird in φ_1 durch $\mu u_2.\Diamond u_2 \vee x_2 \wedge u_3$ ersetzt. Es ergibt sich $\varphi_1 = \Diamond u \vee x_1 \wedge ((\mu u_2.\Diamond u_2 \vee x_2 \wedge u_3) \vee u_3)$.

ii. $k = 3$.

(b) Für $k = 3$:

i. Die Variable u_3 wird in φ_1 durch $\nu u_3.\Box u_3 \wedge x_3 \vee u_1$ ersetzt. Es ergibt sich $\varphi_1 = \Diamond u \vee x_1 \wedge ((\mu u_2.\Diamond u_2 \vee x_2 \wedge \nu u_3.\Box u_3 \wedge x_3 \vee u_1) \vee \nu u_3.\Box u_3 \wedge x_3 \vee u_1)$.

ii. $k = 4$.

(c) In Φ wird u_1 durch $\mu u_1.\varphi_1$ ersetzt. Es gilt $\Phi = x_2 \wedge \mu u_2.\Diamond u_2 \vee x_2 \wedge \nu u_3.\Box u_3 \wedge x_3 \vee \mu u_1.\Diamond u \vee x_1 \wedge ((\mu u_2.\Diamond u_2 \vee x_2 \wedge \nu u_3.\Box u_3 \wedge x_3 \vee u_1) \vee \nu u_3.\Box u_3 \wedge x_3 \vee u_1)$.

4. Es wird mit $\mathsf{FV}((E, \Phi)) = \emptyset$ zu Schritt 2 zurückgesprungen.

2. Es wird Φ zurückgegeben. □

An diesem Beispiel wird auch klar, dass Gleichungssysteme um vieles kompakter als die entsprechenden flachen Formeln sein können.

Syntax und Semantik des vektoriellen μ-Kalküls lassen sich nun wie folgt definieren:

Definition 3.47 (Syntax des vektoriellen μ-Kalküls) *Sei $\mathcal{E}_\mu$ die Menge aller Paare (E, ϕ) in denen E ein μ-Kalkül-Gleichungssystem ist und $\phi \in \mathcal{L}_\mu$ eine flache μ-Kalkül-Formel, in der die Variablen aus den linken Seiten der Gleichungen in E frei sind. Dann ist $\vec{\mathcal{L}}_\mu := \mathcal{L}_\mu \cup \mathcal{E}_\mu$ die Menge der vektoriellen μ-Kalkül-Ausdrücke.*

Definition 3.47 ist lediglich eine Erweiterung von Definition 3.17 um gültige Paare der Form (E, ϕ). Auch die Semantik des vektoriellen μ-Kalküls wird auf die des flachen aus Definition 3.18 zurückgeführt:

Definition 3.48 (Semantik des vektoriellen μ-Kalküls) *Gegeben seien eine Kripke-Struktur $\mathcal{K}$ und ein Paar $(E, \phi) \in \vec{\mathcal{L}}_\mu$. Dann ist $[\![(E, \phi)]\!]_\mathcal{K} := [\![\Phi]\!]_\mathcal{K}$, wobei Φ die Formel ist, die aus der Anwendung von Algorithmus 3.45 auf (E, ϕ) entsteht.*

3.3.2 Abhängigkeitsgraphen von Gleichungssystemen

Die Definitionen 3.47 und 3.48 erlauben es, Zustandsmengen einer gegebenen Kripke-Struktur $\mathcal{K}$ in der Form (E, ϕ) zu beschreiben. Prinzipiell lässt sich die Zustandsmenge $[\![(E, \phi)]\!]_\mathcal{K}$ nach Definition 3.48 ermitteln, indem (E, ϕ) mit Hilfe von Algorithmus 3.45 in eine flache Formel Φ umgewandelt wird, womit dann $[\![\Phi]\!]_\mathcal{K}$ nach Definition 3.18 berechnet werden kann. Wie aber in Abschnitt 3.2.5 bereits erläutert, ist mit der Einführung des vektoriellen μ-Kalküls eine Steigerung der Effizienz beabsichtigt, die erst durch alternative Berechnungsmöglichkeiten erlangt wird.

Die Formel ϕ aus dem Paar (E, ϕ) enthält Variablen aus der linken Seite des Gleichungssystems E. Das Ziel des vektoriellen μ-Kalküls ist es, Ausdrücke für diese Variablen anhand des Gleichungssystems E zu berechnen, was als die *Lösung* des Gleichungssystems bezeichnet wird. Werden diese Ergebnisse in ϕ substituiert, zerfällt diese in eine flache Formel ϕ', die keine Fixpunktformeln enthält. Die Berechnung von $[\![(E, \phi)]\!]_\mathcal{K}$ wird also nach der Substitution auf die von $[\![\phi']\!]_\mathcal{K}$ vereinfacht. Somit besteht die eigentliche Aufgabe in der Lösung des Gleichungssystems, die im Folgenden genauer untersucht wird. Um die Semantik zu erhalten, halten sich die Überlegungen stets an die Auswertung der entsprechenden flachen Formel.

Die Umwandlung eines Paares (E, ϕ) in eine flache Formel Φ erzeugt im Allgemeinen Teilformeln der Form $\sigma u.\varphi$. Im einfachsten Fall können diese unabhängig voneinander berechnet werden, wie z.B. die Ausdrücke für μu_1 und νu_4 in der Formel Φ aus Beispiel 3.42. Aus der Sicht des Gleichungssystems entsprechen solche Teilformeln Gleichungen, zwischen denen keinerlei Abhängigkeit besteht. Gleichzeitig kann jede dieser Fixpunktformeln verschachtelte Fixpunktformeln enthalten. Solche Formeln stammen aus Gleichungen, zwischen denen Abhängigkeiten bestehen und die im Allgemeinen zu verschachtelten Iterationen führen. Diese Zusammenhänge werden durch folgende graphische Darstellung verdeutlicht:

Definition 3.49 (Abhängigkeitsgraph eines Gleichungssystems) *Sei E das Gleichungssystem*

$$\begin{cases} u_n & \overset{\sigma_n}{=} & \varphi_n(u_1, \ldots, u_n) \\ & \vdots & \\ u_1 & \overset{\sigma_1}{=} & \varphi_1(u_1, \ldots, u_n) \end{cases}$$

und $\mathsf{FV}(\varphi_k)$ *die Menge der freien Variablen in* φ_k*. Dann ist* (V, T) *der Abhängigkeitsgraph von E, wobei* $V := \{u_1, \ldots, u_n\}$ *und* $T := \{(u_i, u_k) \mid u_i \in \mathsf{FV}(\varphi_k)\}$.

Eine Kante von Knoten u_i nach Knoten u_k bedeutet demzufolge „u_i beeinflusst den Wert von u_k“.

Beispiel 3.50 Der Abhängigkeitsgraph für das Gleichungssystem aus Beispiel 3.42 ist in Abbildung 3.2 wiedergegeben. Hieraus wird ersichtlich, dass Knoten des Graphen, die nicht miteinander verbunden sind, stets zu

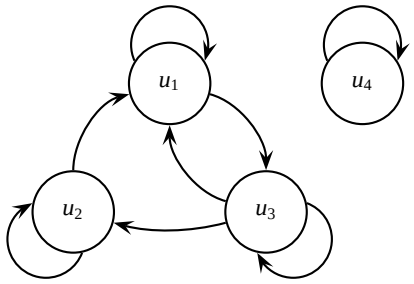


Abbildung 3.2: Der Abhängigkeitsgraph des Gleichungssystems aus Beispiel 3.42

Teilformeln führen, die unabhängig voneinander berechnet werden können. □

In Beispiel 3.50 sind die Abhängigkeiten entweder wechselseitig, wie zwischen u_1, u_2 und u_3, oder gar nicht vorhanden, wie zwischen letzteren und u_4. Das folgende Beispiel zeigt, dass es auch unidirektionale Abhängigkeiten geben kann.

Beispiel 3.51 Sei das Gleichungssystem

$$E := \begin{cases} u_6 & \overset{\mu}{=} & \Diamond u_6 \vee u_4 \\ u_5 & \overset{\nu}{=} & \Box u_5 \wedge x_5 \\ u_4 & \overset{\nu}{=} & \Box(u_4 \vee u_1) \wedge (u_6 \vee u_5) \\ u_3 & \overset{\nu}{=} & \Box u_3 \wedge x_3 \vee u_1 \\ u_2 & \overset{\mu}{=} & \Diamond u_2 \vee x_2 \wedge u_3 \\ u_1 & \overset{\mu}{=} & \Diamond u_1 \vee x_1 \wedge (u_2 \vee u_3). \end{cases}$$

Der entsprechende Abhängigkeitsgraph ist in Abbildung 3.3 zu sehen. □

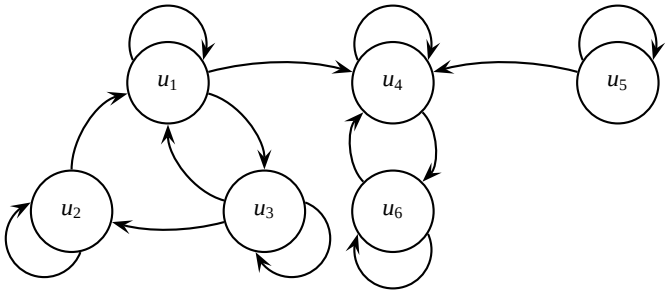


Abbildung 3.3: Der Abhängigkeitsgraph des Gleichungssystems aus Beispiel 3.51

Anhand des Abhängigkeitsgraphen eines Gleichungssystems können folgende allgemein gültige Überlegungen gemacht werden:

- Aus Algorithmus 3.45 geht hervor, dass jede Variable u_i in der Formel ϕ durch eine Fixpunktformel ersetzt wird, in der alle Variablen verschachtelt sind, von deren Knoten es einen Pfad zu dem Knoten u_i im Abhängigkeitsgraphen gibt. Zum Beispiel: Falls zu dem Gleichungssystem in Beispiel 3.51 eine Formel ϕ angegeben wird, die die Variable u_6 enthält, dann führt dies zu einer verschachtelten Teilformel, in der alle Variablen vorkommen. Hingegen würde die Variable u_1 zu einer Teilformel führen, in der nur die Variablen u_1, u_2 und u_3 verschachtelt sind.

- Wenn eine Variable u_i in φ_k vorkommt, gibt es eine Kante von u_i zu u_k. Wenn eine Variable u_i eine Variable u_k indirekt über eine dritte Variable u_j beeinflusst, so gibt es einen Pfad von u_i über u_j nach u_k. Somit bilden Variablen, die *alle* mittelbar oder unmittelbar voneinander abhängen, streng zusammenhängende Komponenten[4]. In Abbildung 3.3 bilden die Knoten $\{u_1, u_2, u_3\}$, $\{u_4, u_6\}$ und $\{u_5\}$ jeweils eine maximale streng zusammenhängende Komponente. Diese werden fortan auch *maximale Komponenten* genannt.

- Bei der Berechnung beeinflussen sich die Gleichungen, die den maximalen Komponenten entsprechen, möglicherweise in eine Richtung, aber niemals gegenseitig. Zum Beispiel hängt in Abbildung 3.3 die Berechnung der Gleichungen für $\{u_4, u_6\}$ von denen für $\{u_1, u_2, u_3\}$ ab, aber nicht umgekehrt. Dies lässt sich mit Hilfe des Abhängigkeitsgraphen visualisieren, indem die maximalen Komponenten zu Superknoten zusammengefasst werden. Dadurch entsteht ein vereinfachter Graph, der nur aus trivialen streng zusammenhängenden Komponenten besteht. Für den Graph aus Abbildung 3.3 ist der vereinfachte Abhängigkeitsgraph der in Abbildung 3.4.

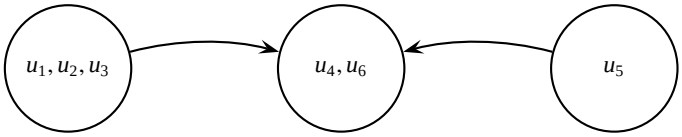


Abbildung 3.4: Der vereinfachte Abhängigkeitsgraph für Beispiel 3.51

- Da die Gleichungen aus zwei verschiedenen Superknoten keine wechselseitigen Abhängigkeiten aufweisen, genügt es, die Berechnung der Fixpunkte bei den Superknoten zu beginnen, die keine ankommende Kante besitzen und die Ergebnisse dann entsprechend der Kanten im Graphen fortzupflanzen. Im Falle der Abbildung 3.4 müssen $\{u_1, u_2, u_3\}$ und $\{u_5\}$, egal in welcher Reihenfolge, vor $\{u_4, u_6\}$ berechnet werden.

- Die Lösung eines Gleichungssystems zerfällt damit in die Berechnung der Fixpunkte der maximalen Komponenten. Die Gleichungen aus einer maximalen Komponente ergeben, wenn mit Algorithmus 3.45 in den flachen μ-Kalkül umgewandelt, eine verschachtelte Formel, bei der sich die Reihenfolge, in der die Fixpunkte ausgewertet werden, aus der Verschachtelungstiefe ergibt. Folglich darf die Reihenfolge der Gleichungen, die eine maximalen Komponente im Abhängigkeitsgraphen bilden, zumindest im Allgemeinen nicht verändert werden. Eine Gleichung darf sich im Gleichungssystem jedoch auf und ab bewegen, solange sie dabei an keiner Gleichung der eigenen maximalen Komponente vorbeizieht. Damit können die Gleichungen so sortiert werden, dass aus jeder maximalen Komponente ein Block Gleichungen entsteht, die hintereinander geschrieben werden. In Beispiel 3.51 können z.B. die Gleichungen für u_5 und u_6 vertauscht werden, so dass die Gleichungen für $\{u_1, u_2, u_3\}$, $\{u_4, u_6\}$ und $\{u_5\}$ jeweils einen solchen Block bilden. Die Blöcke dürfen beliebig miteinander vertauscht werden, ohne die Lösung des Gleichungssystems zu beeinflussen.

Diese Überlegungen werden in der folgenden Definition und dem folgenden Lemma festgehalten.

Definition 3.52 (Maximale Komponente eines Gleichungssystems) *Eine maximale Komponente eines Gleichungssystems E ist ein Gleichungssystem E', dessen Gleichungen in derselben Reihenfolge wie in E stehen und die in dem Abhängigkeitsgraphen von E eine maximale streng zusammenhängende Komponente bilden.*

[4] In einem gerichteten Graphen ist eine streng zusammenhängende Komponente ein Teilgraph, in dem es von jedem Knoten einen Pfad zu allen anderen Knoten gibt. Eine streng zusammenhängende Komponente ist maximal, wenn sie kein echter Teilgraph einer anderen solchen Komponente ist.

Lemma 3.53 (Zerlegung eines Gleichungssystems in maximale Komponenten) *Die Fixpunkte der maximalen Komponenten eines Gleichungssystems E lassen sich getrennt berechnen, so dass die Lösung von E auf die Lösung seiner einzelnen maximalen Komponenten zurückgeführt werden kann.*

Aus der obigen Argumentation folgt, dass die Umwandlung einer flachen Formel Φ in ein Paar (E, ϕ) folgende Vorteile bringt:

- Teilformeln von Φ, die nicht gegenseitig voneinander abhängen, zerfallen in getrennte Berechnungen. Bei einem System mit n Gleichungen sinkt somit die obere Grenze für die Anzahl der Iterationen von $|\mathcal{S}|^n$ auf $|\mathcal{S}|^m$, wobei m die Anzahl der Gleichungen der größten maximalen Komponente ist[5]. Damit werden unnötige Verschachtelungen der Iterationen, wie im Beispiel 3.34 auf Seite 43, per Konstruktion vermieden.
- Wenn eine Fixpunktformel mehrmals in Φ vorkommt, wird jede Instanz davon bei der Berechnung nach dem flachen μ-Kalkül (Definition 3.18) neu ermittelt. In dem Gleichungssystem entsteht für eine solche Teilformel nur *eine* Gleichung, wodurch unnötige Wiederholungen vermieden werden. Eine solche Situation kommt in Beispiel 3.51 nicht vor, würde aber entstehen, wenn es in Abbildung 3.4 eine zusätzliche Kante von $\{u_5\}$ nach $\{u_1, u_2, u_3\}$ gäbe. Die Gleichung für u_5 würde nach wie vor nur einmal berechnet, das Ergebnis aber zweimal verwendet werden.

Im nächsten Abschnitt werden weitere Verbesserungsmöglichkeiten erläutert, die sich aus der Auseinandersetzung mit der Berechnung der einzelnen maximalen Komponenten ergeben.

3.3.3 Lösung von Gleichungssystemen

Dieser Abschnitt enthält Algorithmen zur Lösung von Gleichungssystemen, deren Gleichungen eine maximale Komponente bilden. Die Lösung eines Gleichungssystems im allgemeinen Fall besteht dann laut Lemma 3.53 aus der Anwendung dieser Algorithmen auf die einzelnen maximalen Komponenten.

Aus Algorithmus 3.45 geht hervor, dass Gleichungssysteme, die aus einer maximalen Komponente bestehen, stets in verschachtelte Fixpunktformeln der Form $\sigma u.\varphi$ umgewandelt werden (in diesem Fall ist auch Algorithmus 3.39 anwendbar). Definition 3.18 schreibt vor, dass die Berechnung einer solchen Formel mit dem äußersten Fixpunkt beginnt, und dass jede Verschachtelungsebene eine neue Fixpunktiteration anstößt. Im Falle der Gleichungssysteme folgt aus deren Aufbauweise nach Algorithmus 3.41, dass die Berechnung mit der unteren Gleichung beginnt. Enthält die rechte Seite einer Gleichung eine Variable, für die weiter oben im Gleichungssystem eine Gleichung steht (was einer tieferen Verschachtelungsebene in der Formel entspricht), so muss dieser obere Fixpunkt zunächst bestimmt werden, bevor mit der unteren Gleichung weitergerechnet werden kann. Folgender Algorithmus fasst diese Überlegungen zusammen:

Algorithmus 3.54 (1. Lösung eines Gleichungssystems aus einer maximalen Komponente) *Sei*

$$E := \begin{cases} u_n & \overset{\sigma_n}{=} & \varphi_n(u_1, \ldots, u_n) \\ & \vdots & \\ u_1 & \overset{\sigma_1}{=} & \varphi_1(u_1, \ldots, u_n) \end{cases}$$

ein Gleichungssystem, dessen Abhängigkeitsgraph aus einer maximalen streng zusammenhängenden Komponente besteht. Dann kann E mit den folgenden Formeln gelöst werden:

$$\begin{bmatrix} u_1^{i_1+1} & := & \varphi_1(u_1^{i_1}, \ldots, u_n^{i_1,\infty,\ldots,\infty}) \\ & \vdots & \\ u_n^{i_1,\ldots,i_n+1} & := & \varphi_n(u_1^{i_1}, \ldots, u_n^{i_1,\ldots,i_n}). \end{bmatrix} \tag{3.24}$$

Dabei werden die Anfangswerte $u_k^{i_1,\ldots,i_{k-1},0}$ bei $\sigma_k = \mu$ gleich 0 und bei $\sigma_k = \nu$ gleich 1 gesetzt.

[5] Dies ist ein vorläufiges Ergebnis, das in Abschnitt 3.3.3 noch verbessert wird.

Beispiel 3.55 Im Falle des Gleichungssystems

$$\begin{cases} u_2 & \overset{\sigma_2}{=} & \varphi_2(u_1, u_2) \\ u_1 & \overset{\sigma_1}{=} & \varphi_1(u_1, u_2) \end{cases}$$

ergeben sich wie in Beispiel 3.34 auf Seite 43 die Formeln

$$\begin{bmatrix} u_1^{i_1+1} & := & \varphi_1(u_1^{i_1}, u_2^{i_1,\infty}) \\ u_2^{i_1,i_2+1} & := & \varphi_2(u_1^{i_1}, u_2^{i_1,i_2}). \end{bmatrix}$$

□

Aus der Verschachtelung der Iterationen in 3.24 folgt, dass die Lösung eines Gleichungssystems mit n Gleichungen auf einer Kripke-Struktur mit $|\mathcal{S}|$ Zuständen bis zu $|\mathcal{S}|^n$ Iterationen betragen kann. Im Folgenden wird untersucht, wie sich diese obere Grenze reduzieren lässt. Ausgangspunkt für die Verbesserungen ist die Tatsache, dass das Tarski-Knaster-Theorem nicht unbedingt verlangt, dass die Iterationen jedes Mal mit dem Wert 0 bzw. 1 anfangen. Statt dessen kann jeder Wert, der die Bedingungen μ_1 und μ_2 bzw. ν_1 und ν_2 in Theorem 3.27 erfüllt als Anfangswert dienen. Folgendes Lemma liefert Prä- bzw. Post-Fixpunkte, die bessere Annäherungen zu den gesuchten Fixpunkten bilden und deshalb zu weniger Iterationsschritten führen.

Lemma 3.56 (Übertragung der Monotonie auf verschachtelte Fixpunkte) *Sei das Gleichungssystem*

$$E := \begin{cases} u_n & \overset{\sigma_n}{=} & \varphi_n(u_1, \ldots, u_n) \\ & \vdots & \\ u_1 & \overset{\sigma_1}{=} & \varphi_1(u_1, \ldots, u_n), \end{cases}$$

das aus einer maximalen Komponente besteht und in dem $\sigma_k \in \{\mu, \nu\}$ für $1 \leq k \leq n$. In den Formeln 3.24 gilt dann

$$\text{für } \sigma_k = \mu\text{:} \quad u_{k+1}^{i_1,\ldots,i_k-1,\infty} \sqsubseteq u_{k+1}^{i_1,\ldots,i_k,\infty}, \tag{3.25}$$

$$\text{für } \sigma_k = \nu\text{:} \quad u_{k+1}^{i_1,\ldots,i_k,\infty} \sqsubseteq u_{k+1}^{i_1,\ldots,i_k-1,\infty}. \tag{3.26}$$

Der Beweis dieses Lemmas befindet sich auf Seite 132 in Anhang B.

Lemma 3.56 ist wie folgt zu interpretieren: Wenn $\sigma_k = \sigma_{k+1} = \mu$, dann ist der Wert $u_{k+1}^{i_1,\ldots,i_k-1,\infty}$ ein Prä-Fixpunkt von $u_{k+1}^{i_1,\ldots,i_k,\infty}$ und erfüllt damit Bedingung μ_2 in Theorem 3.27. Es ist deshalb naheliegend, letzteren als Anfangswert für die Berechnung von $u_{k+1}^{i_1,\ldots,i_k,\infty}$ zu verwenden. Dazu muss auch Bedingung μ_1 erfüllt sein. Diese ergibt sich aus der Ungleichung 3.25 und aus der Wahl $u_k^{0,\ldots,0} := 0$ für die allererste Berechnung von u_k. Der Fall $\sigma_k = \sigma_{k+1} = \nu$ ist ähnlich: Der Wert $u_{k+1}^{i_1,\ldots,i_k-1,\infty}$ ist dann ein Post-Fixpunkt von $u_{k+1}^{i_1,\ldots,i_k,\infty}$ und erfüllt Bedingung ν_2 in Theorem 3.27. Bedingung ν_1 ergibt sich aus Ungleichung 3.26 und aus der Wahl $u_k^{0,\ldots,0} := 1$. Zusammenfassend gilt: Falls $\sigma_k = \sigma_{k+1}$, ist es legitim, für die Berechnung von $u_{k+1}^{i_1,\ldots,i_k,\infty}$ für $i_k > 1$ den Anfangswert

$$u_{k+1}^{i_1,\ldots,i_k,0} := u_{k+1}^{i_1,\ldots,i_k-1,\infty}$$

zu wählen. Der Wert auf der rechten Seite ist aber gerade der, bei dem die Iteration für u_{k+1} zuletzt stehen blieb, so dass sich diese Wahl der Anfangswerte automatisch ergibt, wenn die berechneten Werte in ihren Speicherplätzen nicht verändert werden.

Diese Optimierung greift nicht, wenn $\sigma_k \neq \sigma_{k+1}$, denn in diesem Fall steht ein Prä-Fixpunkt zur Verfügung, wenn ein Post-Fixpunkt benötigt wird, oder umgekehrt. Bei einem Wechsel der Fixpunktart wird deshalb wieder auf die immer gültigen Werte 0 (für $\sigma_{k+1} = \mu$) bzw. 1 (für $\sigma_{k+1} = \nu$) zurückgegriffen. Diese Erkenntnisse werden im folgenden Algorithmus zusammengefasst:

Algorithmus 3.57 (2. Lösung von Gleichungssystemen aus einer maximalen Komponente) *Sei*

$$E := \begin{cases} u_n & \overset{\sigma_n}{=} & \varphi_n(u_1, \ldots, u_n) \\ & \vdots & \\ u_1 & \overset{\sigma_1}{=} & \varphi_1(u_1, \ldots, u_n) \end{cases}$$

ein Gleichungssystem, dessen Abhängigkeitsgraph aus einer maximalen streng zusammenhängenden Komponente besteht. Dann kann E mit den folgenden Formeln gelöst werden:

$$\left[\begin{array}{lcl} u_1^{i_1+1} & := & \varphi_1(u_1^{i_1}, \ldots, u_n^{i_1,\infty,\ldots,\infty}) \\ & \vdots & \\ u_n^{i_1,\ldots,i_n+1} & := & \varphi_n(u_1^{i_1}, \ldots, u_n^{i_1,\ldots,i_n}). \end{array}\right. \tag{3.27}$$

Die Anfangswerte $u_k^{0,\ldots,0}$ *für die erste Berechnung von* u_k *werden bei* $\sigma_k = \mu$ *gleich* 0 *und bei* $\sigma_k = \nu$ *gleich* 1 *gesetzt. Danach werden die Anfangswerte* $u_{k+1}^{i_1,\ldots,i_k,0}$ *dann und nur dann auf* 0 *bzw.* 1 *zurückgesetzt, wenn* $\sigma_k \neq \sigma_{k+1}$.

Dieses Prinzip wurde zum ersten Mal von Emerson und Lei in [32] eingesetzt, um die Berechnung flacher μ-Kalkül-Formeln zu verbessern. Unterschiede zwischen Algorithmus 3.57 und den dort vorgestellten Ergebnissen ergeben sich lediglich in der Behandlung von Teilformeln der Form $\sigma u.\varphi$, die im Falle der Gleichungssysteme entfallen. Im Folgenden wird gezeigt, dass Algorithmus 3.57 die gleiche Komplexität wie der von Emerson und Lei hat. Dazu sind folgende Definitionen notwendig:

Definition 3.58 (Maximaler Teilblock) *Ein maximaler Teilblock eines Gleichungssystems, das aus einer maximalen Komponente besteht, ist eine Gruppe aufeinander folgender Gleichungen, die alle denselben Fixpunkttyp haben und die entweder an keine Gleichung oder aber an Gleichungen des entgegengesetzten Fixpunkttyps grenzen.*

Im Folgenden werden maximale Teilblöcke nur noch Teilblöcke genannt. Abbildung 3.5 zeigt ein Gleichungssystem mit ℓ Teilblöcken.

$$\left\{\begin{array}{lcll} u_n & \overset{\sigma_l}{=} & \varphi_n(u_1, \ldots, u_n) & \\ & \vdots & & \textit{Teilblock } B_\ell \\ u_{m+1} & \overset{\sigma_l}{=} & \varphi_{k+1}(u_1, \ldots, u_n) & \\ u_m & \overset{\sigma_{l-1}}{=} & \varphi_k(u_1, \ldots, u_n) & \\ & \vdots & & \textit{Teilblock } B_{\ell-1} \\ u_{j+1} & \overset{\sigma_{l-1}}{=} & \varphi_{j+1}(u_1, \ldots, u_n) & \\ & \vdots & & \vdots \\ u_a & \overset{\sigma_1}{=} & \varphi_i(u_1, \ldots, u_n) & \\ & \vdots & & \textit{Teilblock } B_1 \\ u_1 & \overset{\sigma_1}{=} & \varphi_1(u_1, \ldots, u_n) & \end{array}\right.$$

Abbildung 3.5: Die maximalen Teilblöcke eines Gleichungssystems

Definition 3.59 (Alternierungstiefe eines Gleichungssystems) *Die Alternierungstiefe eines Gleichungssystems, das aus einer maximalen Komponente besteht, ist die Anzahl seiner maximalen Teilblöcke*[6].

Die Alternierungstiefe eines Gleichungssystems aus mehreren maximalen Komponenten ist das Maximum der Alternierungstiefen seiner maximalen Komponenten.

Gleichungssysteme, in denen alle Gleichungen denselben Fixpunkttyp haben, bestehen aus einem einzigen Teilblock, haben Alternierungstiefe 1 und werden alternierungsfrei genannt.

[6]Die hier gegebene Definition entspricht der von Seidl [76] und ist für die Zwecke dieser Arbeit ausreichend. Eine feinere Version von Niwiński [65] führt unter besonderen Umständen zu einem Wert, der um eins kleiner ist. Diese Optimierung ist auch hier möglich.

Definition 3.60 (Größe eines Gleichungssystems) *Die Größe einer Gleichung eines Gleichungssystems ist die Anzahl der Operatoren und Variablen auf ihrer rechten Seite. Die Größe eines Gleichungssystems E wird mit $|E|$ bezeichnet und ergibt sich aus der Summe der Größen der einzelnen Gleichungen.*

Theorem 3.61 (Komplexität der μ-Kalkül-Modellprüfung nach Algorithmus 3.57) *Gegeben seien eine Kripke-Struktur $\mathcal{K} = \langle \mathcal{S}, \mathcal{I}, \mathcal{R}, \mathcal{L} \rangle$ und ein Gleichungssystem E der Alternierungstiefe ℓ und der Größe $|E|$ über den Variablen von $\mathcal{K}$. Die Lösung von E wird von Algorithmus 3.57 innerhalb der Zeit $O\left(\left(\frac{|E||\mathcal{S}|}{\ell}\right)^{\ell} |\mathcal{R}|\,|E|\right)$ berechnet.*

Der Beweis dieses Theorems befindet sich auf Seite 133 in Anhang B.

Diese obere Grenze für die Berechnungsdauer von Algorithmus 3.57 entspricht den Optimierungen von Emerson und Lei in [32] und stellt eine wesentliche Verbesserung von Algorithmus 3.54 dar. Zum Beispiel wächst bei diesem auch bei Gleichungssystemen der Alternierungstiefe 1 die Anzahl der Iterationen exponentiell mit der Größe des Gleichungssystems, während dieses Wachstum bei Algorithmus 3.57 nur noch linear ist.

Eine noch geringere Komplexität wurde von Cleaveland et al. [21] durch Buchführung von Zwischenresultaten erzielt, so dass unnötige Iterationen erspart bleiben. Das Ergebnis ist in folgendem Theorem festgehalten:

Theorem 3.62 (Komplexität der μ-Kalkül-Modellprüfung [21]) *Gegeben seien eine Kripke-Struktur $\mathcal{K} = \langle \mathcal{S}, \mathcal{I}, \mathcal{R}, \mathcal{L} \rangle$ und ein Gleichungssystem E der Alternierungstiefe ℓ und der Größe $|E|$ über den Variablen von $\mathcal{K}$. Die Lösung von E kann innerhalb der Zeit $O\left(\left(\frac{|E||\mathcal{S}|}{\ell}\right)^{\ell-1} |\mathcal{R}|\,|E|\right)$ berechnet werden.*

Um diese Verringerung der Komplexität zu erreichen, verwendet der Algorithmus von Cleaveland et al. einen Vektor der Länge $|\mathcal{S}|$, dessen Elemente einzeln adressiert werden. Deshalb ist selbst bei einer symbolischen Darstellung der Transitionsrelation diese Lösung auf Probleme mit moderaten Zustandsräumen und großen Alternierungstiefen beschränkt. In der Modellprüfung wird aber gerade das Gegenteil benötigt, so dass für die Praxis eine symbolische Implementierung von Algorithmus 3.57 immer noch die bessere Wahl ist. Dies gilt selbst wenn durch die symbolische Darstellung z.B. mit binären Entscheidungsdiagrammen die Berechnungen auf der Transitionsrelation nicht mehr in linearer Zeit durchzuführen sind.

Des Weiteren gibt es noch eine Verbesserung von Browne, Long et al. [10, 58] für Gleichungssysteme mit Alternierungstiefe $\ell \geq 3$. Die Übertragung der Monotonie wird auf alle ungeraden Teilblöcke erstreckt, so dass nur noch gerade Teilblöcke auf 0 bzw. 1 zurückgesetzt werden müssen. Es werden Hilfsstrukturen in der Größenordnung des Zustandsraumes angelegt, die eine explizite Adressierung verlangen. Damit ist auch dieser Algorithmus nicht für den praktischen Einsatz in der Modellprüfung geeignet. Unabhängig davon ist das Ergebnis wichtig für die Abschätzung der Komplexität des Problems. Unter den gleichen Voraussetzungen für die Berechnungen auf der Transitionsrelation wie in Theorem 3.61 ergibt sich Folgendes:

Theorem 3.63 (Komplexität der μ-Kalkül-Modellprüfung [10])) *Gegeben seien eine Kripke-Struktur $\mathcal{K} = \langle \mathcal{S}, \mathcal{I}, \mathcal{R}, \mathcal{L} \rangle$ und ein Gleichungssystem E der Alternierungstiefe $\ell \geq 3$ und der Größe $|E|$ über den Variablen von $\mathcal{K}$. Dann kann die Lösung von E innerhalb der Zeit $O\left(\ell^2 \left(|\mathcal{S}| + 1\right)^{\lceil \ell/2 \rceil} |\mathcal{R}|\,|E|\right)$ berechnet werden.*

Das Ergebnis von Browne, Long et al. wurde von Seidl [76] und Jurdziński [44, 45] bestätigt und stellt die geringste bekannte Komplexitätsgrenze für das Modellprüfungsproblem dar. Es ist noch unklar, ob die Lösung eines Gleichungssystems im Allgemeinen in polynomieller Zeit möglich ist. Diese Vermutung liegt jedoch nahe, zumal das μ-Kalkül-Modellprüfungsproblem zur Zeit das einzig bekannte Problem ist, dass in der Komplexitätsklasse $\mathsf{NP} \cap \mathsf{co\text{-}NP}$ liegt, für dessen Lösung kein polynomieller Algorithmus gefunden wurde.

Es folgen noch zwei Theoreme für alternierungsfreie Gleichungssysteme. Das erste besagt, dass die Iterationen für die Lösung alternierungsfreier Gleichungssysteme auch vektoriell – also nicht verschachtelt – laufen können:

Theorem 3.64 (Vektorielle Lösung für alternierungsfreie Gleichungssysteme) *Sei*

$$E := \left\{ \begin{array}{lcl} u_n & \stackrel{\sigma}{=} & \varphi_n(u_1, \ldots, u_n) \\ & \vdots & \\ u_1 & \stackrel{\sigma}{=} & \varphi_1(u_1, \ldots, u_n) \end{array} \right.$$

ein Gleichungssystem mit Alternierungstiefe 1 über den Variablen der Kripke-Struktur $\mathcal{K} = \langle \mathcal{S}, \mathcal{I}, \mathcal{R}, \mathcal{L} \rangle$, dessen Abhängigkeitsgraph aus einer maximalen streng zusammenhängenden Komponente besteht und in dem $\sigma \in \{\mu, \nu\}$. Ausgehend von den Anfangswerten $v_k^0 := 0$ für $\sigma = \mu$ bzw. $v_k^0 := 1$ für $\sigma = \nu$, $1 \leq k \leq n$, berechnen die Formeln

$$\left[\begin{array}{lcl} v_1^{i+1} & := & \varphi_1(v_1^i, \ldots, v_{n-1}^i, v_n^i) \\ & \vdots & \\ v_n^{i+1} & := & \varphi_n(v_1^{i+1}, \ldots, v_{n-1}^{i+1}, v_n^i) \end{array} \right. \tag{3.28}$$

innerhalb der Zeit $O\left(|E|^2 \, |\mathcal{S}| \, |\mathcal{R}|\right)$ die gleichen Fixpunkte wie Algorithmus 3.57.

Der Beweis dieses Theorems befindet sich auf Seite 135 in Anhang B.

Folgendes Theorem besagt, dass die Lösung von Gleichungssystemen mit Alternierungstiefe 1 gegenüber Vertauschungen der Gleichungen invariant ist. Die Vertauschungen haben allerdings keinen weiteren Einfluss auf die Komplexität.

Theorem 3.65 (Vertauschbarkeit der Gleichungen gleichförmiger Gleichungssysteme) *Sei*

$$E := \left\{ \begin{array}{lcl} u_n & \stackrel{\sigma}{=} & \varphi_n(u_1, \ldots, u_n) \\ & \vdots & \\ u_1 & \stackrel{\sigma}{=} & \varphi_1(u_1, \ldots, u_n) \end{array} \right.$$

ein Gleichungssystem mit Alternierungstiefe 1, dessen Abhängigkeitsgraph aus einer maximalen streng zusammenhängenden Komponente besteht und in dem $\sigma \in \{\mu, \nu\}$. Dann können die Gleichungen von E beliebig miteinander vertauscht werden, ohne dessen Lösung zu beeinflussen.

Der Beweis dieses Theorems befindet sich auf Seite 136 in Anhang B. Hiermit sind nun alle Grundlagen eingebracht, die in den nächsten Kapiteln die Anwendung des μ-Kalküls auf die Überwachersynthese ermöglichen werden.

3.4 Verbesserungsmöglichkeiten

Die Algorithmen für die Modellprüfung gehen davon aus, dass ein Modell des zu verifizierenden Systems in Form einer Kripke-Struktur vorliegt. Es gibt also keinen Ansatz, der den Entwickler bei der Erstellung dieses Modells unterstützt. Der Schwierigkeitsgrad dieser Aufgabe ist jedoch nicht zu unterschätzen. Unter Umständen bedeutet dies, dass der Entwickler sehr viele Versuche machen muss, bis er zu einem fehlerfreien Modell kommt, oder im Extremfall nicht in der Lage ist, ein solches Modell zu erstellen.

In Abschnitt 2.2 wurde deutlich, dass die Überwachersynthese in vielen Fällen aus Spezifikationen, die nicht alle Anforderungen an das zu entwerfende System erfüllen, eine brauchbare Spezifikation gewinnen kann. Es ist deshalb naheliegend, die Überwachersynthese für die Erstellung der Modelle heranzuziehen. Ein solches Modell hat natürlich auch die Nachteile, die in Abschnitt 2.8 erwähnt wurden und die mit Hilfe der Modellprüfung zu beheben sind. In den nächsten Kapiteln wird deshalb untersucht, wie beide Methoden in diesem Sinne kombiniert werden können.

Teil II

Ein neuer Ansatz zur Überwachersynthese

Kapitel 4

Die Überwachersynthese im μ-Kalkül

In diesem Kapitel wird die Überwachersynthese, die bisher auf endlichen Automaten und auf den in Kapitel 2 vorgestellten Algorithmen basiert, in ein μ-Kalkül-Modellprüfungsproblem übersetzt. Die Vorgehensweise wird in Abbildung 4.1 verdeutlicht. Der untere Weg von dem Automaten $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ zu dem Überwacher $\mathcal{A}_\mathcal{S}$ stellt den herkömmlichen Syntheseprozess aus Kapitel 2 dar. Neu in dieser Arbeit ist die Alternative, die Synthese mit Hilfe der Modellprüfung in dem μ-Kalkül durchzuführen. Dazu müssen als erstes die Unterschiede bei der Modellierung überwunden werden, da die Modellprüfung nicht auf Automaten, sondern auf Kripke-Strukturen basiert. Zu diesem Zweck wird in Abschnitt 4.1 eine Transformation des Automaten $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ in eine Kripke-Struktur definiert, wie auf der linken Seite der Abbildung angedeutet ist. Für den Schritt auf der rechten Seite wird eine Abbildung einer Zustandsmenge der Kripke-Struktur auf den ursprünglichen Automaten definiert. Die eigentliche Synthese besteht aus dem oberen Schritt in der Abbildung. Um diesen zu realisieren, wird in Abschnitt 4.2 aus den Synthesealgorithmen in Kapitel 2 ein Gleichungssystem abgeleitet. Die Lösung dieses Gleichungssystems ist eine Zustandsmenge, deren Abbildung auf den Automaten $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ die Zustandsmenge des Überwachers ergibt, die ansonsten durch die Anwendung der Synthesealgorithmen berechnet wird.

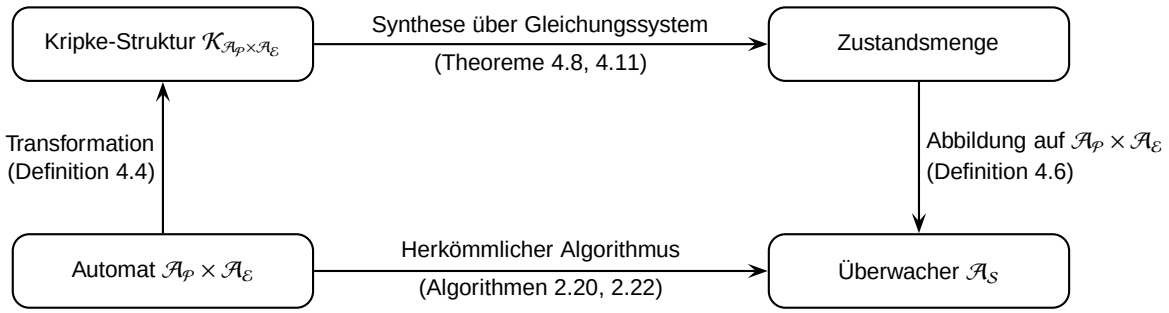


Abbildung 4.1: Die Übersetzung der Überwachersynthese in den μ-Kalkül

Dieses neue Verfahren wurde erstmals in [93] veröffentlicht. Seine Vorteile zeigen sich besonders in folgenden Punkten:

* *Bekämpfung der Zustandsexplosion:* Herkömmliche Methoden der Überwachersynthese benutzen eine explizite Darstellung der Transitionsrelation. Wie bereits in Abschnitt 2.8 erwähnt, sind deren Möglichkeiten durch das Problem der Zustandsexplosion auf Systeme beschränkt, die nicht die Größenordnung praktischer Anwendungen erreichen. Durch die Übersetzung eröffnen sich für die Überwachersynthese automatisch alle Möglichkeiten der symbolischen Darstellung, wie sie in der Modellprüfung üblich ist. Dadurch wird es möglich, bedeutend größere Systeme als bisher zu behandeln, wie durch das Beispiel in Abschnitt 5.5 belegt wird.
* *Überprüfbarkeit des Syntheseergebnisses:* Die durch die Übersetzung gewonnene Kripke-Struktur eignet sich sowohl für die Überwachersynthese als auch für die Modellprüfung. Wie ebenfalls in Abschnitt 2.8

angesprochen, bietet das Ramadge-Wonham-Modell keine Möglichkeit, das Ergebnis der Synthese auf gewünschte Eigenschaften zu überprüfen. Mit der hier vorgestellten Struktur können zum ersten Mal beide Verfahren auf dasselbe Modell angewendet werden. Dies bedeutet, dass ein System, das mit Hilfe der Überwachersynthese entworfen wird, ohne Zwischentransformationen auf gewünschte Eigenschaften überprüft werden kann.

* *Vereinfachung der Synthesewerkzeuge:* Durch die Formulierung der Synthesealgorithmen im μ-Kalkül bleibt dem Programmierer eines Synthesewerkzeugs das Erlernen der Überwachersynthese erspart. Die Aufgabe vereinfacht sich im Wesentlichen auf das Einlesen der Modelle und auf die Lösung eines μ-Kalkül-Gleichungssystems z.B. nach Algorithmus 3.57. Ein flexibles Programm sollte allerdings in der Lage sein, verschiedene Gleichungssysteme zu behandeln, was den unterschiedlichen Verbesserungen, die in den folgenden Kapiteln noch vorgestellt werden, entgegen kommt.

* *Integration der Synthese in vorhandene Modellprüfer:* Die Darstellung der Überwachersynthese als Modellprüfungsproblem ermöglicht es, bereits existierende Modellprüfer für die Synthese zu verwenden. Eventuelle Anpassungen beschränken sich auf die Ein- und Ausgabe der Daten sowie auf die Erweiterung um eine einfache Zustandstransformationsfunktion, die leicht zu implementieren ist.

* *Weiterentwicklung der Synthesealgorithmen:* Die Darstellung der Überwachersynthese mit Gleichungssystemen an Stelle der Algorithmen aus Kapitel 2 ermöglicht nicht nur die Zusammenführung von Überwachersynthese und Modellprüfung, sondern auch eine Weiterentwicklung der Synthesealgorithmen. Der Grund dafür ist, dass es leichter ist, nach den Regeln der booleschen Algebra Änderungen an einem Gleichungssystem vorzunehmen, als einen in natürlicher Sprache – z.B. Deutsch oder Englisch – verfassten Algorithmus entsprechend anzupassen. Die Gleichungssysteme, die in diesem Kapitel aus den Synthesealgorithmen entstehen, dienen als Ausgangspunkt für Verbesserungen, die in den Kapiteln 5 und 6 vorgestellt werden. Auch die Komplexität der neuen Lösungen lässt sich anhand der Theoreme in Abschnitt 3.3.3 im Gegensatz zu der Komplexität eines Algorithmus leicht analysieren.

4.1 Eine Kripke-Struktur für Ramadge und Wonham

Wie in Kapitel 2 beschrieben, beginnt der Syntheseprozess mit zwei Automaten, einer Systembeschreibung $\mathcal{A}_\mathcal{P}$ und einer Spezifikation $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ für das gewünschte Verhalten des überwachten Systems. Der Automat $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ ist auch der Startpunkt für die Übersetzung des Syntheseproblems in den μ-Kalkül. Für die Transformation auf der linken Seite von Abbildung 4.1 werden die Zustände von $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ mit booleschen Variablen beschriftet, und zwar in einer Form, die es später ermöglicht, den Überwacher zu berechnen. Für die Beschriftung ist die Kenntnis der verbotenen Zustände von $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ nötig (s. Abschnitt 2.4.1).

Definition 4.1 (Beschriftung von $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$) *Gegeben sei ein Automat $\mathcal{A} := \mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ mit Zustandsmenge $Q_\mathcal{A}$. Seien $\mathcal{V}^x := \{x_{k-1}, \ldots, x_0\}$ und $\mathcal{V}^y := \{y_{k-1}, \ldots, y_0\}$ zwei Mengen boolescher Variablen, wobei $k \geq \lceil log_2(|Q_\mathcal{A}|) \rceil$. Seien ferner $\lambda^x : Q_\mathcal{A} \to 2^{\mathcal{V}^x}$ und $\lambda^y : Q_\mathcal{A} \to 2^{\mathcal{V}^y}$ eineindeutige Beschriftungen der Zustände von $Q_\mathcal{A}$ mit den Variablen in $\mathcal{V}^x$ bzw. $\mathcal{V}^y$. Dann wird der Zustand $q \in Q_\mathcal{A}$ mit den Variablen $\mathcal{V}_\mathcal{A} := \mathcal{V}^x \cup \mathcal{V}^y \cup \{x_m, x_b\}$ von der Funktion $\mathcal{L}_\mathcal{A} : Q_\mathcal{A} \to 2^{\mathcal{V}_\mathcal{A}}$ folgendermaßen beschriftet:*

$$\mathcal{L}_\mathcal{A}(q) := \lambda^x(q) \cup \lambda^y(q) \cup \begin{cases} \{x_m\} & \text{falls } q \in M \\ \{\overline{x}_m\} & \text{sonst} \end{cases} \cup \begin{cases} \{x_b\} & \text{falls q Bedingung 2.1 nicht erfüllt} \\ \{\overline{x}_b\} & \text{sonst.} \end{cases}$$

Die Variablen in $\mathcal{V}^x$ bzw. $\mathcal{V}^y$ in Definition 4.1 verleihen jedem Zustand $q_i \in Q_\mathcal{A}$ eine eindeutige Kodierung, wie in den Abschnitten 3.1.1 und 3.1.2 für Kripke-Strukturen erläutert. Zusätzlich werden markierte Zustände mit x_m und verbotene Zustände mit x_b gekennzeichnet.

Beispiel 4.2 Zur Illustration dient erneut das Beispiel mit der Fertigungszelle aus Kapitel 2. Die Automaten $\mathcal{A}_\mathcal{P}$ und $\mathcal{A}_\mathcal{E}$ wurden in Beispiel 2.9 erstellt und sind in Abbildung 2.15 auf Seite 19 zu sehen. Der Automat $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ aus Abbildung 2.16 wird in Abbildung 4.2 wiedergegeben. Um die Schreibweise zu vereinfachen,

werden die Zustände neu nummeriert. Eindeutige Beschriftungen der Zustände mit den Variablen der Mengen $\mathcal{V}^x$ bzw. $\mathcal{V}^y$ ergeben sich am einfachsten aus der Darstellung der Zustandsnummern als Dualzahlen. Um dies zu verdeutlichen, werden die Variablen aus $\mathcal{V}^x$ auch in negierter Form dargestellt. Der besseren Lesbarkeit halber werden die Variablen aus $\mathcal{V}^y$ weggelassen und die Variablen x_m und x_b nur in positiver Form dargestellt. □

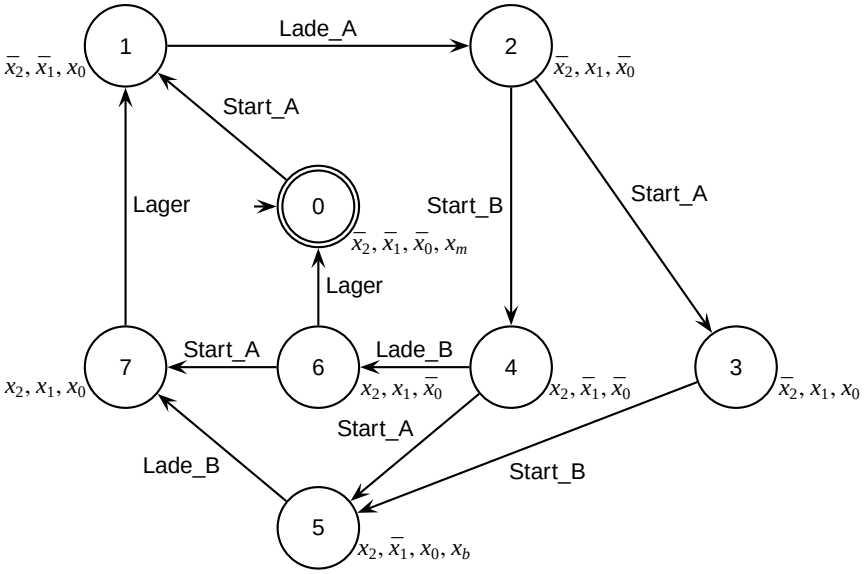


Abbildung 4.2: Der Automat $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ für Beispiel 4.2

In Anlehnung an Definition 3.5 wird die Zustandsmenge eines Automaten $\mathcal{A}$, die symbolisch durch den Ausdruck φ dargestellt wird, mit $[\![\varphi]\!]_\mathcal{A}$ bezeichnet.

Beispiel 4.3 Für den Automaten in Abbildung 4.2 gilt $[\![x_0]\!]_{\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}} = [\![y_0]\!]_{\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}} = \{1, 3, 5, 7\}$. □

Wie in Abschnitt 2.3.1 erwähnt, wird der Automat $\mathcal{A}_\mathcal{P}$ in der Regel durch die parallele Komposition kleinerer, überschaubarer Automaten erstellt. Dadurch kann die Zustandsmenge $Q_\mathcal{P}$ bereits so groß werden, dass eine explizite Darstellung von $\mathcal{A}_\mathcal{P}$ nicht mehr möglich ist. Dies würde auch die Beschriftung der Zustände mit booleschen Variablen unmöglich machen. Dieses Problem lässt sich jedoch umgehen, indem die parallele Komposition bereits auf symbolischer Ebene durchgeführt wird. Dazu genügt es, die genannten kleineren Komponenten laut Definition 4.1 zu beschriften, und zwar ohne die Variable x_b, weil die Verbotenen Zustände nur anhand des Produktes $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ berechnet werden können. Die Beschriftung von $\mathcal{A}_\mathcal{P}$ mit den Variablen $\mathcal{V}^x \cup \mathcal{V}^y \cup \{x_m\}$ ergibt sich dann aus der parallelen Komposition. Eine symbolische Implementierung der parallelen Komposition ist z.B. mit den gängigen BDD-Paketen problemlos durchführbar, zumal die benötigten Operationen grundsätzlich vorhanden sind.

Das gleiche gilt für den Automaten $\mathcal{A}_\mathcal{E}$. Damit kann auch $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ symbolisch berechnet werden. Die Beschriftung aus Definition 4.1 lässt sich vervollständigen, indem die verbotenen Zustände von $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ anhand der symbolischen Darstellungen der Automaten $\mathcal{A}_\mathcal{P}$ und $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ mit der booleschen Variable x_b beschriftet werden. Eine Lösung für dieses Problem wurde im Rahmen der Vorbereitung dieser Arbeit gefunden und in [91] veröffentlicht.

Die Übersetzung eines in dieser Weise beschrifteten Automaten in eine Kripke-Struktur wird mit der folgenden Definition festgelegt. Dabei kommt die Unterscheidung zwischen steuerbaren und nicht steuerbaren Transitionen zum Tragen (s. Abschnitt 2.4.1). Außerdem wird auch die Information über die verbotenen und die markierten Zustände übernommen, die bereits in der Beschriftung des Automaten enthalten ist.

Definition 4.4 (Kripke-Struktur eines Automaten) *Sei der Automat $\mathcal{A} = \langle \Sigma, Q, \delta, q^0, M \rangle$ eine Spezifikation der Form $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$, dessen Zustände nach Definition 4.1 mit den booleschen Variablen $\mathcal{V}_\mathcal{A}$ beschriftet sind, und sei $\Sigma_u \subseteq \Sigma$ die Menge seiner nicht steuerbaren Ereignisse. Dann ist $\mathcal{K}_\mathcal{A} := \langle \mathcal{S}, \mathcal{I}, \mathcal{R}, \mathcal{L} \rangle$ die Kripke-Struktur zu $\mathcal{A}$, die wie folgt über den Variablen $\mathcal{V} := \mathcal{V}_\mathcal{A} \cup \{x_u\}$ definiert wird:*

- $\mathcal{S} := Q \times \{0,1\}$
- $\mathcal{I} := \{(q^0,0),(q^0,1)\}$
- $\mathcal{R}((q,0),(q',0)) :\Leftrightarrow \exists \sigma \in \Sigma_u.\delta(q,\sigma,q')$
- $\mathcal{R}((q,1),(q',1)) :\Leftrightarrow \exists \sigma \in \Sigma.\delta(q,\sigma,q')$
- $\mathcal{L}((q,0)) := \mathcal{L}_{\mathcal{A}}(q) \cup \{x_u\}$
- $\mathcal{L}((q,1)) := \mathcal{L}_{\mathcal{A}}(q)$.

In Definition 4.4 sind $\mathcal{S}$ die Zustandsmenge der Kripke-Struktur, $\mathcal{I} \subseteq \mathcal{S}$ die Menge der Initialzustände und $\mathcal{R}$ die Transitionsrelation. Für jeden Zustand $q \in Q$ werden zwei Zustände in $\mathcal{S}$ angelegt, nämlich $(q,0)$ und $(q,1)$, so dass $|\mathcal{S}| = 2|Q|$. Ein Zustandspaar $((q,0),(q',0))$ ist genau dann in $\mathcal{R}$, wenn auf $\mathcal{A}$ eine nicht steuerbare Transition von q nach q' führt. Hingegen gilt $((q,1),(q',1)) \in \mathcal{R}$ genau dann, wenn auf $\mathcal{A}$ eine Transition – steuerbar oder nicht – von q nach q' führt. Damit gilt $|\mathcal{R}| \leq 2|\Sigma||Q|$, und $\mathcal{K}_{\mathcal{A}}$ besteht aus zwei getrennten Kopien der Zustände von $\mathcal{A}$. Die Funktion $\mathcal{L}$ kopiert die Beschriftung jedes Zustands $q \in Q$ auf die Zustände $(q,0)$ und $(q,1)$. Um die beiden Hälften der Kripke-Struktur voneinander zu unterscheiden, werden die Zustände $(q,0)$ darüber hinaus mit der Variablen x_u beschriftet. Einige wichtige Mengen sind demnach:

$$
\begin{aligned}
[\![x_u]\!]_{\mathcal{K}_{\mathcal{A}}} &:= \{(q,0) \mid q \in Q\} \\
[\![\overline{x}_u]\!]_{\mathcal{K}_{\mathcal{A}}} &:= \{(q,1)\} \mid q \in Q\} \\
[\![x_b x_u]\!]_{\mathcal{K}_{\mathcal{A}}} &:= \{(q,0) \mid q \textit{ verletzt Bedingung 2.1}\} \\
[\![\overline{x}_m \overline{x}_u]\!]_{\mathcal{K}_{\mathcal{A}}} &:= \{(q,1) \mid q \notin M\}.
\end{aligned}
$$

Beispiel 4.5 Die Kripke-Struktur zu dem Automaten $\mathcal{A}_{\mathcal{P}} \times \mathcal{A}_{\mathcal{E}}$ aus Beispiel 4.2 wird in Abbildung 4.3 wiedergegeben. Die Einteilung der Ereignisse ist wie in Beispiel 2.19, nämlich $\Sigma_c = \{\mathsf{Start\_A}\}$ und $\Sigma_u = \{\mathsf{Start\_B}, \mathsf{Lade\_A}, \mathsf{Lade\_B}, \mathsf{Lager}\}$. Die Beschriftung der Zustände des Automaten wird übernommen, wobei die Zustände der linken Hälfte der Kripke-Struktur zusätzlich mit der Variablen x_u beschriftet werden. Es gibt auf dieser Hälfte keine steuerbaren Transitionen. □

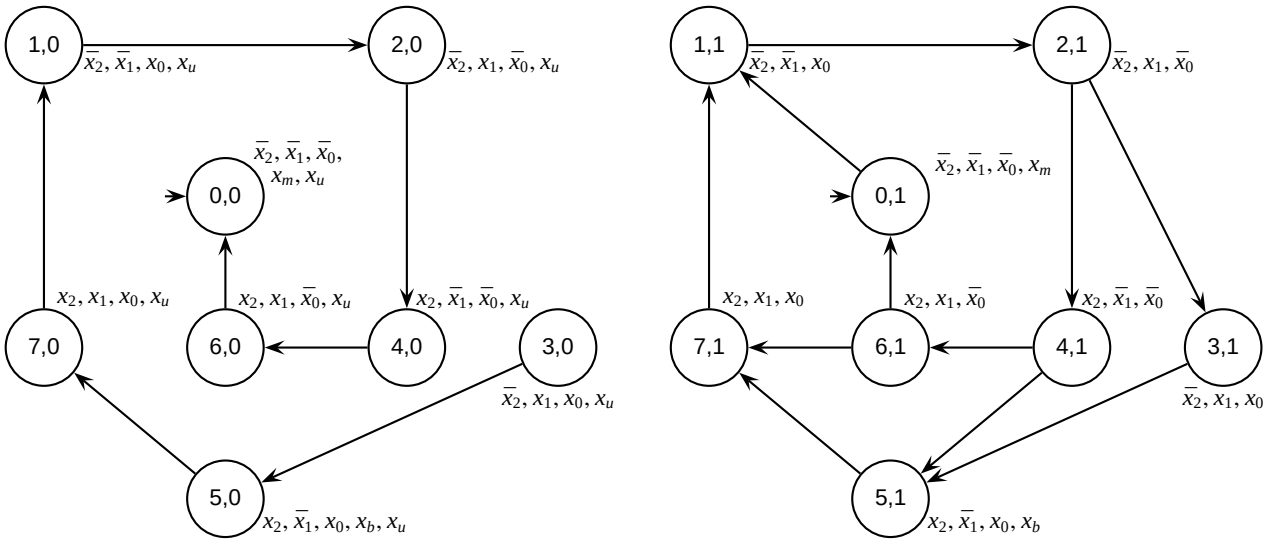


Abbildung 4.3: Die Kripke-Struktur für den Automaten $\mathcal{A}_{\mathcal{P}} \times \mathcal{A}_{\mathcal{E}}$ aus Beispiel 4.5

In Anlehnung an Abbildung 4.3 wird der Teil der Struktur mit den Zuständen x_u fortan auch *linke Hälfte*, die mit den Zuständen $\overline{x}_u$ auch *rechte Hälfte* der Kripke-Struktur genannt.

Um von einer Hälfte der Kripke-Struktur in die andere zu wechseln, wird für eine gegebene Menge $P \subseteq \mathcal{S}$ die Zustandstransformation

$$\pi(P) := \{(q, \neg i) \mid (q,i) \in P\}$$

definiert. Sie gibt eine Zustandsmenge zurück, in der die Zustände in P durch ihre Kopie auf der jeweils anderen Hälfte der Kripke-Struktur ausgetauscht sind. Diese Funktion ist monoton (vgl. Definition 3.26) und lässt sich deshalb in Definition 3.18 verwenden. Auf symbolischer Ebene gilt

$$\kappa_\pi(\varphi_P) := [\varphi_P]^{\neg x_u, x_u}_{x_u, \neg x_u} .$$

Die Funktion κ_π kippt die Variable x_u, die die Strukturhälfte angibt, zu der die Zustände in $[\![\varphi_P]\!]$ gehören. Da sonst keine weiteren Funktionen dieser Art benötigt werden, wird fortan die vereinfachte Notation κ für κ_π verwendet.

Die Abbildung einer Zustandsmenge $P \subseteq \mathcal{S}$ der Kripke-Struktur zurück auf den Automaten, wie auf der rechten Seite der Abbildung 4.1 vorgesehen, wird wie folgt definiert:

Definition 4.6 (Abbildung von $\mathcal{K}_\mathcal{A}$ auf $\mathcal{A}$) *Gegeben seien ein Automat $\mathcal{A}$ mit Zustandsmenge $Q_\mathcal{A}$, sowie die nach Definition 4.4 erstellte Kripke-Struktur $\mathcal{K}_\mathcal{A} = \langle \mathcal{S}, \mathcal{I}, \mathcal{R}, \mathcal{L} \rangle$ und eine Zustandsmenge $P \subseteq \mathcal{S}$. Dann ist*

$$P \mapsto \{q \in Q_\mathcal{A} \mid (q,0) \in P \vee (q,1) \in P\}$$

die Abbildung der Zustandsmenge P auf $Q_\mathcal{A}$.

Auf symbolischer Ebene entspricht dies der existentiellen Quantifizierung der Variablen x_u (s. Definition 3.13). Für einen Automaten $\mathcal{A}$ und die dazugehörige Kripke-Struktur $\mathcal{K}_\mathcal{A}$, sowie den booleschen Ausdruck φ_P, der die Zustandsmenge P auf $\mathcal{K}_\mathcal{A}$ symbolisch darstellt, gilt

$$[\![\exists x_u.\varphi_P]\!]_\mathcal{A} = \{q \in Q_\mathcal{A} \mid (q,0) \in P \vee (q,1) \in P\} .$$

Beispiel 4.7 Sei $\varphi_P := (\overline{x_2}\,\overline{x_1} \vee x_2 x_1 \vee \overline{x_0}) \wedge x_u$ die symbolische Darstellung der Menge $P := \{(0,1),(1,1),(2,1),(4,1),(6,1),(7,1)\}$ auf der Kripke-Struktur in Abbildung 4.3. Dann ist

$$\exists x_u.\varphi_P = \overline{x_2}\,\overline{x_1} \vee x_2 x_1 \vee \overline{x_0}$$

die Abbildung von P auf den Automaten in Abbildung 4.2. Wie zu erwarten, entspricht dieser Ausdruck der Menge $\{0,1,2,4,6,7\}$. □

Definition 4.6 vervollständigt die Reihe der Werkzeuge für den Umgang mit Automaten und Kripke-Strukturen, die für die Transformationen auf der linken und der rechten Seite in Abbildung 4.1 notwendig sind. In dem nächsten Abschnitt wird aus der algorithmischen Darstellung der Überwachersynthese in Kapitel 2 ein μ-Kalkül-Gleichungssystem gewonnen, mit dem der obere Schritt in der genannten Abbildung vollzogen werden kann.

4.2 Vom Algorithmus zum Gleichungssystem

In diesem Abschnitt wird das Überwachersyntheseproblem in ein Modellprüfungsproblem umgewandelt. Zunächst werden die Algorithmen für die Überwachersynthese aus Kapitel 2 hinsichtlich ihrer Übersetzbarkeit in ein μ-Kalkül-Gleichungssystem bewertet.

Der Ansatz von Wonham und Ramadge besteht aus Algorithmus 2.18 auf Seite 23. In diesem bilden die Schritte 2 und 3 eine Schleife, die in Schritt 2 die Berechnung der verbotenen Zustände anhand der Automaten $\mathcal{A}_\mathcal{P}$ und $\mathcal{A}_\mathcal{S} := \mathsf{Tr}(\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E})$ enthält, wobei letzterer in jeder Iteration erneuert wird. Außer $\mathcal{A}_\mathcal{S}$ muss also während der ganzen Berechnung auch der Automat $\mathcal{A}_\mathcal{P}$ zur Verfügung stehen. Das Ziel, das hier verfolgt werden soll, besteht jedoch in der Übersetzung des Überwachersyntheseproblems in ein Modellprüfungsproblem, bei dem ausschließlich die Kripke-Struktur $\mathcal{K}_{\mathsf{Tr}(\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E})}$ zur Verfügung steht. Folglich sollte die Berechnung der anfänglich verbotenen Zustände vor der Übersetzung des Automaten $\mathsf{Tr}(\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E})$ in diese Kripke-Struktur

geschehen, danach aber nicht wiederholt werden. Im Prinzip wäre es möglich, die Kripke-Struktur in Definition 4.4 mit der notwendigen Information aus $\mathcal{A}_\mathcal{P}$ zu erweitern. Es stellt sich jedoch heraus, dass der Ansatz von Kumar und Garg ohne diese Erweiterung auskommt.

Der Ansatz von Kumar und Garg ist auf Seite 25 als Algorithmus 2.20 wiedergegeben. Im Gegensatz zu Algorithmus 2.18 ist die Berechnung der ersten verbotenen Zustände in Schritt 2.1 nicht Teil der Iterationsschleife, die sich über die Schritte 3 bis 5 erstreckt. Darüber hinaus beinhaltet die Kripke-Struktur in Definition 4.4 alle Informationen, die für die Berechnungen innerhalb dieser Schleife notwendig sind. Das gleiche gilt für die Variante dieses Ansatzes, die zu Algorithmus 2.22 auf Seite 27 führt. Deshalb geht die Übersetzung der Überwachersynthese in den μ-Kalkül von diesen Algorithmen aus.

4.2.1 Übersetzung des Algorithmus von Kumar und Garg

Als Startpunkt dient Algorithmus 2.20 in Schritt 2.1 die Menge Q_b der verbotenen Zustände von $\mathcal{A}_\mathcal{S} := \mathsf{Tr}(\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E})$, die auf dem Automaten symbolisch durch den Ausdruck x_b dargestellt werden. In Schritt 3 fügt der Algorithmus dieser Menge die Zustände hinzu, die über nicht steuerbare Transitionen einen Zustand in Q_b erreichen können. Die Überlegung dabei ist, dass diese Zustände nicht Teil des Überwachers sein können, zumal ansonsten ein nicht steuerbarer Pfad von ihnen zu einem verbotenen Zustand entstehen würde. Die gleiche Berechnung kann auf der linken Hälfte der Kripke-Struktur nachvollzogen werden, auf der nur die nicht steuerbaren Transitionen abgebildet sind. Die gesuchten Zustände werden in der Menge u_b gesammelt. Beginnend mit $x_b x_u$ müssen die existentiellen Vorgänger dieser Zustände schrittweise gesammelt werden, bis keine weiteren Zustände mehr hinzukommen. Dies entspricht der Fixpunktgleichung

$$u_b \overset{\mu}{=} \Diamond u_b \vee x_b x_u. \tag{4.1}$$

Die Konjunktion $x_u x_b$ beschränkt die Zustände, die gesammelt werden, auf die linke Hälfte der Kripke-Struktur. Weil der Ausdruck ein kleinster Fixpunkt ist, dessen Berechnung mit $u_b^0 := 0$ beginnt, können nur Zustände aus der linken Hälfte der Struktur hinzukommen.

Als nächstes nehmen Schritte 4.1 und 4.2 in Algorithmus 2.20 die Einschränkung des Automaten $\mathcal{A}_\mathcal{S}$ auf die Menge $\overline{Q_b}$ vor. In Schritt 4.3 wird die Menge der co-erreichbaren Zustände dieses Automaten berechnet. Co-erreichbare Zustände von $\mathcal{A}_\mathcal{S}$ können auf der rechten Hälfte der Kripke-Struktur berechnet werden, zumal für die Co-Erreichbarkeit sowohl steuerbare als nicht steuerbare Transitionen zu berücksichtigen sind. Es gilt, die Zustände zu bestimmen, die einen Pfad zu einem markierten Zustand auf dieser Hälfte der Struktur haben. Die Einschränkung auf die Menge $\overline{Q_b}$ bedeutet, dass nur Zustände gesammelt werden dürfen, die nicht bereits auf der linken Hälfte der Struktur der Menge u_b hinzugefügt wurden. Diese Zustände werden mit Hilfe der Funktion κ von der linken auf die rechte Hälfte der Kripke-Struktur gespiegelt. Die co-erreichbaren Zustände werden in der Menge u_c gesammelt. Beginnend mit $x_m \overline{x_u}$ müssen die existentiellen Vorgänger dieser Zustände, die nicht in $\kappa(u_b)$ sind, gesammelt werden, bis keine weiteren Zustände mehr hinzukommen. Dies entspricht der Fixpunktgleichung

$$u_c \overset{\mu}{=} (\Diamond u_c \vee x_m \overline{x_u}) \wedge \neg \kappa(u_b)\,. \tag{4.2}$$

In Schritt 4.4 wird die Menge der neuen nicht co-erreichbaren Zustände ermittelt. Diese werden – sofern die Menge nicht leer ist – in Schritt 5 der Menge Q_b hinzugefügt, so dass Q_b dann alle Zustände enthält, die bis zu diesem Schritt als nicht co-erreichbar oder verboten erkannt wurden. Auf dem Automaten muss hierfür lediglich das Komplement der Menge der co-erreichbaren Zustände gebildet werden. Auf der Kripke-Struktur entspricht dies jedoch nicht einfach dem Komplement der Menge u_c, zumal dies unerwünschte Zustände der linken Hälfte mit einbringen würde. Vielmehr muss das Komplement durch eine Konjunktion mit $\overline{x_u}$ wieder auf die rechte Hälfte der Struktur beschränkt werden. Aus Lemma 3.29 folgt, dass das Komplement der Menge u_c durch den größten Fixpunkt

$$u_n \overset{\nu}{=} \Box u_n \wedge (\overline{x_m} \vee x_u) \vee \kappa(u_b)$$

gegeben wird, wobei die Negation von u_c in u_n umbenannt wurde, da der Fixpunkt nun die Menge der nicht co-erreichbaren Zustände darstellt. Nach der Konjunktion der rechten Seite der Gleichung mit $\overline{x_u}$ ergibt sich dann

$$u_n \overset{\nu}{=} \Box u_n \wedge \overline{x_m}\,\overline{x_u} \vee \kappa(u_b) \wedge \overline{x_u}.$$

Da $\kappa(u_b)$ immer eine Menge auf der rechten Hälfte der Kripke-Struktur darstellt, kann die zweite Konjunktion mit $\overline{x_u}$ weggelassen werden. Damit ergibt sich

$$u_n \overset{\nu}{=} \Box u_n \wedge \overline{x_m}\,\overline{x_u} \vee \kappa(u_b) \tag{4.3}$$

als Ausdruck für die Zustände, die für einen gegebenen Wert u_b nicht co-erreichbar oder verboten sind.

Damit stellen die nacheinander berechneten Fixpunkte von u_n exakt die Zustandsmengen von $\mathcal{K}_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}$ dar, deren Abbildungen gemäß Definition 4.6 mit den Zustandsmengen von $\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}$ identisch sind, die Algorithmus 2.20 in jeder Iteration der Schleife über die Schritte 3 bis 5 berechnet. Diese Fixpunkte selbst werden in jeder Iteration größer, bis keine weiteren Zustände dazukommen. Sie konvergieren deshalb zu dem kleinsten Fixpunkt von

$$u_{bn} \overset{\mu}{=} u_n. \tag{4.4}$$

Wenn der Algorithmus auf Schritt 3 zurückspringt, entspricht Q_b dem zuletzt berechneten Fixpunkt von u_n. Damit Gleichung 4.1 diese Zustände berücksichtigen kann, wird sie um eine Disjunktion mit dem Ausdruck u_{bn} ergänzt. Zuvor müssen die Zustände noch auf die linke Hälfte der Kripke-Struktur gespiegelt werden, was in Gleichung 4.4 eingebaut werden kann. Mit diesen Änderungen entsteht aus den Gleichungen 4.1, 4.3 und 4.4 das Gleichungssystem im folgenden Theorem, das die Überwachersynthese in ein μ-Kalkül-Modellprüfungsproblem überführt.

Theorem 4.8 (Überwachersynthese im μ-Kalkül nach Algorithmus 2.20) *Gegeben sei ein Automat $\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}$, der nach Definition 4.1 beschriftet ist und das gewünschte Verhalten für das System $\mathcal{A}_\mathcal{P}$ beschreibt. Seien wie in Algorithmus 2.20 $\mathcal{A}_\mathcal{S} := \mathsf{Tr}(\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E})$ und $\mathcal{A}_\mathcal{S}|_{\overline{Q_b}}$ der gesuchte Überwacher. Dann ist $Q_b = \left[\!\left[\exists x_u.u_{bn}^\infty\right]\!\right]_{\mathcal{A}_\mathcal{S}}$, wobei u_{bn}^∞ die Lösung des Gleichungssystems*

$$\left\{\begin{array}{lcl} u_b & \overset{\mu}{=} & \Diamond u_b \vee x_b x_u \vee u_{bn} \\ u_n & \overset{\nu}{=} & \Box u_n \wedge \overline{x_m}\,\overline{x_u} \vee \kappa(u_b) \\ u_{bn} & \overset{\mu}{=} & \kappa(u_n) \end{array}\right. \tag{4.5}$$

über den Variablen der Kripke-Struktur $\mathcal{K}_{\mathsf{Tr}(\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E})}$ ist, die mit Hilfe der Formeln

$$\left[\begin{array}{lcl} u_{bn}^{i_1+1} & := & \kappa\left(u_n^{i_1,\infty}\right) \\ u_n^{i_1,i_2+1} & := & \Box u_n^{i_1,i_2} \wedge \overline{x_m}\,\overline{x_u} \vee \kappa\left(u_b^{i_1,0,\infty}\right) \\ u_b^{i_1,i_2,i_3+1} & := & \Diamond u_b^{i_1,i_2,i_3} \vee x_b x_u \vee u_{bn}^{i_1} \end{array}\right. \tag{4.6}$$

innerhalb der Zeit $O\left(\left|Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}\right|^2 |\Sigma|\right)$ berechnet werden kann. Ist $M_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}} = Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}$, reduziert sich die Komplexitätsgrenze auf $O\left(\left|Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}\right| |\Sigma|\right)$.

Beweis: Wie der Abhängigkeitsgraph in Abbildung 4.4 zeigt, besteht das Gleichungssystem 4.5 aus einer einzigen maximalen Komponente. Es kann deshalb mit Algorithmus 3.57 berechnet werden, was zu den Iterationsformeln

$$\left[\begin{array}{lcl} u_{bn}^{i_1+1} & := & \kappa\left(u_n^{i_1,\infty}\right) \\ u_n^{i_1,i_2+1} & := & \Box u_n^{i_1,i_2} \wedge \overline{x_m}\,\overline{x_u} \vee \kappa\left(u_b^{i_1,i_2,\infty}\right) \\ u_b^{i_1,i_2,i_3+1} & := & \Diamond u_b^{i_1,i_2,i_3} \vee x_b x_u \vee u_{bn}^{i_1} \end{array}\right.$$

führt. Die erste und die dritte dieser Formeln sind bereits in den Formeln 4.6 enthalten, die zweite weicht jedoch von diesen ab. Der Unterschied liegt in dem zweiten Iterationszähler der Variablen u_b. Während es im allgemeinen Fall notwendig ist, für jeden Schritt $u_n^{i_1,i_2+1}$ einen neuen Fixpunkt $u_b^{i_1,i_2,\infty}$ zu berechnen, bleibt der Zähler i_2 in den Formeln 4.6 auf 0 stehen. Dies hat zur Folge, dass nur dann ein neuer Fixpunkt für u_b berechnet wird, wenn sich der Iterationszähler i_1 erhöht. Bei der Berechnung eines Fixpunktes für u_n werden folglich die Iterationsschritte so ausgeführt, als gäbe es keinen tiefer verschachtelten Fixpunkt zu berechnen. Diese Vereinfachung wird durch den Abhängigkeitsgraphen in Abbildung 4.4 gerechtfertigt. Aus diesem ist ersichtlich, dass eine neue Berechnung von u_b nach jedem Schritt von u_n nichts ändern würde.
Folgende Beobachtungen zeigen, dass die Formeln 4.6 zu den gleichen Berechnungen führen wie Algorithmus 2.20:

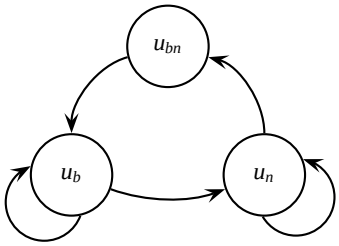


Abbildung 4.4: Der Abhängigkeitsgraph des Gleichungssystems 4.5

1. Durch den Aufbau der Kripke-Struktur $\mathcal{K}_{\mathsf{Tr}(\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E})}$ entspricht der Ausdruck $x_b x_u$ der Menge Q_b, die der Algorithmus in Schritt 2.1 ermittelt.

2. In Schritt 2.2 bricht der Algorithmus vorzeitig ab, falls $Q_b = \emptyset$. Diese Überprüfung ist jedoch nicht zwingend, zumal der Algorithmus auch korrekt arbeitet, wenn die Schritte 3 und 4 trotzdem durchlaufen werden. Es wird dann in Schritt 5 mit demselben Ergebnis wie in Schritt 2.2 abgebrochen. Dies bedeutet, dass die Lösung des Gleichungssystems korrekt ist, solange die weiteren Berechnungen mit denen des Algorithmus übereinstimmen.

3. Der erste Fixpunkt, der für die Lösung des Gleichungssystems berechnet wird, ist $u_b^{0,0,\infty}$. Da u_{bn}^0 gleich 0 ist, verhält sich die Gleichung für $u_b^{i_1,i_2,i_3+1}$ zunächst wie Gleichung 4.1. Wie auf Seite 66 erläutert, werden damit auf der linken Hälfte der Kripke-Struktur die Zustände gesammelt, deren Abbildung auf den Automaten $\mathcal{A}_\mathcal{S}$ die Zustandsmenge ergibt, die der Algorithmus in dem ersten Durchgang in Schritt 3 ermittelt. Somit entspricht $u_b^{0,0,\infty}$ der Menge Q_b nach der ersten Berechnung von Schritt 3.

4. In Schritt 4 berechnet Algorithmus 2.20 die nicht co-erreichbaren Zustände des Automaten $\mathcal{A}_\mathcal{S}|_{\bar{Q}_b}$. In Schritt 5 werden diese Zustände – sofern nicht abgebrochen wird – der Menge Q_b hinzugefügt, so dass diese bei dem Rücksprung auf Schritt 3 alle Zustände enthält, die nicht co-erreichbar sind oder bereits als verboten erkannt wurden. Wie auf Seite 67 erläutert, werden diese Zustände auf der rechten Hälfte der Kripke-Struktur mit Hilfe der Gleichung 4.3 ermittelt. Ein Vergleich mit den Gleichungen 4.6 zeigt, dass die zweite Gleichung dieser entspricht, und dass dort der passende Wert $u_b^{0,0,\infty}$ eingesetzt wird. Da die Berechnung von $u_n^{0,\infty}$ ohne jegliche Wiederholungen der Berechnungen für u_b geschieht, wird an dieser Stelle genauso gerechnet, wie bei der Lösung von Gleichung 4.3. Somit entspricht $u_n^{0,\infty}$ der Zustandsmenge Q_b die der Algorithmus in Schritt 5 ermittelt, bevor auf Schritt 3 zurückgesprungen wird.

5. Es folgt die Berechnung $u_{bn}^1 := \kappa\left(u_n^{0,\infty}\right)$. Dabei werden lediglich die bisher gesammelten Zustände auf die linke Hälfte der Kripke-Struktur gespiegelt und auf die Variable u_{bn} übertragen, so dass dieser Schritt keinerlei Berechnungen des Algorithmus entspricht.

6. Wenn Algorithmus 2.20 auf Schritt 3 zurückspringt, enthält Q_b die Zustände, die nun in $u_{bn}^{i_1}$ stehen. Dies wird in der Gleichung für $u_b^{i_1,0,\infty}$ durch die Disjunktion mit $u_{bn}^{i_1}$ berücksichtigt. Diese sorgt dafür, dass $u_b^{i_1,0,1}$ gleich $u_{bn}^{i_1}$ ist. Damit verhält sich die Gleichung für u_b auch für $i_1 > 0$ wie Schritt 3 im Algorithmus.

7. Die Berechnung von u_n und u_{bn} in den weiteren Durchgängen geschieht wie im ersten Durchgang. Damit entsprechen die Fixpunkte von u_n jeweils den Berechnungen in den Schritten 4 und 5.

8. Der Algorithmus terminiert, wenn in Schritt 4 keine neuen nicht co-erreichbaren Zustände gefunden werden. Nach Algorithmus 3.57 ist das Abbruchkriterium für die Formeln 4.6 das Erreichen des Fixpunktes u_{bn}^∞. Die Äquivalenz dieser beiden Kriterien wird in zwei Schritten gezeigt. Der erste ist trivial: Ist der Fixpunkt u_{bn}^∞ erreicht, ändern sich auch die anderen Fixpunkte nicht mehr, und es können keine neuen nicht co-erreichbaren Zustände gefunden werden. Die umgekehrte Richtung der Äquivalenz ist wie folgt: Sei i_1 der Wert, für den gilt, dass $u_n^{i_1,\infty}$ keine neuen nicht co-erreichbaren Zustände findet. Dann ist $u_n^{i_1,\infty} = \kappa\left(u_b^{i_1,0,\infty}\right)$. Daraus folgt $u_{bn}^{i_1+1} = u_b^{i_1,0,\infty}$. Folglich wird auch $u_b^{i_1+1,0,\infty} = u_b^{i_1,0,\infty}$ sein, und somit $u_n^{i_1+1,\infty} = u_n^{i_1,\infty}$. Damit ist $u_{bn}^{i_1+2} = u_{bn}^{i_1+1} = u_{bn}^\infty$.

Damit entspricht u_{bn}^{∞} der Menge Q_b, wenn Algorithmus 2.20 terminiert, und es gilt $Q_b = \left[\!\left[\exists x_u.u_{bn}^{\infty}\right]\!\right]_{\mathcal{A}_S}$.

Für die Komplexitätsanalyse seien $\mathcal{S}$ und $\mathcal{R}$ die Zustandsmenge bzw. die Transitionsrelation der Kripke-Struktur $\mathcal{K}_{\mathsf{Tr}(\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E})}$. Theorem 3.61 liefert für die Komplexität der Lösung eines Gleichungssystems der Alternierungstiefe 3 mit Algorithmus 3.57 die obere Grenze $O\left(|\mathcal{S}|^3\,|\mathcal{R}|\right)$. Die Vereinfachung der Iterationsformeln 4.6 gegenüber dem allgemeinen Fall führt jedoch zu einem besseren Ergebnis. Die Berechnung des Fixpunktes $u_b^{i_1,0,\infty}$ beginnt zwar jedes Mal von neuem mit 0, erreicht aber bereits nach dem ersten Schritt den Wert $u_{bn}^{i_1}$, der mindestens so groß ist, wie der zuletzt berechnete Fixpunkt $u_b^{i_1-1,0,\infty}$. Dies bewirkt, dass die Gleichung für $u_b^{i_1,i_2,i_3+1}$ höchstens $2\,|\mathcal{S}|$-mal ausgewertet werden kann, bevor der Fixpunkt für u_{bn} erreicht wird. In jeder Auswertung erfolgt die Berechnung der existentiellen Vorgänger innerhalb der Zeit $O(|\mathcal{R}|)$. Folglich liegt der Zeitaufwand dieser Gleichung insgesamt in $O(|\mathcal{S}|\,|\mathcal{R}|)$.

Die Berechnung eines Fixpunktes $u_n^{i_1,\infty}$ kann innerhalb der Zeit $O(|\mathcal{R}|)$ durchgeführt werden. Bei $O(|\mathcal{S}|)$ Wiederholungen ergibt dies ebenfalls einen Aufwand, der in $O(|\mathcal{S}|\,|\mathcal{R}|)$ liegt. Die gesamten Berechnungen der $u_{bn}^{i_1}$ finden innerhalb der Zeit $O(|\mathcal{S}|)$ statt, und die Kripke-Struktur selbst lässt sich innerhalb der Zeit $O(|\mathcal{R}|)$ erstellen. Insgesamt liegt die Komplexität des Algorithmus also in $O(|\mathcal{S}|\,|\mathcal{R}|)$. Da $|\mathcal{S}| \leq 2\left|Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}\right|$ und $|\mathcal{R}| \leq 2\left|Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}\right|\,|\Sigma|$ (siehe Definition 4.4), ergibt sich für den gesamten Algorithmus die Komplexität $O\left(\left|Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}\right|^2\,|\Sigma|\right)$.

Im Sonderfall $M_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}} = Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}$ sind alle Zustände co-erreichbar. Dann findet die Gleichung für u_n keine neuen Zustände und der Fixpunkt u_{bn}^{∞} wird bereits im ersten Durchgang erreicht. Damit lassen sich alle Berechnungen innerhalb der Zeit $O(|\mathcal{R}|)$ bzw. $O(\left|Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}\right|\,|\Sigma|)$ durchführen. Beide Ergebnisse zur Komplexität stimmen mit den Angaben für Algorithmus 2.20 in Abschnitt 2.5.2 überein. □

Korollar 4.9 *Um Iterationen zu sparen, kann die Berechnung der Formeln 4.6 bereits dann abgebrochen werden, wenn* $u_n^{i_1,\infty} = \kappa\left(u_b^{i_1,0,\infty}\right)$. *Es gilt dann* $u_{bn}^{\infty} = u_b^{i_1,0,\infty}$.

Beweis: Der Beweis ist in dem Beweis von Theorem 4.8 unter Punkt 8 enthalten. □

Beispiel 4.10 Als Beispiel wird der Überwacher für die Fertigungszelle aus Kapitel 2 berechnet. Die dazugehörige Kripke-Struktur wurde bereits in Beispiel 4.5 erstellt und ist auf Abbildung 4.3 zu sehen. Es folgen die Iterationsschritte der Formeln 4.6. Um die Lesbarkeit zu erhöhen, werden die Mengen nicht in symbolischer Darstellung, sondern explizit angegeben.

$u_{bn}^{0} = \{\}$
$\quad u_n^{0,0} = \mathcal{S}$
$\quad\quad u_b^{0,0,0} = \{\}$
$\quad\quad u_b^{0,0,1} = \{(5,0)\}$
$\quad\quad u_b^{0,0,2} = \{(3,0),(5,0)\}$
$\quad\quad u_b^{0,0,\infty} = \{(3,0),(5,0)\}$
$\quad u_n^{0,1} = \{(1,1),(2,1),(3,1),(4,1),(5,1),(6,1),(7,1)\}$
$\quad u_n^{0,2} = \{(1,1),(2,1),(3,1),(4,1),(5,1),(7,1)\}$
$\quad u_n^{0,3} = \{(1,1),(2,1),(3,1),(5,1),(7,1)\}$
$\quad u_n^{0,4} = \{(1,1),(3,1),(5,1),(7,1)\}$
$\quad u_n^{0,5} = \{(3,1),(5,1),(7,1)\}$
$\quad u_n^{0,6} = \{(3,1),(5,1)\}$
$\quad u_n^{0,\infty} = \{(3,1),(5,1)\}$

Das Haltekriterium aus Korollar 4.9, nämlich $u_n^{i_1,\infty} = \kappa\left(u_b^{i_1,0,\infty}\right)$, ist für $i_1 = 0$ bereits erfüllt. Deshalb gilt $u_{bn}^{\infty} = u_b^{0,0,\infty} = \{(3,0),(5,0)\}$. Die Abbildung dieser Menge auf den Automaten $\mathsf{Tr}\,(\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E})$ ist die Menge $\{3,5\}$. Der Überwacher – die Einschränkung von $\mathsf{Tr}\,(\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E})$ auf das Komplement dieser Menge – ist in Abbildung 4.5 zu sehen und bestätigt das Ergebnis der Beispiele 2.19 und 2.21. □

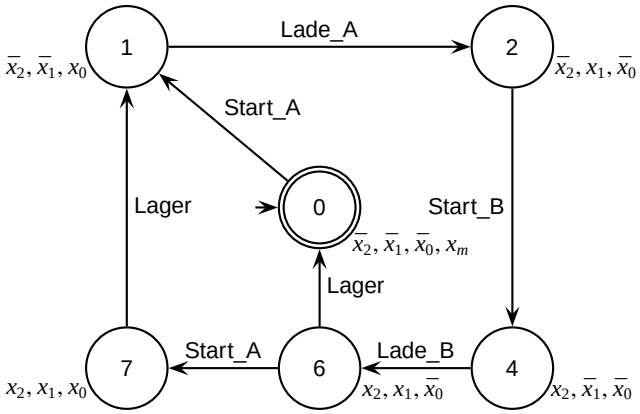


Abbildung 4.5: Der Überwacher für Beispiel 4.10

In Kapitel 3 wurden obere Grenzen für die Lösung eines Gleichungssystems aus einer maximalen Komponente festgelegt. Da das Gleichungssystem 4.6 die Alternierungstiefe 3 besitzt, gilt laut Theorem 3.61 die obere Grenze $O(|\mathcal{S}|^3\,|\mathcal{R}|)$. Bei $|\mathcal{S}| \leq 2\left|Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}\right|$ und $|\mathcal{R}| \leq 2\left|Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}\right||\Sigma|$ ergibt dies $O\left(\left|Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}\right|^4|\Sigma|\right)$. Damit liegen die Formeln 4.6 mit einer Komplexität von $O\left(\left|Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}\right|^2|\Sigma|\right)$ deutlich unter dieser Grenze. Dieser Unterschied erklärt sich dadurch, dass die Abschätzung in Theorem 3.61 nicht auf Besonderheiten des Gleichungssystems eingeht. Die geringere Komplexität der Formeln 4.6 entsteht einerseits durch Berücksichtigung der Abhängigkeiten zwischen den Gleichungen, andererseits durch die Disjunktion mit u_{bn} in der Gleichung für u_b, die bewirkt, dass der Fixpunkt schneller erreicht wird.

Damit ist die Übersetzung von Algorithmus 2.20 in den μ-Kalkül beendet. Die gleiche Prozedur lässt sich auch auf Algorithmus 2.22 (s. Seite 27) anwenden.

4.2.2 Übersetzung der Variante des Algorithmus von Kumar und Garg

In diesem Abschnitt wird auch der Algorithmus 2.22 in den μ-Kalkül übersetzt. Interessant ist, dass dabei ein Gleichungssystem der Alternierungstiefe 2 entsteht, während das Gleichungssystem 4.6 in Theorem 4.8 die Alternierungstiefe 3 hat. Obwohl die Komplexität der Lösung dadurch nicht weiter verringert wird, kommt die geringere Alternierungstiefe der weiteren Entwicklung in den Kapiteln 5 und 6 zugute.

Theorem 4.11 (Überwachersynthese im μ-Kalkül nach Algorithmus 2.22) *Gegeben sei ein Automat $\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}$, der nach Definition 4.1 beschriftet ist und das gewünschte Verhalten für das System $\mathcal{A}_\mathcal{P}$ beschreibt. Seien wie in Algorithmus 2.22 $\mathcal{A}_\mathcal{S} := \mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}$ und $\mathcal{A}_\mathcal{S}|_{\bar{Q}_b}$ der gesuchte Überwacher. Dann ist $Q_b = \left[\!\left[\exists x_u.u_{bn}^\infty\right]\!\right]_{\mathcal{A}_\mathcal{S}}$, wobei u_{bn}^∞ die Lösung des Gleichungssystems*

$$\left\{\begin{array}{lcl} u_n & \overset{\nu}{=} & \Box u_n \wedge \bar{x}_m\bar{x}_u \vee u_{bn} \\ u_b & \overset{\mu}{=} & \Diamond u_b \vee x_b x_u \vee \kappa(u_n) \\ u_{bn} & \overset{\mu}{=} & \kappa(u_b) \end{array}\right. \tag{4.7}$$

über den Variablen der Kripke-Struktur $\mathcal{K}_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}$ ist, die mit Hilfe der Formeln

$$\left[\begin{array}{lcl} u_{bn}^{i_1+1} & := & \kappa\left(u_b^{i_1,\infty}\right) \\ u_b^{i_1,i_2+1} & := & \Diamond u_b^{i_1,i_2} \vee x_b x_u \vee \kappa\left(u_n^{i_1,0,\infty}\right) \\ u_n^{i_1,i_2,i_3+1} & := & \Box u_n^{i_1,i_2,i_3} \wedge \bar{x}_m\bar{x}_u \vee u_{bn}^{i_1} \end{array}\right. \tag{4.8}$$

innerhalb der Zeit $O\left(\left|Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}\right|^2|\Sigma|\right)$ berechnet werden kann. Ist $M_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}} = Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}$, reduziert sich die Komplexitätsgrenze auf $O\left(\left|Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}\right||\Sigma|\right)$.

Beweis: Wie der Abhängigkeitsgraph in Abbildung 4.6 zeigt, besteht das Gleichungssystem 4.7 aus einer einzigen maximalen Komponente. Es kann deshalb mit Algorithmus 3.57 berechnet werden, was zu den Iterationsformeln

$$\left[\begin{array}{lcl} u_{bn}^{i_1+1} & := & \kappa\left(u_b^{i_1,\infty}\right) \\ u_b^{i_1,i_2+1} & := & \Diamond u_b^{i_1,i_2} \vee x_b x_u \vee \kappa\left(u_n^{i_1,i_2,\infty}\right) \\ u_n^{i_1,i_2,i_3+1} & := & \Box u_n^{i_1,i_2,i_3} \wedge \overline{x}_m \overline{x}_u \vee u_{bn}^{i_1} \end{array}\right.$$

führt. Die erste und die dritte dieser Formeln sind bereits in den Formeln 4.8 enthalten, die zweite weicht jedoch von diesen ab. Der Unterschied liegt in dem zweiten Iterationszähler der Variablen u_n. Während es im allgemein Fall notwendig ist, für jeden Schritt $u_b^{i_1,i_2+1}$ einen neuen Fixpunkt $u_n^{i_1,i_2,\infty}$ zu berechnen, bleibt der Zähler i_2 in den Formeln 4.8 auf 0 stehen. Dies hat zur Folge, dass nur dann ein neuer Fixpunkt für u_n berechnet wird, wenn sich der Iterationszähler i_1 erhöht. Bei der Berechnung eines Fixpunktes für u_b werden folglich die Iterationsschritte so ausgeführt, als gäbe es keinen tiefer verschachtelten Fixpunkt zu berechnen. Diese Vereinfachung wird durch den Abhängigkeitsgraphen in Abbildung 4.6 gerechtfertigt. Aus diesem ist ersichtlich, dass eine neue Berechnung von u_n nach jedem Schritt von u_b nichts ändern würde.

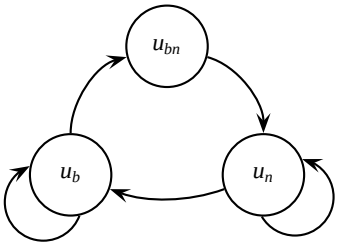


Abbildung 4.6: Der Abhängigkeitsgraph des Gleichungssystems 4.7

Folgende Beobachtungen zeigen, dass die Formeln 4.8 zu den gleichen Berechnungen führen wie Algorithmus 2.22:

1. Der erste Fixpunkt, der für die Lösung des Gleichungssystems berechnet wird, ist $u_n^{0,0,\infty}$. Da der Anfangswert u_{bn}^0 gleich 0 ist, verhält sich die Gleichung für $u_n^{0,0,i_3+1}$ wie Gleichung 4.3, wenn dort u_b gleich 0 ist. Wie auf Seite 67 erläutert, werden damit auf der rechten Hälfte der Kripke-Struktur die nicht co-erreichbaren Zustände gesammelt. Folglich entspricht $u_n^{0,0,\infty}$ der Menge der nicht co-erreichbaren Zustände, mit der Algorithmus 2.22 in Schritt 2.2 die Menge Q_b initialisiert.

2. Durch den Aufbau der Kripke-Struktur $\mathcal{K}_{\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}}$ entspricht der Ausdruck $x_b x_u$ der Menge $\left\{(p,q) \in Q_\mathcal{S} \mid \mathsf{act}_{\mathcal{A}_\mathcal{P}}(p) \cap \Sigma_u \nsubseteq \mathsf{act}_{\mathcal{A}_\mathcal{S}}((p,q))\right\}$, die der Algorithmus in Schritt 3.1 ermittelt. Diese Menge wird der Menge Q_b hinzugefügt, die zu diesem Zeitpunkt die nicht co-erreichbaren Zustände aus Schritt 2.2 enthält. Auf der Kripke-Struktur entspricht dies der Operation $x_b x_u \vee \kappa\left(u_n^{0,0,\infty}\right)$. Diese kommt in der ersten Berechnung in Schritt 4.1 vor: Dort ist $u_b^{0,0}$ gleich 0, so dass $u_b^{0,1} = x_b x_u \vee \kappa\left(u_n^{0,0,\infty}\right)$. Die weiteren Iterationsschritte für die Berechnung von $u_b^{0,\infty}$ entsprechen den Schritten 4.1 und 4.2 in Algorithmus 2.22 (s. Erläuterung zu Gleichung 4.1 auf Seite 66), so dass der Fixpunkt $u_b^{0,\infty}$ der Menge Q_b am Ende von Schritt 4 in der ersten Iteration des Algorithmus entspricht. Damit enthält $u_b^{0,\infty}$ sowohl die nicht co-erreichbaren als auch die verbotenen Zustände, die bisher gesammelt wurden.

3. In Schritt 3.2 bricht der Algorithmus vorzeitig ab, falls $Q_b = \emptyset$. Diese Überprüfung ist jedoch nicht zwingend, zumal der Algorithmus auch korrekt arbeitet, wenn die Schritte 4 und 5 trotzdem durchlaufen werden. Es wird dann in Schritt 6 mit demselben Ergebnis wie in Schritt 3.2 abgebrochen. Ein solcher Abbruch ist bei der Lösung des Gleichungssystems nicht vorgesehen und auch nicht zulässig, zumal die Mengenvereinigung aus Schritt 3.1 erst in der Berechnung von $u_b^{0,1}$ stattfindet. Die Lösung des Gleichungssystems ist trotzdem in jedem Fall korrekt, solange die weiteren Berechnungen mit denen des Algorithmus übereinstimmen.

4. Es folgt die Berechnung $u_{bn}^1 := \kappa\left(u_b^{0,\infty}\right)$. Dabei werden lediglich die bisher gefundenen Zustände auf die rechte Hälfte der Kripke-Struktur gespiegelt und auf die Variable u_{bn} übertragen, so dass dieser Schritt keinerlei Berechnungen des Algorithmus entspricht.

5. Als nächstes berechnet der Algorithmus in Schritt 5 die nicht co-erreichbaren Zustände des Automaten $\mathcal{A}_{\mathcal{S}}|_{\bar{Q}_b}$. In Schritt 6 werden diese Zustände – sofern nicht abgebrochen wird – der Menge Q_b hinzugefügt, so dass diese bei dem Rücksprung auf Schritt 4 alle Zustände enthält, die nicht co-erreichbar sind oder bereits als verboten erkannt wurden. Wie auf Seite 67 erläutert, werden diese Zustände auf der rechten Hälfte der Kripke-Struktur mit Hilfe der Gleichung 4.3 ermittelt. Bei der Lösung des Gleichungssystems steht nun die Berechnung des nächsten Fixpunktes $u_n^{1,0,\infty}$ an. Ein Vergleich mit den Gleichungen 4.8 zeigt, dass dabei wie bei der Lösung von Gleichung 4.3 gerechnet wird, und dass dort der passende Wert u_{bn}^1 eingesetzt wird. Dies gilt auch für die weiteren Iterationen, so dass $u_n^{i_1,0,\infty}$ stets der Zustandsmenge Q_b entspricht, die der Algorithmus in Schritt 6 ermittelt, bevor auf Schritt 4 zurückgesprungen wird.

6. Von nun an wechseln sich die Berechnungen der Gleichungen ab. Dies entspricht auch dem Verlauf des Algorithmus, in dem fortan die Schritte 4 bis 6 wiederholt werden.

7. Der Algorithmus terminiert, wenn in Schritt 5 keine neuen nicht co-erreichbaren Zustände gefunden werden. Nach Algorithmus 3.57 ist das Abbruchkriterium für die Formeln 4.8 das Erreichen des Fixpunktes u_{bn}^∞. Die Äquivalenz dieser beiden Kriterien wird in zwei Schritten gezeigt. Der erste ist trivial: Ist der Fixpunkt u_{bn}^∞ erreicht, ändern sich auch die anderen Fixpunkte nicht mehr, und es können keine neuen nicht co-erreichbaren Zustände gefunden werden. Für die umgekehrte Richtung muss zuerst bemerkt werden, dass der Fixpunkt $u_n^{0,0,\infty}$ nicht Schritt 5, sondern Schritt 2.1 entspricht. Sei deshalb $i_1 \geq 1$ der Wert, für den gilt, dass $u_n^{i_1,0,\infty}$ keine neuen nicht co-erreichbaren Zustände in Schritt 5 findet. Dann ist $u_n^{i_1,0,\infty} = u_{bn}^{i_1} = \kappa\left(u_b^{i_1-1,\infty}\right)$. Wird dieser Wert in die Gleichung für u_b eingesetzt, entsteht die Gleichung $u_b^{i_1,i_2+1} := \Diamond u_b^{i_1,i_2} \vee x_b x_u \vee u_b^{i_1-1,\infty}$. Da der Fixpunkt $u_b^{i_1-1,\infty}$ die Menge $x_b x_u$ enthält, ist $u_b^{i_1,\infty} = u_b^{i_1-1,\infty}$. Daraus folgt $\kappa\left(u_b^{i_1,\infty}\right) = \kappa\left(u_b^{i_1-1,\infty}\right)$, d.h. $u_{bn}^{i_1+1} = u_{bn}^{i_1} = u_{bn}^\infty$.

Damit entspricht u_{bn}^∞ der Menge Q_b, wenn Algorithmus 2.22 terminiert, und es gilt $Q_b = \left[\!\left[\exists x_u.u_{bn}^\infty\right]\!\right]_{\mathcal{A}_{\mathcal{S}}}$. Die Komplexitätsanalyse ist der in dem Beweis von Theorem 4.8 ähnlich. □

Korollar 4.12 *Um Iterationen zu sparen, kann die Berechnung der Formeln 4.8 bereits dann abgebrochen werden, wenn für* $i_1 \geq 1$ $u_n^{i_1,0,\infty} = \kappa\left(u_b^{i_1-1,\infty}\right)$. *Es gilt dann* $u_{bn}^\infty = u_n^{i_1,0,\infty}$.

Beweis: Der Beweis ist in dem Beweis von Theorem 4.11 unter Punkt 7 enthalten. □

Beispiel 4.13 Zum Vergleich mit der Lösung nach Theorem 4.8 wird Beispiel 4.10 mit Theorem 4.11 berechnet. Die dazugehörige Kripke-Struktur wurde bereits in Beispiel 4.5 erstellt und ist auf Abbildung 4.3 zu sehen. Es folgen die Iterationsschritte der Formeln 4.8. Um die Lesbarkeit zu erhöhen, werden die Mengen nicht in symbolischer Darstellung, sondern explizit angegeben.

$$
\begin{aligned}
u_{bn}^0 &= \{\} \\
u_b^{0,0} &= \{\} \\
u_n^{0,0,0} &= \mathcal{S} \\
u_n^{0,0,1} &= \{(1,1),(2,1),(3,1),(4,1),(5,1),(6,1),(7,1)\} \\
u_n^{0,0,2} &= \{(1,1),(2,1),(3,1),(4,1),(5,1),(7,1)\} \\
u_n^{0,0,3} &= \{(1,1),(2,1),(3,1),(5,1),(7,1)\} \\
u_n^{0,0,4} &= \{(1,1),(3,1),(5,1),(7,1)\} \\
u_n^{0,0,5} &= \{(3,1),(5,1),(7,1)\} \\
u_n^{0,0,6} &= \{(3,1),(5,1)\} \\
u_n^{0,0,7} &= \{(3,1)\}
\end{aligned}
$$

$$
\begin{aligned}
u_n^{0,0,8} &= \{\} \\
u_n^{0,0,\infty} &= \{\} \\
u_b^{0,1} &= \{\} \\
u_b^{0,2} &= \{(5,0)\} \\
u_b^{0,3} &= \{(3,0),(5,0)\} \\
u_b^{0,\infty} &= \{(3,0),(5,0)\} \\
u_{bn}^{1} &= \{(3,1),(5,1)\} \\
u_b^{1,0} &= \{(3,0),(5,0)\} \\
u_n^{1,0,0} &= \mathcal{S} \\
u_n^{1,0,1} &= \{(1,1),(2,1),(3,1),(4,1),(5,1),(6,1),(7,1)\} \\
u_n^{1,0,2} &= \{(1,1),(2,1),(3,1),(4,1),(5,1),(7,1)\} \\
u_n^{1,0,3} &= \{(1,1),(2,1),(3,1),(5,1),(7,1)\} \\
u_n^{1,0,4} &= \{(1,1),(3,1),(5,1),(7,1)\} \\
u_n^{1,0,5} &= \{(3,1),(5,1),(7,1)\} \\
u_n^{1,0,6} &= \{(3,1),(5,1)\} \\
u_n^{1,0,\infty} &= \{(3,1),(5,1\}.
\end{aligned}
$$

Das Haltekriterium aus Korollar 4.12, nämlich $u_n^{i_1,0,\infty} = \kappa\left(u_b^{i_1-1,\infty}\right)$, ist für $i_1 = 1$ erfüllt. Deshalb gilt $u_{bn}^{\infty} = u_n^{1,0,\infty} = \{(3,1),(5,1)\}$. Die Abbildung dieser Menge auf den Automaten $\mathcal{A}_{\mathcal{P}} \times \mathcal{A}_{\mathcal{E}}$ ist die Menge $\{3,5\}$. Der Überwacher – die Einschränkung von $\mathcal{A}_{\mathcal{P}} \times \mathcal{A}_{\mathcal{E}}$ auf das Komplement dieser Menge – ist in Abbildung 4.5 zu sehen und bestätigt das Ergebnis des vorangegangenen Beispiels. □

4.2.3 Gute statt verbotene Zustände berechnen

Die Gleichungssysteme 4.5 und 4.7 berechnen nach den Algorithmen 2.20 bzw. 2.22 die Menge der Zustände, die *nicht* Teil des gesuchten Überwachers sind, weshalb für die Lösung des Syntheseproblems das Komplement dieser Menge gebildet werden muss. Dieser letzte Schritt kann leicht in die Gleichungssysteme integriert werden, indem die letzte Gleichung – und infolgedessen auch die zwei anderen – unter Anwendung von Lemma 3.29 negiert werden. Dann ist die Lösung des Gleichungssystems bereits die Zustandsmenge des Überwachers. Im Folgenden wird diese Transformation für Gleichungssystem 4.7 vorgenommen. Wie sich in Kapitel 6 zeigen wird, bildet das dadurch gewonnene Gleichungssystem 4.9 den Ausgangspunkt für die dort vorgenommene Verallgemeinerung der Überwachersynthese.

Für die Bezeichnung der komplementären Mengen werden die Variablen $u_c := \bar{u}_n$, $u_g := \bar{u}_b$ und $u_{cg} := \bar{u}_{bn}$ eingeführt. Die Zustände in u_c sind die co-erreichbaren, die in u_g die nicht verbotenen. Diese werden fortan als *gute* Zustände bezeichnet. Schließlich stellt u_{cg} die gleichzeitig co-erreichbaren und guten Zustände dar.

Korollar 4.14 (Überwachersynthese im μ-Kalkül mit komplementären Mengen) *Gegeben sei ein Automat $\mathcal{A}_{\mathcal{P}} \times \mathcal{A}_{\mathcal{E}}$, der nach Definition 4.1 beschriftet ist und das gewünschte Verhalten für das System $\mathcal{A}_{\mathcal{P}}$ beschreibt. Sei $\mathcal{A}_{\mathcal{S}}|_{Q_{cg}}$ der gesuchte Überwacher, wobei $\mathcal{A}_{\mathcal{S}} := \mathcal{A}_{\mathcal{P}} \times \mathcal{A}_{\mathcal{E}}$. Dann ist $Q_{cg} = \left[\!\left[\exists x_u . u_{cg}^{\infty}\right]\!\right]_{\mathcal{A}_{\mathcal{S}}}$, wobei u_{cg}^{∞} die Lösung des Gleichungssystems*

$$
\left\{
\begin{array}{lcl}
u_c & \overset{\mu}{=} & (\Diamond u_c \vee x_m \bar{x}_u) \wedge u_{cg} \\
u_g & \overset{\nu}{=} & \Box u_g \wedge \bar{x}_b x_u \wedge \kappa(u_c) \\
u_{cg} & \overset{\nu}{=} & \kappa\left(u_g\right)
\end{array}
\right. \tag{4.9}
$$

über den Variablen der Kripke-Struktur $\mathcal{K}_{\mathcal{A}_{\mathcal{P}} \times \mathcal{A}_{\mathcal{E}}}$ ist, die mit Hilfe der Formeln

$$
\left[
\begin{array}{lcl}
u_{cg}^{i_1+1} & := & \kappa\left(u_g^{i_1,\infty}\right) \\
u_g^{i_1,i_2+1} & := & \Box u_g^{i_1,i_2} \wedge \bar{x}_b x_u \wedge \kappa\left(u_c^{i_1,0,\infty}\right) \\
u_c^{i_1,i_2,i_3+1} & := & (\Diamond u_c^{i_1,i_2,i_3} \vee x_m \bar{x}_u) \wedge u_{cg}^{i_1}
\end{array}
\right. \tag{4.10}
$$

innerhalb der Zeit $O\left(\left|Q_{\mathcal{A}_{\mathcal{P}} \times \mathcal{A}_{\mathcal{E}}}\right|^2 |\Sigma|\right)$ berechnet werden kann. Ist $M_{\mathcal{A}_{\mathcal{P}} \times \mathcal{A}_{\mathcal{E}}} = Q_{\mathcal{A}_{\mathcal{P}} \times \mathcal{A}_{\mathcal{E}}}$, reduziert sich die Komplexitätsgrenze auf $O\left(\left|Q_{\mathcal{A}_{\mathcal{P}} \times \mathcal{A}_{\mathcal{E}}}\right| |\Sigma|\right)$.

Beweis: Das Ergebnis folgt aus der Negation beider Seiten der Gleichungen des Gleichungssystems 4.7 unter Berücksichtigung zusätzlicher Konjunktionen mit x_u bzw. $\overline{x_u}$ um die Mengen auf die gewünschten Strukturhälften zu beschränken. □

Korollar 4.15 *Um Iterationen zu sparen, kann die Berechnung der Formeln 4.10 bereits dann abgebrochen werden, wenn für $i_1 \geq 1$ $u_c^{i_1,0,\infty} = \kappa\left(u_g^{i_1-1,\infty}\right)$. Es gilt dann $u_{cg}^{\infty} = u_c^{i_1,0,\infty}$.*

Beweis: Sei $i_1 \geq 1$ der Wert für den gilt, dass $u_c^{i_1,0,\infty} = \kappa\left(u_g^{i_1-1,\infty}\right)$. Dann ist $u_g^{i_1,1} = \Box 1 \wedge \overline{x_b} x_u \wedge u_g^{i_1-1,\infty} = u_g^{i_1-1,\infty}$ und somit ein Fixpunkt, woraus folgt $u_g^{i_1,\infty} = u_g^{i_1-1,\infty}$. Damit ist auch $u_{cg}^{\infty} = \kappa\left(u_g^{i_1-1,\infty}\right) = u_c^{i_1,0,\infty}$. □

Beispiel 4.16 Zum Vergleich mit den vorangegangenen Beispiel wird das Beispiel 4.10 mit Hilfe des Gleichungssystems 4.9 berechnet. Die dazugehörige Kripke-Struktur wurde bereits in Beispiel 4.5 erstellt und ist auf Abbildung 4.3 zu sehen. Es folgen die Iterationsschritte der Formeln 4.10. Um die Lesbarkeit zu erhöhen, werden die Mengen nicht in symbolischer Darstellung, sondern explizit angegeben.

$u_{cg}^{0} = \mathcal{S}$
- $u_g^{0,0} = \mathcal{S}$
  - $u_c^{0,0,0} = \{\}$
  - $u_c^{0,0,1} = \{(0,1)\}$
  - $u_c^{0,0,2} = \{(0,1),(6,1)\}$
  - $u_c^{0,0,3} = \{(0,1),(4,1),(6,1)\}$
  - $u_c^{0,0,4} = \{(0,1),(2,1),(4,1),(6,1)\}$
  - $u_c^{0,0,5} = \{(0,1),(1,1),(2,1),(4,1),(6,1)\}$
  - $u_c^{0,0,6} = \{(0,1),(1,1),(2,1),(4,1),(6,1),(7,1)\}$
  - $u_c^{0,0,7} = \{(0,1),(1,1),(2,1),(4,1),(5,1),(6,1),(7,1)\}$
  - $u_c^{0,0,8} = \{(0,1),(1,1),(2,1),(3,1),(4,1),(5,1),(6,1),(7,1)\}$
  - $u_c^{0,0,\infty} = \{(0,1),(1,1),(2,1),(3,1),(4,1),(5,1),(6,1),(7,1)\}$
- $u_g^{1,0} = \mathcal{S}$
- $u_g^{0,2} = \{(0,0),(1,0),(2,0),(3,0),(4,0),(6,0),(7,0)\}$
- $u_g^{0,3} = \{(0,0),(1,0),(2,0),(4,0),(6,0),(7,0)\}$
- $u_g^{0,\infty} = \{(0,0),(1,0),(2,0),(4,0),(6,0),(7,0)\}$

$u_{cg}^{1} = \{(0,1),(1,1),(2,1),(4,1),(6,1),(7,1)\}$
- $u_g^{1,0} = \{(0,0),(1,0),(2,0),(4,0),(6,0),(7,0)\}$
  - $u_c^{1,0,0} = \{\}$
  - $u_c^{1,0,1} = \{(0,1)\}$
  - $u_c^{1,0,2} = \{(0,1),(6,1)\}$
  - $u_c^{1,0,3} = \{(0,1),(4,1),(6,1)\}$
  - $u_c^{1,0,4} = \{(0,1),(2,1),(4,1),(6,1)\}$
  - $u_c^{1,0,5} = \{(0,1),(1,1),(2,1),(4,1),(6,1)\}$
  - $u_c^{1,0,6} = \{(0,1),(1,1),(2,1),(4,1),(6,1),(7,1)\}$
  - $u_c^{1,0,\infty} = \{(0,1),(1,1),(2,1),(4,1),(6,1),(7,1)\}$.

Das Haltekriterium aus Korollar 4.15, nämlich $u_c^{i_1,0,\infty} = \kappa\left(u_g^{i_1-1,\infty}\right)$, ist für $i_1 = 1$ erfüllt. Deshalb gilt $u_{cg}^{\infty} = u_c^{1,0,\infty} = \{(0,1),(1,1),(2,1),(4,1),(6,1),(7,1)\}$. Die Abbildung dieser Menge auf den Automaten $\mathcal{A}_{\mathcal{P}} \times \mathcal{A}_{\mathcal{E}}$ ist die Menge $\{0,1,2,4,6,7\}$, was dem Überwacher in Abbildung 4.5 entspricht. □

4.3 Verwendung der erzielten Ergebnisse

Die Ergebnisse aus diesem Kapitel dienen als Grundlage für die Weiterentwicklung der Überwachersynthese in zwei unterschiedliche Richtungen. In Kapitel 5 werden sie zur Verringerung der Komplexität der Überwachersynthese herangezogen. In Kapitel 6 dienen sie als Startpunkt für eine Verallgemeinerung der Überwachersynthese, die neben den bisher erlaubten Sicherheits- und Lebendigkeitseigenschaften auch Fortdauer- und Fairnesseigenschaften zulässt. Beide Kapitel können weitestgehend unabhängig voneinander gelesen werden. Die Ausnahme hierzu bildet Abschnitt 6.5, der das Beispiel in Abschnitt 5.4 nochmals aufgreift und dessen Kenntnis voraussetzt. Es ist ebenfalls möglich, die Theorie in Kapitel 5 zunächst zu überspringen und die Lektüre bei den Anwendungsbeispielen in den Abschnitten 5.4 und 5.5 fortzusetzen.

Kapitel 5

Ein besserer Synthesealgorithmus

Die Komplexitätsanalyse in Abschnitt 2.5.4 zeigt, dass das Überwachersyntheseproblem im Allgemeinen innerhalb der Zeit $O\left(\left|Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}\right|^2 |\Sigma|\right)$ gelöst werden kann. Diese obere Grenze gilt sowohl für Algorithmus 2.18 (Wonham und Ramadge) als auch für Algorithmus 2.20 (Kumar und Garg) und dessen Variante (Algorithmus 2.22). Darüber hinaus sind die zwei letzteren in der Lage, in dem Sonderfall $M_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}} = Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}$ das Problem innerhalb der Zeit $O\left(\left|Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}\right| |\Sigma|\right)$ zu lösen. Allerdings kommt dieser Fall in der Praxis äußerst selten vor.

In diesem Kapitel wird eine besondere Problemklasse erkannt, bei denen die Kripke-Struktur $\mathcal{K}_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}$ frei von *verborgenen Verklemmungskomponenten (VVK)* ist, die ebenfalls im Laufe des Kapitels definiert werden. Probleme, bei denen $\mathcal{K}_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}$ keine solche Komponenten enthält, werden *VVK-frei* genannt. Dies ist immer der Fall, wenn $M_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}} = Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}$, aber auch ein Teil der Probleme, bei denen $M_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}$ echt in $Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}$ enthalten ist, ist VVK-frei. Der Zusammenhang zwischen den genannten Problemklassen ist in Abbildung 5.1 dargestellt (vgl. Abbildung 2.20 auf Seite 28). Aus Platzgründen stehen hier M und Q für $M_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}$ bzw. $Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}$.

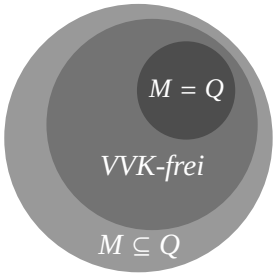


Abbildung 5.1: Die neue Klasse der VVK-freien Probleme

Es wird ein neuer Algorithmus vorgestellt, der das Überwachersyntheseproblem für alle VVK-freien Probleme mit linearer Komplexität lösen kann, während dies bisher nur im Sonderfall $M_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}} = Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}$ möglich war. Der Algorithmus ist aber nicht nur auf die neue Problemklasse ausgelegt, sondern kann auch den allgemeinen Fall effizienter als die bisherigen lösen, auch wenn hier die obere Grenze für die Komplexität bei $O\left(\left|Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}\right|^2 |\Sigma|\right)$ bleibt. Die höhere Effizienz ist auf die Tatsache zurückzuführen, dass der neue Algorithmus die Berechnungen, die in den herkömmlichen Algorithmen zu der quadratischen Laufzeit führen, nur noch für die Suche nach verborgenen Verklemmungskomponenten verwendet. Da die Anzahl dieser in der Regel viel kleiner als die Anzahl der Zustände ist, verhält sich die Laufzeit eher linear als quadratisch. Der Anwender muss nicht im Voraus wissen, um welchen Fall es sich bei seinem Problem handelt. Der neue Algorithmus wird immer in optimaler Zeit fertig.

Abschnitt 5.1 beginnt mit einer intuitiven Darstellung der neuen Berechnungsmethode, die in Abschnitt 5.2 formalisiert wird. Die Komplexität des neuen Ansatzes wird in Abschnitt 5.3 diskutiert. Darüber hinaus enthalten die Abschnitte 5.4 und 5.5 zwei größere Synthesebeispiele. Das erste ist eine Anwendung der Überwachersynthese zur Steuerung einer Telefonnummernauskunftzentrale, die von Seidl et al. [77] entwickelt wurde. Deren

Modellierung wird im Detail analysiert und die Syntheseergebnisse werden mit den hier entwickelten Methoden bestätigt. Darüber hinaus wird eine subtile Ungereimtheit in der beschriebenen Lösung festgestellt. Mit dem gegebenen Systemmodell lässt sich diese allerdings nicht mit den bisher vorgestellten Synthesemethoden beseitigen, weil die Spezifikation dazu um eine Fairnessbedingung erweitert werden muss. Dies wird jedoch in Kapitel 6 ermöglicht, so dass das Beispiel dort wieder aufgegriffen und entsprechend gelöst werden kann. Das zweite Beispiel handelt von der Synthese einer Gewinnstrategie für ein Zweipersonenspiel. Aus theoretischer Sicht kommt die Aufgabe, ein reaktives System zu steuern, der Anwendung einer Gewinnstrategie für ein Spiel gleich. Zweipersonenspiele sind deshalb Metaphern für reaktive Systeme, und die Ermittlung einer Gewinnstrategie ist in diesem Sinne mit der Synthese einer Steuerung gleichzusetzen. Das verwendete Nim-Spiel hat den Vorteil, dass es sich leicht skalieren lässt und somit eine ganze Testreihe liefert, mit der die Überlegenheit der Ergebnisse dieser Arbeit über die herkömmlichen Methoden belegt werden kann. Die Ergebnisse aus diesem Kapitel wurden in [94] veröffentlicht.

5.1 Die Berechnung co-erreichbarer Zustände

In Kapitel 4 wurde die Berechnung von Zustandsmengen eines Automaten auf die dazugehörige Kripke-Struktur übertragen. Sei insbesondere Q_b eine Menge, die auf der linken Hälfte der Kripke-Struktur durch den Ausdruck u_b dargestellt wird. Dann kann die Menge der Zustände, die nach dem Entfernen der Zustände in Q_b co-erreichbar bleiben, mit Hilfe von Gleichung 4.3 berechnet werden. Diese wird hier als Gleichung 5.1 wiedergegeben:

$$u_n \stackrel{\nu}{=} \Box u_n \wedge \overline{x}_m \overline{x}_u \vee \kappa(u_b)\,. \tag{5.1}$$

Diese Gleichung ist auch Teil der Gleichungssysteme 4.5 und (leicht verändert) 4.7. Während es sich dabei um einen größten Fixpunkt handelt, sind die beiden anderen Gleichungen in diesen Gleichungssystemen kleinste Fixpunkte, wodurch sich die Alternierungstiefen 3 bzw. 2 ergeben. In Abschnitt 3.3.3 wurde gezeigt, dass die Komplexität der Lösung eines Gleichungssystems von seiner Alternierungstiefe abhängt. Folglich muss es möglich sein, die Komplexität der Überwachersynthese zu verringern, wenn es gelingt, die nicht co-erreichbaren Zustände mit Hilfe eines kleinsten statt eines größten Fixpunktes zu berechnen. Im Folgenden wird eine neue Gleichung hergeleitet, die in der Lage ist, durch die Berechnung eines kleinsten Fixpunktes zumindest einen Teil der nicht co-erreichbaren Zustände zu sammeln. Das Ziel ist es, eines der Gleichungssysteme aus Kapitel 4 mit der neuen Gleichung zu ergänzen, um damit zu einer effizienteren Lösung des Überwachersyntheseproblems zu gelangen. Das gewünschte Ergebnis ist von beiden Gleichungssystemen aus zu erreichen. Im Folgenden wird dafür Gleichungssystem 4.5 gewählt, das hier unter der Nummer 5.2 wiedergegeben wird:

$$\begin{cases} u_b & \stackrel{\mu}{=} & \Diamond u_b \vee x_b x_u \vee u_{bn} \\ u_n & \stackrel{\nu}{=} & \Box u_n \wedge \overline{x}_m \overline{x}_u \vee \kappa(u_b) \\ u_{bn} & \stackrel{\mu}{=} & \kappa(u_n)\,. \end{cases} \tag{5.2}$$

Die neue Gleichung für die nicht co-erreichbaren Zustände entsteht aus der Überlegung, dass ein Zustand nicht co-erreichbar ist, wenn er selbst kein markierter Zustand ist und alle von ihm ausgehenden Ereignisse in bereits verbotene Zustände führen. Dies entspricht der Gleichung

$$u_v \stackrel{\mu}{=} \Box(u_v \vee \kappa(u_b)) \wedge \overline{x}_m \overline{x}_u, \tag{5.3}$$

wobei u_b die Menge der bereits verbotenen Zustände darstellt und wie üblich durch die Funktion $\kappa(\cdot)$ von der linken auf die rechte Hälfte der Kripke-Struktur gespiegelt wird. Gleichung 5.3 liefert die universellen Vorgänger der verbotenen Zustände. Damit die Fixpunkte so schnell wie möglich erreicht werden, ist es nützlich, die verbotenen Zustände selbst in das Ergebnis einzubeziehen. Dies wird durch eine Disjunktion der obigen Gleichung mit dem Ausdruck $\kappa(u_b)$ erreicht. Dann aber muss die Disjunktion mit $\kappa(u_b)$ innerhalb des $\Box$-Operators nicht mehr angegeben werden, denn sie kommt ohnehin in dem ersten Iterationsschritt vor. Daraus ergibt sich die Gleichung

$$u_v \stackrel{\mu}{=} \Box u_v \wedge \overline{x}_m \overline{x}_u \vee \kappa(u_b) \tag{5.4}$$

für die Menge der Zustände, die mit der neuen Methode gefunden werden oder bereits verboten sind.

Gleichung 5.4 läuft per Konstruktion keine Gefahr, co-erreichbare Zustände fälschlicherweise als nicht co-erreichbar einzustufen. Wie folgendes Gegenbeispiel zeigt, werden jedoch nicht unbedingt alle nicht co-erreichbaren Zustände entdeckt.

Beispiel 5.1 Sei die Struktur in Abbildung 5.2 die rechte Hälfte einer Kripke-Struktur, die aus der Übersetzung eines Automaten entstanden ist. Aus der Beschriftung der Zustände ist ersichtlich, dass Zustand 5 ein verbotener Zustand ist, so dass $u_b^0 = x_2\overline{x_1}x_0x_bx_u$, was Zustand 5 auf der linken Hälfte der Kripke-Struktur entspricht.

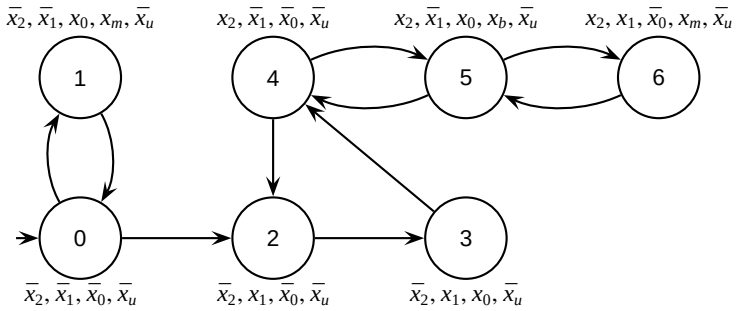


Abbildung 5.2: Rechte Hälfte der Kripke-Struktur für Beispiel 5.1

Für die Lösung von Gleichung 5.4 ergeben sich die Schritte[1] $u_v^0 = \{\}$, $u_v^1 = \{5\}$, $u_v^2 = \{5, 6\} = u_v^\infty$. Ohne die Zustände 5 und 6 sind die Zustände 2, 3, und 4 nicht co-erreichbar. Sie werden jedoch nicht in die Menge u_v^∞ aufgenommen. □

Es genügt also nicht, die zweite Gleichung in 5.2 durch Gleichung 5.4 zu ersetzen, wie in

$$\begin{cases} u_b & \overset{\mu}{=} & \Diamond u_b \vee x_b x_u \vee u_{bn} \\ u_v & \overset{\mu}{=} & \Box u_v \wedge \overline{x_m}\,\overline{x_u} \vee \kappa(u_b) \\ u_{bv} & \overset{\mu}{=} & \kappa(u_v)\,. \end{cases} \tag{5.5}$$

Falls die Transition zwischen von Zustand 4 nach Zustand 5 in Beispiel 5.1 nicht steuerbar ist, werden die fehlenden Zustände in weiteren Iterationen zwar noch erkannt, weil Zustand 4 verboten wird. Ist diese Transition jedoch steuerbar, werden die Zustände 2, 3 und 4 fälschlicherweise für co-erreichbar gehalten. Infolgedessen bleiben diese und eventuell noch andere verbotene Zustände unentdeckt.

Im Folgenden werden die Bedingungen analysiert, unter denen ein nicht co-erreichbarer Zustand während der Lösung von Gleichungssystem 5.5 unentdeckt bleibt. Dazu ist folgende Definition hilfreich:

Definition 5.2 (Verborgene Verklemmungskomponente (VVK)) *Gegeben sei ein Überwachersyntheseproblem mit den Automaten $\mathcal{A}_\mathcal{P}$ und $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$. Eine verborgene Verklemmungskomponente, abgekürzt VVK, ist eine Zustandsmenge der Kripke-Struktur $\mathcal{K}_{\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}}$, dessen Zustände folgende Bedingungen erfüllen:*

1. *sie sind nicht co-erreichbar;*
2. *während des Syntheseprozesses werden von ihnen keine nicht steuerbaren Transitionen gestrichen;*
3. *sie sind Teil einer schwach zusammenhängenden Komponente[2], in der jeder Zustand mindestens einen Nachfolger hat, der ebenfalls in dieser Komponente liegt.*

Eine Kripke-Struktur, in der solche Komponenten nicht auftreten, wird VVK-frei genannt.

Beispiel 5.3 Die Zustände 2, 3 und 4 in Abbildung 5.2 bilden eine verborgene Verklemmungskomponente. □

Lemma 5.4 (Unerkannte nicht co-erreichbare Zustände) *Ein nicht co-erreichbarer Zustand bleibt während der Lösung des Gleichungssystems 5.5 genau dann unentdeckt, wenn er einer verborgenen Verklemmungskomponente angehört.*

[1] Der besseren Lesbarkeit halber werden die Mengen nicht symbolisch, sondern explizit dargestellt.

[2] In einem gerichteten Graphen ist eine schwach zusammenhängende Komponente ein Teilgraph, in dem es für jedes Knotenpaar (u, v) einen Pfad von u nach v oder von v nach u gibt. Eine schwach zusammenhängende Komponente ist maximal, wenn sie kein echter Teilgraph einer anderen solchen Komponente ist. Verborgene Verklemmungskomponenten müssen jedoch nicht maximal sein.

Beweis: ($\rightarrow$): Sei p ein nicht co-erreichbarer Zustand, der während der Lösung von Gleichungssystem 5.5 unentdeckt bleibt. Dann ist Bedingung 1 in Definition 5.2 trivialerweise erfüllt. Bedingung 2 muss ebenfalls erfüllt sein, zumal p sich sonst als verboten herausstellen würde und die verbotenen Zustände Teil der Lösung des Gleichungssystems sind. Bedingung 3 wird wie folgt nachgewiesen: Es steht fest, dass p mindestens eine ausgehende Transition haben muss, denn sonst wäre er im ersten Iterationsschritt erkannt worden (s. Gleichung 3.13 auf Seite 39). Da p nicht von der ersten Gleichung in 5.2 erkannt wird und auch nicht markiert ist, muss mindestens eine Transition von p in einen Zustand q führen, der nicht co-erreichbar ist (ansonsten wäre p auch co-erreichbar) und ebenfalls unentdeckt bleibt (ansonsten würden alle ausgehenden Transitionen von p zu bereits entdeckten Zuständen führen, und p wäre erkannt worden). Um unentdeckt zu bleiben, muss q aber die gleichen Bedingungen erfüllen. Damit bilden p, q und all ihre Nachfolger eine schwach zusammenhängende Komponente mit der Besonderheit, dass jeder Zustand einen Nachfolger innerhalb dieser Komponente hat.

($\leftarrow$): Zustände einer verborgenen Verklemmungskomponente sind laut Definition 5.2 nicht co-erreichbar (Bedingung 1) und werden auch nicht verboten (Bedingung 2). Damit bleiben sie für die zweite Gleichung in 5.2 unerkannt. Da sie jeweils einen Nachfolger haben, der ebenfalls diese Bedingungen erfüllt (Bedingung 3), können sie auch nicht von der ersten Gleichung erkannt werden. □

5.2 Verbesserung der Überwachersynthese

Aus Lemma 5.4 folgt, dass VVK-freie Probleme mit Hilfe von Gleichungssystem 5.5 gelöst werden können. Da dieses Alternierungsfrei ist, ist auch die Komplexität geringer als bei den Lösungen nach den Theoremen 4.8 oder 4.11. Es lässt sich jedoch nur in besonderen Fällen voraussagen, ob ein Problem VVK-frei ist oder nicht. Dies ist z.B. der Fall, wenn alle Zustände markiert und damit co-erreichbar sind, oder wenn es im System keine Schleifen gibt, zumal eine VVK auf Grund der Bedingung 3 in Definition 5.2 mindestens eine Schleife haben muss. Um in jedem Fall einsetzbar zu sein, muss dieses Gleichungssystem folglich noch so ergänzt werden, dass auch bei nicht VVK-freien Problemen alle nicht co-erreichbaren Zustände gefunden werden. Dies kann mit Hilfe von Gleichung 5.1 gewährleistet werden, wenn an Stelle von u_b alle bisher gefundenen Zustände eingesetzt werden. Der Gedanke dabei ist, die Alternierung zwischen dem größten Fixpunkt in Gleichung 5.1 und den kleinsten Fixpunkten in den anderen Gleichungen in 5.5 so gering wie möglich zu halten. Folgendes Theorem stellt ein Gleichungssystem vor, das die effizientere Suche nach nicht co-erreichbaren Zuständen aus Gleichungssystem 5.5 nutzt und trotzdem die nicht co-erreichbaren Zustände in verborgenen Verklemmungskomponenten berücksichtigt.

Theorem 5.5 *Gegeben seien die Automaten $\mathcal{A}_\mathcal{P}$ und $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$, die ein System und dessen gewünschtes Verhalten beschreiben. Seien wie in Algorithmus 2.22 $\mathcal{A}_\mathcal{S} := \mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ und $\mathcal{A}_\mathcal{S}|_{\bar{Q}_b}$ der gesuchte Überwacher. Dann ist $Q_b = \left[\!\left[\exists x_u . u_{bn}^\infty\right]\!\right]_{\mathcal{A}_\mathcal{S}}$, wobei u_{bn}^∞ die Lösung des Gleichungssystems*

$$\left\{\begin{array}{lcl} u_n & \overset{\nu}{=} & \Box u_n \wedge \overline{x}_m \overline{x}_u \vee u_{bn} \\ u_b & \overset{\mu}{=} & \Diamond u_b \vee x_b x_u \vee \kappa(u_v \vee u_n) \\ u_v & \overset{\mu}{=} & \Box u_v \wedge \overline{x}_m \overline{x}_u \vee \kappa(u_b) \\ u_{bn} & \overset{\mu}{=} & u_v \end{array}\right. \tag{5.6}$$

über den Variablen der Kripke-Struktur $\mathcal{K}_{\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}}$ ist, die mit Hilfe der Formeln

$$\left[\begin{array}{lcl} u_{bn}^{i_1+1} & := & u_v^{i_1,\infty} \\ u_v^{i_1,i_2+1} & := & \Box u_v^{i_1,i_2} \wedge \overline{x}_m \overline{x}_u \vee \kappa\left(u_b^{i_1,i_2,\infty}\right) \\ u_b^{i_1,i_2,i_3+1} & := & \Diamond u_b^{i_1,i_2,i_3} \vee x_b x_u \vee \kappa\left(u_v^{i_1,i_2} \vee u_n^{i_1,0,0,\infty}\right) \\ u_n^{i_1,i_2,i_3,i_4+1} & := & \Box u_n^{i_1,i_2,i_3,i_4} \wedge \overline{x}_m \overline{x}_u \vee u_{bn}^{i_1} \end{array}\right. \tag{5.7}$$

innerhalb der Zeit $O\left(\left|Q_{\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}}\right|^2 |\Sigma|\right)$ berechnet werden kann. Ist $\mathcal{K}_{\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}}$ VVK-frei, kann die Lösung innerhalb der Zeit $O\left(\left|Q_{\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}}\right| |\Sigma|\right)$ berechnet werden.

Beweis: Wie der Abhängigkeitsgraph in Abbildung 4.4 zeigt, besteht das Gleichungssystem 5.6 aus einer einzigen maximalen Komponente. Es kann deshalb mit Algorithmus 3.57 berechnet werden, was zu den Iterationsformeln

$$\left[\begin{array}{lcl} u_{bn}^{i_1+1} & := & u_v^{i_1,\infty} \\ u_v^{i_1,i_2+1} & := & \Box u_v^{i_1,i_2} \wedge \overline{x}_m \overline{x}_u \vee \kappa\left(u_b^{i_1,i_2,\infty}\right) \\ u_b^{i_1,i_2,i_3+1} & := & \Diamond u_b^{i_1,i_2,i_3} \vee x_b x_u \vee \kappa\left(u_v^{i_1,i_2} \vee u_n^{i_1,i_2,i_3,\infty}\right) \\ u_n^{i_1,i_2,i_3,i_4+1} & := & \Box u_n^{i_1,i_2,i_3,i_4} \wedge \overline{x}_m \overline{x}_u \vee u_{bn}^{i_1} \end{array}\right.$$

führt. Diese Lösung unterscheidet sich in der dritten Formel von den Formeln 5.7. Der Unterschied liegt in den Iterationszählern der Variablen u_n. Während es im Allgemeinen notwendig ist, für jeden Schritt $u_b^{i_1,i_2,i_3+1}$ einen neuen Fixpunkt $u_n^{i_1,i_2,i_3,\infty}$ zu berechnen, bleiben die Zähler i_2 und i_3 in den Formeln 5.7 auf 0 stehen. Dies hat zur Folge, dass nur dann ein neuer Fixpunkt für u_n berechnet wird, wenn sich der Iterationszähler i_1 erhöht. Bei der Berechnung eines Fixpunktes für u_b werden folglich die Iterationsschritte so ausgeführt, als gäbe es keinen inneren Fixpunkt zu berechnen. Diese Vereinfachung wird durch den Abhängigkeitsgraphen in Abbildung 5.3 gerechtfertigt. Aus diesem ist ersichtlich, dass eine neue Berechnung von u_n nach jedem Schritt von u_b nichts ändern würde.

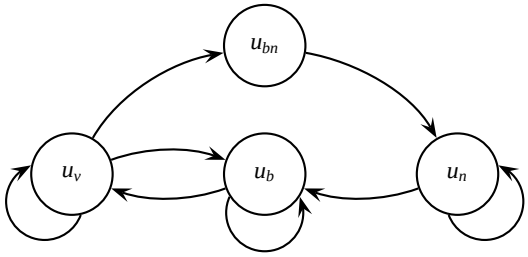


Abbildung 5.3: Der Abhängigkeitsgraph des Gleichungssystems 5.6

Die Korrektheit der Berechnungen kann wie folgt auf Theorem 4.11 zurückgeführt werden: Das Überwachersyntheseproblem kann mit Hilfe von Gleichungssystem 4.7 gelöst werden. Aus den Formeln 4.8 folgt, dass die Fixpunkte stets in der Reihenfolge u_n, u_b, u_{bn} berechnet werden, und dass u_b, wenn die Berechnung eines Fixpunktes für u_{bn} an der Reihe ist, die bis zu diesem Zeitpunkt verbotenen und nicht co-erreichbaren Zustände enthält. Das Gleichungssystem 5.6 unterscheidet sich von 4.7 durch die neue Gleichung für u_v und durch Änderungen an den Gleichungen für u_b und u_{bn}. Aus den Formeln 5.7 ergibt sich für die Berechnung der Fixpunkte die Reihenfolge: u_n, dann u_b in Wechselwirkung mit u_v, und schließlich u_{bn}. Die Gleichung für u_b in 5.6 ist eine Erweiterung der zweiten Gleichung in 4.7, die zusätzlich zu den nicht co-erreichbaren Zuständen in u_n auch die in u_v berücksichtigt. Wie auf Seite 78 erläutert, läuft die Gleichung für u_v keine Gefahr, einen Zustand, der co-erreichbar ist, fälschlicherweise als nicht co-erreichbar einzustufen. Damit bleiben auch die Berechnungen von u_b korrekt. Die Gleichung für u_v verlangt nach jedem Iterationsschritt einen neuen Fixpunkt für u_b. Dies bewirkt, dass ein Fixpunkt für u_v genau dann erreicht wird, wenn alle verbotenen und nicht co-erreichbaren Zustände mit Ausnahme der verborgenen Verklemmungskomponenten gefunden wurden. Insofern liegt der Unterschied zu den Berechnungen laut Theorem 4.11 lediglich darin, dass u_v im Allgemeinen eine größere Menge verbotener und nicht co-erreichbarer Zustände als dort u_b gesammelt hat, wenn die nächste Berechnung eines Fixpunktes für u_{bn} ansteht. Dies kann die Lösung aber nicht beeinträchtigen, da die gefundenen Zustände auf jeden Fall entfernt werden müssen und damit einen Prä-Fixpunkt der gesuchten Lösung bilden.

Für die Bestimmung der Komplexität seien $\mathcal{S}$ und $\mathcal{R}$ die Zustandsmenge bzw. die Transitionsrelation der Kripke-Struktur $\mathcal{K}_{\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}}$. Die Gleichung für u_{bn} kann in konstanter Zeit berechnet werden und erreicht in höchstens $|\mathcal{S}|$ Iterationen den Fixpunkt u_{bn}^∞. Die Gleichung für u_n wird genauso oft ausgewertet. Die Berechnung eines Fixpunktes $u_n^{i_1,0,0,\infty}$ kann innerhalb der Zeit $O(|\mathcal{R}|)$ durchgeführt werden, und bei $O(|\mathcal{S}|)$ Wiederholungen ergibt dies einen Aufwand, der in $O(|\mathcal{S}|\,|\mathcal{R}|)$ liegt.

Die Gleichungen für u_b und u_v haben beide den gleichen Fixpunkttyp. Damit werden die Berechnungen der aufeinander folgenden Fixpunkte laut Algorithmus 3.57 jeweils mit dem zuletzt ermittelten Fixpunkt gestartet. Dies führt dazu, dass alle Berechnungen dieser Gleichungen innerhalb $|\mathcal{S}|$ Schritten abgeschlossen werden können. Die Berechnung der Fixpunkte kann wie im Fall von u_n innerhalb der Zeit $O(|\mathcal{R}|)$ durchgeführt werden. Bei

$O(|\mathcal{S}|)$ Wiederholungen ergibt dies ebenfalls einen Aufwand, der in $O(|\mathcal{S}|\,|\mathcal{R}|)$ liegt. Damit ergibt sich $O(|\mathcal{S}|\,|\mathcal{R}|)$ als obere Grenze für die Lösung von Gleichungssystem 5.6. Da $|\mathcal{S}| = 2\left|Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}\right|$ und $|\mathcal{R}| \leq 2\left|Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}\right|\,|\Sigma|$ (siehe Definition 4.4), ergibt sich für den gesamten Algorithmus die Komplexität $O\left(\left|Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}\right|^2\,|\Sigma|\right)$.

Wenn $\mathcal{K}_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}$ darüber hinaus VVK-frei ist, werden alle nicht co-erreichbaren Zustände von der Gleichung für u_v gefunden. Folglich kommen durch die Berechnung des Fixpunktes $u_n^{1,0,0,\infty}$ keine Zustände mehr hinzu, und die anderen Fixpunkte können sich nicht mehr ändern. Somit können alle Berechnungen innerhalb der Zeit $O(|\mathcal{R}|) = O\left(\left|Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}\right|\,|\Sigma|\right)$ durchgeführt werden.

Lemma 5.6 *Die Berechnung der Formeln 5.7 kann bereits abgebrochen werden, wenn (1)* $u_b^{i_1,0,\infty} = \kappa\left(u_n^{i_1,0,0,\infty}\right)$ *oder (2) wenn* $i_1 \geq 1$ *und* $u_n^{i_1,0,0,\infty} = u_v^{i_1-1,\infty}$. *In beiden Fällen gilt dann* $u_{bn}^{\infty} = u_n^{i_1,0,0,\infty}$.

Beweis: (1) Sei i_1 der Wert, für den gilt, dass $u_b^{i_1,0,\infty} = \kappa\left(u_n^{i_1,0,0,\infty}\right)$. Aus der zweiten Formel in 5.7 folgt $u_v^{i_1,i_2+1} := \Box u_v^{i_1,i_2} \wedge \overline{x}_m\overline{x}_u \vee u_n^{i_1,0,0,\infty}$. Da aber $u_n^{i_1,0,0,\infty}$ bereits alle derzeit nicht co-erreichbaren Zustände gesammelt hat, kann diese Formel keine weiteren Zustände finden. Deshalb gilt $u_v^{i_1,\infty} = u_n^{i_1,0,0,\infty}$ und somit $u_{bn}^{i_1+1} = u_n^{i_1,0,0,\infty}$. In der nächsten Iteration ist deshalb $u_n^{i_1+1,0,0,\infty} = u_n^{i_1,0,0,\infty}$, so dass sich kein Fixpunkt mehr ändert. Damit ist $u_{bn}^{\infty} = u_{bn}^{i_1+1} = u_n^{i_1,0,0,\infty}$.

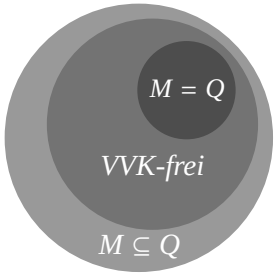
(2) Wenn $u_n^{i_1,0,0,\infty} = u_v^{i_1-1,\infty}$, dann konnte die Gleichung für u_n keine neuen nicht co-erreichbaren Zustände finden. Dies bedeutet, dass u_v bereits alle verbotenen und nicht co-erreichbaren Zustände enthält und sich deshalb nicht mehr ändern wird, also ist $u_v^{\infty,\infty} = u_v^{i_1-1,\infty} = u_n^{i_1,0,0,\infty}$ und folglich $u_{bn}^{\infty} = u_n^{i_1,0,0,\infty}$. □

5.3 Eine neue Problemklasse

Bisher wurden in dieser Arbeit vier Lösungen für das Überwachersyntheseproblem vorgestellt:

1. Der Ansatz von Wonham und Ramadge (Algorithmus 2.18).
2. Der Ansatz von Kumar und Garg (Algorithmus 2.20 bzw. Theorem 4.8).
3. Die Variante dieses Ansatzes (Algorithmus 2.22 bzw. Theorem 4.11).
4. Der neue Ansatz in Theorem 5.5.

Im Allgemeinen gilt für all diese Lösungen die obere Komplexitätsgrenze $O\left(\left|Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}\right|^2\,|\Sigma|\right)$. Für den Ansatz von Kumar und Garg und dessen Variante kann diese Grenze im Sonderfall $M_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}} = Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}$ auf $O\left(\left|Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}\right|\,|\Sigma|\right)$ herabgesenkt werden. Mit der Lösung aus Theorem 5.5 erstreckt sich dieser Vorteil auf alle VVK-freien Probleme – gemeint sind die Fälle, in denen die Kripke-Struktur $\mathcal{K}_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}$ VVK-frei ist. Die Klasse dieser Probleme enthält den Sonderfall $M_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}} = Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}$, zumal dort alle Zustände co-erreichbar sind und folglich keine verborgenen Verklemmungskomponenten entstehen können. Dieser Zusammenhang ist in Abbildung 5.4 festgehalten (vgl. Abbildung 2.20 auf Seite 28). Aus Platzgründen stehen M und Q für $M_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}$ bzw. $Q_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}$.



	Wonham und Ramadge	Kumar und Garg	Theorem 5.5
$M = Q$	$O\left(\lvert Q\rvert^2\,\lvert\Sigma\rvert\right)$	$O\left(\lvert Q\rvert\,\lvert\Sigma\rvert\right)$	$O\left(\lvert Q\rvert\,\lvert\Sigma\rvert\right)$
$M \subset Q$, *VVK-frei*	$O\left(\lvert Q\rvert^2\,\lvert\Sigma\rvert\right)$	$O\left(\lvert Q\rvert^2\,\lvert\Sigma\rvert\right)$	$O\left(\lvert Q\rvert\,\lvert\Sigma\rvert\right)$ ☜
Nicht VVK-frei	$O\left(\lvert Q\rvert^2\,\lvert\Sigma\rvert\right)$	$O\left(\lvert Q\rvert^2\,\lvert\Sigma\rvert\right)$	$O\left(\lvert Q\rvert^2\,\lvert\Sigma\rvert\right)$

Abbildung 5.4: Verringerung der Komplexität für VVK-freie Probleme durch Theorem 5.5

Die neue Lösung nach Theorem 5.5 ist aber auch für nicht VVK-freie Probleme effizienter als die bisher bekannten. Obwohl die obere Grenze für die Komplexität dann quadratisch bleibt, werden immer noch viele nicht co-erreichbare Zustände mit Hilfe der Gleichung für u_v in linearer Zeit gesammelt. Folglich findet die Gleichung für u_n nur dann neue Zustände, wenn verborgene Verklemmungskomponenten aufgedeckt werden. Die Anzahl solcher Komponenten ist in der Regel deutlich kleiner als $|\mathcal{S}|$, so dass in den meisten Fällen die Komplexität der neuen Lösung eher linear als quadratisch ausfällt. Die Methode hat darüber hinaus den Vorteil, dass der Anwender sich nicht um die Frage kümmern muss, ob sein Problem VVK-frei ist oder nicht. Der Algorithmus findet alle unerwünschten Zustände und setzt dazu automatisch die jeweils effizientere Gleichung ein.

In den nächsten Abschnitten werden zwei Beispiele diskutiert. Das erste beschreibt eine kommerzielle Anwendung der Überwachersynthese, die von der Münchner Firma Varetis AG zur Steuerung einer Telefonnummernauskunftzentrale entwickelt wurde. Das zweite Beispiel handelt von der Synthese einer Gewinnstrategie für ein Zweipersonenspiel. Das hier vorgestellte Spiel lässt sich einfach skalieren und erlaubt es dadurch, die Vorteile der symbolischen Darstellung und auch der neuen Methode aus Theorem 5.5 zu verdeutlichen.

5.4 Beispiel Telefonnummernauskunft

Die Telefonnummernauskunft wird durch große Auskunftzentralen, auch *call centers* genannt, realisiert. Dieser Abschnitt beschreibt eine von Seidl et al. [77] entwickelte Anwendung der Überwachersynthese zur Koordination der Komponenten einer solchen Zentrale. Zunächst werden ihre Berechnungen anhand der hier vorgestellten Methoden bestätigt. Darüber hinaus wird gezeigt, dass das herkömmliche Ramadge-Wonham-Modell in diesem Fall nicht in der Lage ist, ein wirklich verklemmungsfreies Verhalten des Systems zu gewährleisten. Dies gelingt jedoch mit Hilfe der verallgemeinerten Überwachersynthese in Kapitel 6, weshalb das Beispiel in Abschnitt 6.5 nochmals aufgegriffen und verbessert wird.

5.4.1 Systembeschreibung

Eine Auskunftzentrale besteht aus mehreren Arbeitsplätzen, an denen jeweils ein Bearbeiter externe Anrufe entgegennimmt. Die Anrufe aus dem öffentlichen Telefonnetz treffen wie in Abbildung 5.5 dargestellt über eine Nebenstellenanlage ein.

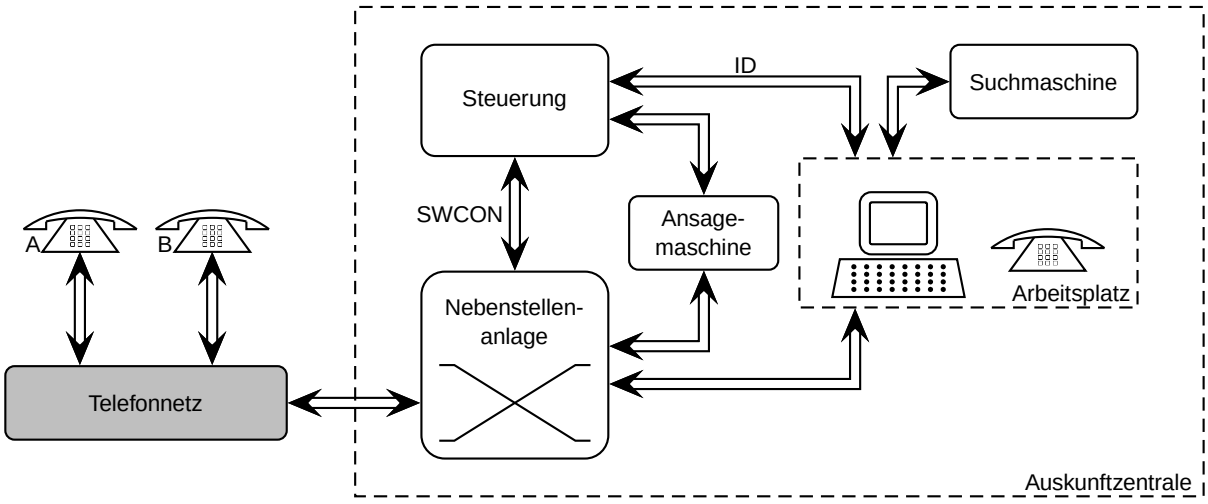


Abbildung 5.5: Aufbau der Telefonnummernauskunftzentrale

Der normale Ablauf besteht etwa aus einem Anruf eines Teilnehmers A, der die Rufnummer von B erfahren möchte und mit einem freien Bearbeiter verbunden wird. Gleichzeitig werden Daten wie die Rufnummer von

A und eventuell die Vorgeschichte des Anrufs dem Bearbeiter auf dem Bildschirm seines Arbeitsplatzes präsentiert. Dieser ist mit einer Suchmaschine verbunden, mit deren Hilfe die Rufnummer von B herauszufinden ist. Die gesuchte Nummer wird an eine Ansagemaschine weitergegeben, mit der A daraufhin verbunden wird. Davor kann A beispielsweise noch den Wunsch äußern, nach der Ansage mit B automatisch verbunden zu werden, oder auch die Nummer von B zusätzlich über andere Dienste (E-Mail, SMS) zu bekommen. Der Bearbeiter leitet dann die entsprechenden Maßnahmen ein, schaltet sich frei und wartet auf den nächsten Anruf.

Die Bearbeitung eines Anrufs kann aber auch deutlich von dem Normalfall abweichen. Der Teilnehmer A kann z.B. über die Tastatur seines Telefons während der automatischen Ansage das System veranlassen, ihn nochmals mit einem Bearbeiter zu verbinden, um weitere Dienste in Anspruch zu nehmen. In besonderen Fällen kann er in Wartestellung gebracht werden, während der Bearbeiter ein Rückfragegespräch mit seinem Gruppenleiter führt, der eventuell die Bearbeitung übernimmt. Andererseits kann der Gruppenleiter besetzt sein, wenn der Bearbeiter versucht, ihn anzurufen. A kann während des Wartens auflegen. Bei einer weiteren Ansage können alle Ausgänge der Ansagemaschine belegt sein. In mehreren Zuständen werden Zeitglieder gestartet, die in jedem anderen Zustand ablaufen können. Die Nebenläufigkeit mehrerer Prozesse führt in der Regel dazu, dass der konventionelle Systementwurf nicht mehr alle Möglichkeiten abdecken kann. Die Folge sind unerwartete Reaktionen des Systems wie z.B. Verklemmungen und verlorene oder falsche Verbindungen, die den Betrieb erheblich beeinträchtigen können. Große Auskunftzentralen besitzen hunderte oder gar tausende Arbeitsplätze und bearbeiten in den Spitzenzeiten mehrere zehntausend Anrufe pro Stunde. Die Fehlerrate muss also gering gehalten werden, um kritische oder gar katastrophale Situationen auszuschließen.

Außer den vermittlungstechnischen Aufgaben fallen auch Verwaltungsaufgaben an. Zum Beispiel muss die Nebenstellenanlage wissen, welche Arbeitsplätze zu jedem Zeitpunkt bedient werden. Dazu meldet sich jeder Bearbeiter zu Beginn seiner Schicht bei der Nebenstellenanlage an und am Ende wieder ab. Dies geschieht über An- und Abmeldeanforderungen seitens des Bearbeiters, die von der Nebenstellenanlage bestätigt werden. Weiterhin kann er für kurze Pausen zwischen den Zuständen „aktiv“ und „nicht aktiv“ hin- und herwechseln. Auch für diesen Fall sind Anforderungen seitens des Bearbeiters und die entsprechenden Bestätigungen seitens der Nebenstellenanlage vorgesehen. Erst wenn ein Bearbeiter angemeldet und aktiv ist, dürfen Gespräche von der Nebenstellenanlage zu ihm vermittelt werden.

In Anbetracht dieser Beschreibung ist es naheliegend, die gesamte Steuerung der Auskunftzentrale innerhalb der Nebenstellenanlage zu realisieren, zumal dort alle benötigten Informationen abrufbar sind. Dies ist prinzipiell richtig, jedoch werden für den Entwurf der entsprechenden Software in der Regel Methoden eingesetzt, die die o.g. Probleme nicht ausschließen können. Um eine korrekte Funktionalität zu gewährleisten, wurde die direkte Verbindung zwischen Nebenstellenanlage und Arbeitsplatz in Abbildung 5.5 aufgetrennt und zwischen beiden eine mit Hilfe der Überwachersynthese erstellte Steuerung eingefügt.

5.4.2 Modellierung

Aus der Sicht des Ramadge-Wonham-Modells besteht das zu steuernde System aus den Schnittstellen SWCON (*switch control*) und ID (*inquiry desk*), die von der Steuerung bedient werden. Der Zustandsraum bleibt trotz der vielen Arbeitsplätze verhältnismäßig klein, weil für jeden Anruf nur ein Arbeitsplatz berücksichtigt werden muss. Darüber hinaus stellt sich heraus, dass die Verwaltung der Arbeitsplätze, z.B. die An- und Abmeldung eines Bearbeiters, unabhängig von der Koordination des Verbindungsablaufs ist, weil die Ereignismengen beider Vorgänge disjunkt sind. Die Steuerung zerfällt deshalb in zwei Überwacher, die parallel und unabhängig voneinander laufen. In [77] beschreiben die Entwickler die Automaten für die Verwaltung der Arbeitsplätze im Detail, so dass sich die Berechnungen für diesen Überwacher nachvollziehen lässt. Die Automaten für die Steuerung des Verbindungsablaufs werden nicht vollständig wiedergegeben und deshalb hier nicht weiter behandelt.

Die Schnittstelle ID zwischen Steuerung und Arbeitsplatz wird durch den Automaten in Abbildung 5.6 beschrieben. Steuerbare Ereignisse (in der Abbildung kursiv dargestellt) bedeuten Befehle der Steuerung an den Arbeitsplatz, nicht steuerbare Ereignisse (nicht kursiv) hingegen Meldungen des letzteren an die Steuerung. Die Ereignisse sind wie folgt zu interpretieren:

- idLoginReq : Bearbeiter fordert Anmeldung an.
- idLogin : Bearbeiter wird angemeldet.
- idReadyReq : Bearbeiter fordert Aktivierung an.
- idReady : Bearbeiter wird in den aktiven Zustand versetzt.
- idNotReadyReq : Bearbeiter fordert Deaktivierung an.
- idNotReady : Bearbeiter wird in den nicht aktiven Zustand versetzt.
- idLogoutReq : Bearbeiter fordert Abmeldung an.
- idLogout : Bearbeiter wird abgemeldet.
- idEOS (End of Service): Bearbeiter fordert sofortige Abmeldung an.

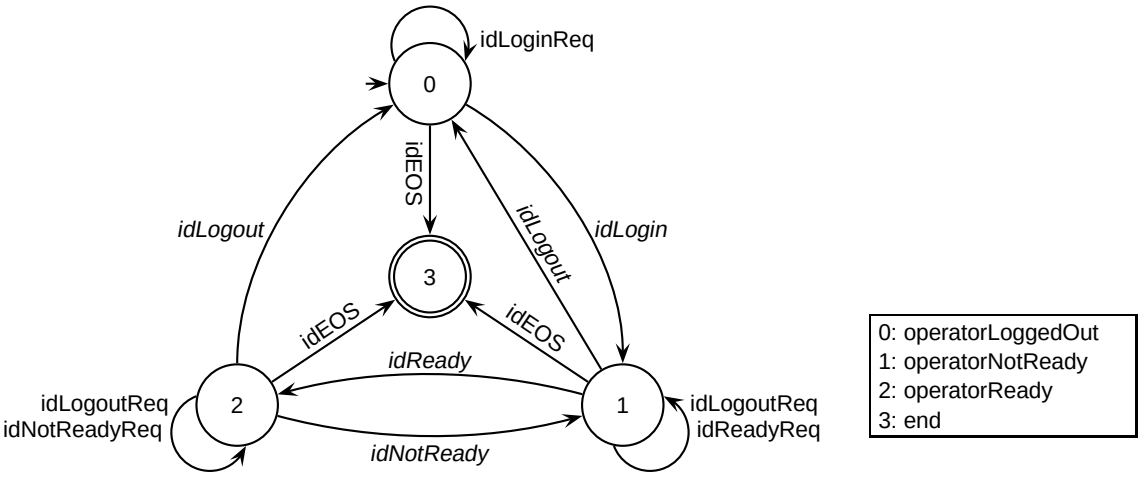


Abbildung 5.6: Die Schnittstelle ID zwischen Steuerung und Arbeitsplatz

Im Initialzustand ist der Bearbeiter nicht angemeldet. Die einzig mögliche Aktion seinerseits ist eine Anmeldeanforderung idLoginReq, die er z.B. durch das Drücken einer Taste erzeugen kann. Die Steuerung kann den Arbeitsplatz mit dem (steuerbaren) Ereignis idLogin freigeben und wartet dann im Zustand 1 (operatorNotReady) entweder auf eine Anforderung zur Aktivierung (idReadyReq) oder zur Abmeldung (idLogoutReq). Mögliche Reaktionen der Steuerung sind idLogout, gefolgt von einer Rückkehr in den Initialzustand, oder die Aktivierung mittels idReady, gefolgt von einem Wechsel in den Zustand 2 (operatorReady). Hier wird ähnlich wie in Zustand 1 verfahren. Darüber hinaus bietet das Ereignis idEOS in jedem Zustand die Möglichkeit, die Verbindung zu unterbrechen und in Zustand 3 zu gehen, um dort zu verbleiben. Dieses Ereignis erlaubt eine rasche Abmeldung in jedem Zustand. Die Rückkehr zum Initialzustand ist nicht Teil der Modellierung und setzt deshalb einen in der Quelle nicht beschriebenen Vorgang voraus.

Offensichtlich erlaubt dieser Automat nicht nur die korrekten Ereignisfolgen, sondern auch solche, die zu keinem brauchbaren Ergebnis führen (z.B. ein idLogout als Reaktion auf ein idReadyReq). Dies ist im Sinne des Ramadge-Wonham-Modells, bei dem die Systembeschreibung alle physikalisch möglichen Vorgänge zu erfassen hat.

Die Schnittstelle SWCON zwischen Steuerung und Nebenstellenanlage wird durch den Automaten in Abbildung 5.7 beschrieben. Steuerbare Ereignisse (in der Abbildung kursiv dargestellt) bedeuten Befehle der Steuerung an die Nebenstellenanlage, nicht steuerbare Transitionen (nicht kursiv) hingegen Meldungen der letzteren an die Steuerung. Die Ereignisse sind wie folgt zu interpretieren:

- swLogIn : Befehl der Steuerung zur Anmeldung eines Bearbeiters.
- swLoggedIn : Bestätigung des Befehls zur Anmeldung.
- swActivate : Befehl der Steuerung zur Aktivierung eines Bearbeiters.
- swReady : Bestätigung des Befehls zur Aktivierung.

- swDeactivate: Befehl der Steuerung zur Deaktivierung eines Bearbeiters.
- swNotReady : Bestätigung des Befehls zur Deaktivierung.
- swLogOut : Befehl der Steuerung zur Abmeldung eines Bearbeiters.
- swLoggedOut : Bestätigung des Befehls zur Abmeldung.
- swDeregister: Befehl zur langfristigen Stilllegung des Bearbeiters.

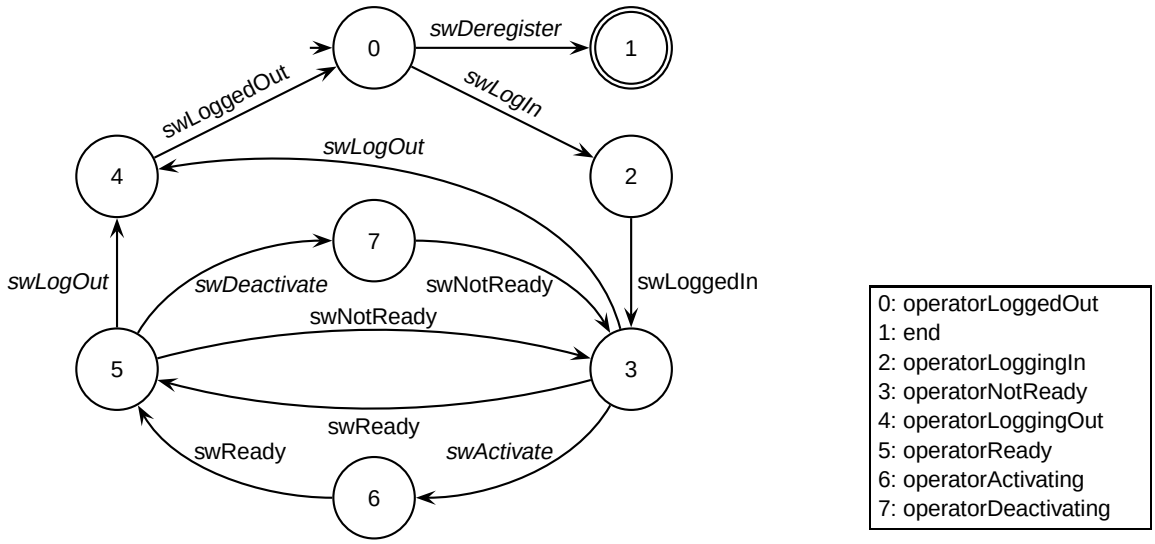


Abbildung 5.7: Die Schnittstelle SWCON zwischen Steuerung und Nebenstellenanlage

Im Initialzustand sind die Befehle swDeregister und swLogIn möglich. swDeregister bedeutet die Stilllegung des Bearbeiters und führt deshalb in den Zustand 1 (end), aus dem kein Weg mehr herausführt. Eine Wiederaufnahme der Tätigkeit geschieht wie oben erwähnt außerhalb des modellierten Systems und führt in den Initialzustand zurück. Wichtiger ist der Befehl swLogIn, mit dem die Steuerung die Nebenstellenanlage auffordert, den Bearbeiter anzumelden. Die Steuerung wartet im Zustand 2 (operatorLoggingIn) auf die Bestätigung swLoggedIn und schaltet danach in den Zustand 3 (operatorNotReady). Wie auf der Abbildung verfolgt werden kann, sind auch die Möglichkeiten zur Aktivierung bzw. Deaktivierung und zur Abmeldung des Bearbeiters gegeben. Die Aktivierung erfolgt im Normalfall nach der Anforderung swActivate, die als Befehl an die Nebenstellenanlage gesendet und mit swReady beantwortet wird. Es ist aber auch eine unaufgeforderte Aktivierung durch das Ereignis swReady in Zustand 3 möglich. Dies ist z.B. der Fall, wenn für den Arbeitsplatz eine automatische Aktivierung unmittelbar nach der Anmeldung eingerichtet wird. Wie im Falle des Automaten ID sind auch hier sowohl brauchbare als unbrauchbare Ereignisfolgen möglich.

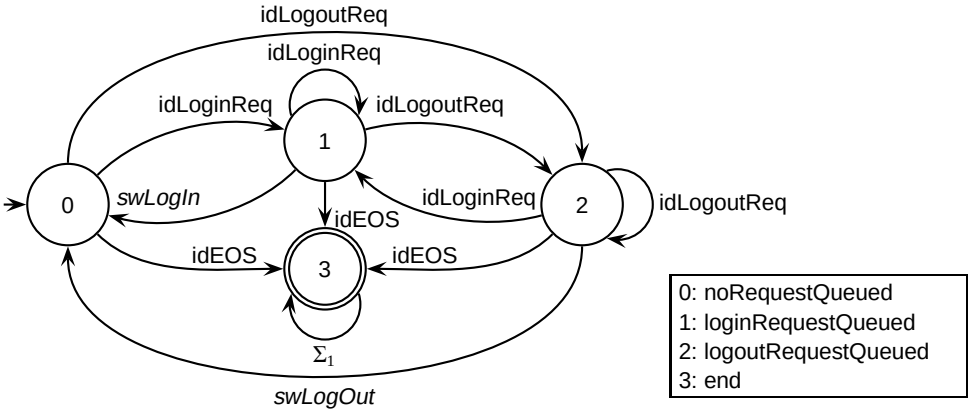
Das Modell für das System, $\mathcal{A}_{\mathcal{P}}$, wird aus der parallelen Komposition der Automaten ID und SWCON gewonnen. Daraus ergibt sich für das System das Alphabet in Tabelle 5.1. Der resultierende Automat hat 32 Zustände und 152 Transitionen und kann hier auf Grund seiner Größe nicht vollständig abgebildet werden. Ein Ausschnitt davon ist Abbildung 5.13 zu sehen, allerdings werden auch dort nicht alle Transitionen der gezeigten Zustände wiedergegeben.

Die Spezifikation für das gewünschte Verhalten wird mit Hilfe der Automaten EA1 bis EA5 in den Abbildungen 5.8 bis 5.12 erstellt. Darin werden die Ereignisse der Automaten ID und SWCON miteinander verknüpft und damit die physikalisch möglichen Ereignisfolgen auf ein sinnvolles Protokoll eingeschränkt.

Die Automaten EA1 und EA2 bestimmen die Kommunikationsmöglichkeiten in der Richtung vom Arbeitsplatz zur Nebenstellenanlage. EA1 schreibt z.B. vor, dass der Befehl swLogIn nur unmittelbar nach einer Anforderung idLoginReq an die Nebenstellenanlage gesendet werden darf. In ähnlicher Form werden Reihenfolgen für die Ereignisse idLogoutReq und swLogOut festgelegt. Die Ereignisse, die dabei keine Rolle spielen, werden durch das Hinzufügen von Schleifen, die von jedem Zustand zu ihm selbst zurückführen (engl. *selfloops*), ignoriert. Der Übersichtlichkeit halber sind diese Schleifen nicht eingezeichnet, sondern lediglich am unteren

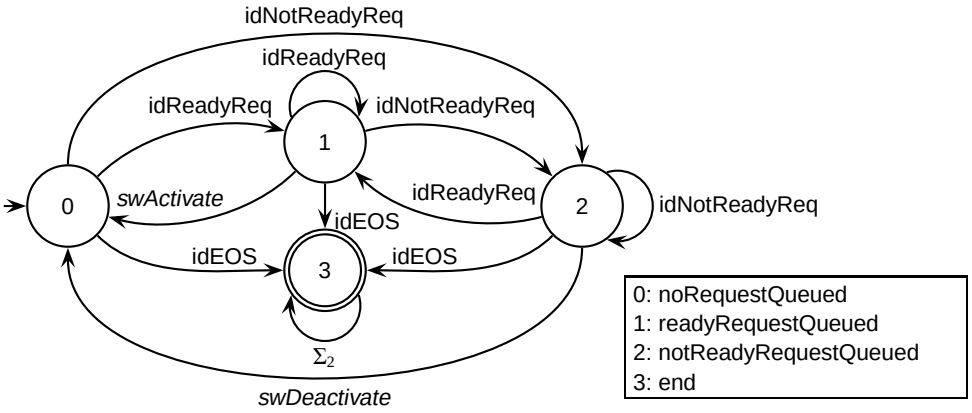
Steuerbar	Nicht steuerbar
idLogin	idLoginReq
idLogout	idLogoutReq
idReady	idReadyReq
idNotReady	idNotReadyReq
swDeregister	idEOS
swLogIn	swLoggedIn
swLogOut	swLoggedOut
swActivate	swReady
swDeactivate	swNotReady

Tabelle 5.1: Das Alphabet der Automaten für die Telefonnummernauskunft



Σ_1 = {idLoginReq, idLogoutReq, swLogIn, swLogOut}
Selfloops: $\Sigma \setminus$ {idLoginReq, idLogoutReq, swLogIn, swLogOut, idEOS}

Abbildung 5.8: Der Hilfsautomat EA1



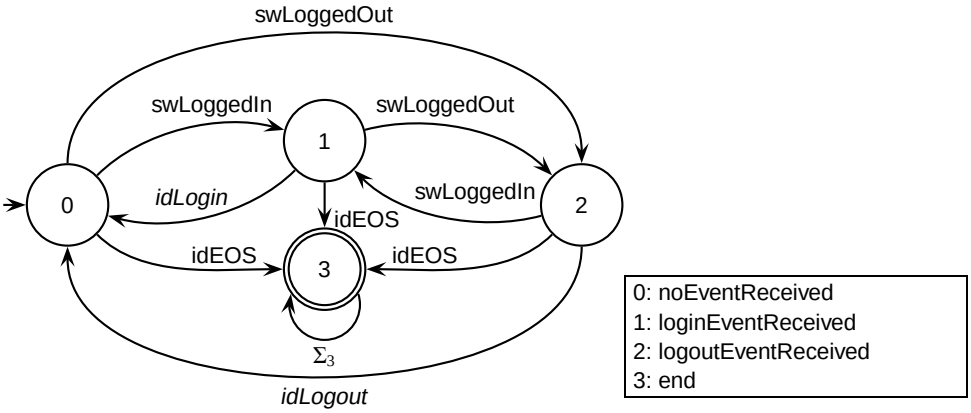
Σ_2 = {idReadyReq, idNotReadyReq, swActivate, swDeactivate}
Selfloops: $\Sigma \setminus$ {idReadyReq, idNotReadyReq, swActivate, swDeactivate, idEOS}

Abbildung 5.9: Der Hilfsautomat EA2

Rand der Abbildung angegeben. EA2 schreibt das gleiche Verhalten bezüglich der Ereignisse idReadyReq, swActivate, idNotReadyReq und swDeactivate vor.

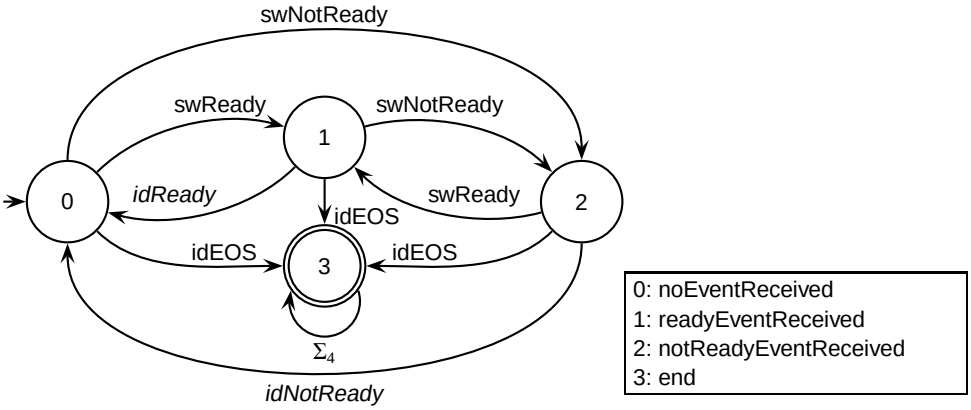
Die Automaten EA3 und EA4 bestimmen die Kommunikationsmöglichkeiten in der entgegengesetzten Richtung. EA3 legt z.B. fest, dass die Anmeldebestätigung idLogin nur unmittelbar nach einer Bestätigung swLog-

gedIn seitens der Nebenstellenanlage erfolgen darf. EA4 schreibt das gleiche Verhalten bezüglich der Ereignisse idReady und swReady vor.



Σ_3 = {swLoggedIn, swLoggedOut, idLogin, idLogout}
Selfloops: $\Sigma \setminus$ {swLoggedIn, swLoggedOut, idLogin, idLogout, idEOS}

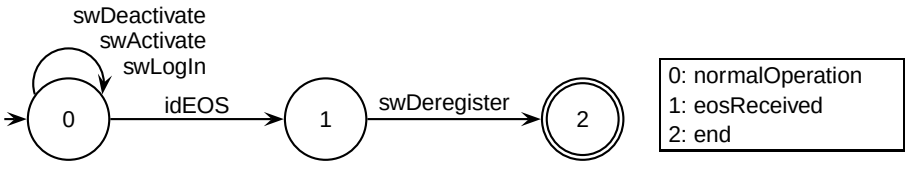
Abbildung 5.10: Der Hilfsautomat EA3



Σ_4 = {swReady, swNotReady, idReady, idNotReady}
Selfloops: $\Sigma \setminus$ {swReady, swNotReady, idReady, idNotReady, idEOS}

Abbildung 5.11: Der Hilfsautomat EA4

Ein letzter Automat EA5 gibt an, dass nach einem Verbindungsabbruch mit dem Ereignis idEOS die Ereignisse swLogIn, swActivate und swDeactivate gesperrt werden. Damit bleibt laut Abbildung 5.7 nur noch die Möglichkeit, den Bearbeiter abzumelden und danach mit swDeregister stillzulegen.



Selfloops: $\Sigma \setminus$ {swLogIn, swActivate, swDeactivate, swDeregister, idEOS}

Abbildung 5.12: Der Hilfsautomat EA5

Mit diesen Automaten wird der Automat $\mathcal{A}_{\mathcal{E}}$ erstellt. Es gilt

$$\mathcal{A}_{\mathcal{E}} := EA1 \times \ldots \times EA5.$$

Mit $\mathcal{A}_{\mathcal{E}}$ wird, wie in den Algorithmen für die Überwachersynthese vorgesehen, das Produkt $\mathcal{A}_{\mathcal{P}} \times \mathcal{A}_{\mathcal{E}}$ gebildet, das die Spezifikation für das gewünschte Verhalten darstellt. Das Ergebnis ist ein Automat mit 286 Zuständen und 1160 Transitionen. 23 dieser Zustände sind verboten, so dass sich die Spezifikation $\mathcal{A}_{\mathcal{P}} \times \mathcal{A}_{\mathcal{E}}$ nicht wie gewünscht realisieren lässt, obwohl die Anforderungen, die durch die Automaten EA1 bis EA5 dargestellt werden, keineswegs abwegig erscheinen. In der Quelle [77] wird der Frage, warum die Spezifikation nicht steuerbar ist, nicht weiter nachgegangen. Eine tiefere Analyse führt jedoch zu einem besseren Verständnis des Beispiels, das für spätere Erläuterungen notwendig ist.

5.4.3 Gründe für die Nicht-Steuerbarkeit

Ein Gegenbeispiel für die Steuerbarkeit der Spezifikation kann anhand der verbotenen Zustände gefunden werden. Um dies zu verdeutlichen, zeigen die Abbildungen 5.13 und 5.14 Ausschnitte der Automaten $\mathcal{A}_{\mathcal{P}}$ und $\mathcal{A}_{\mathcal{P}} \times \mathcal{A}_{\mathcal{E}}$. Insbesondere kann in Zustand 10 des Systems $\mathcal{A}_{\mathcal{P}}$ das nicht steuerbare Ereignis swReady auftreten. Der entsprechende Zustand (10,196) in $\mathcal{A}_{\mathcal{P}} \times \mathcal{A}_{\mathcal{E}}$ sieht dieses Ereignis jedoch nicht vor und ist deshalb verboten.

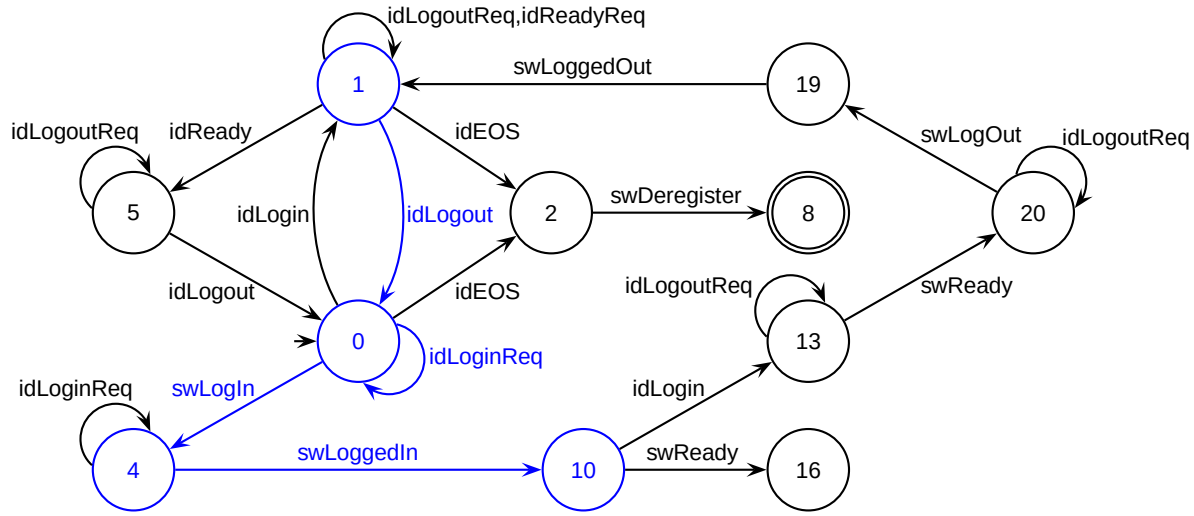


Abbildung 5.13: Teil des Automaten $\mathcal{A}_{\mathcal{P}}$ = ID || SWCON

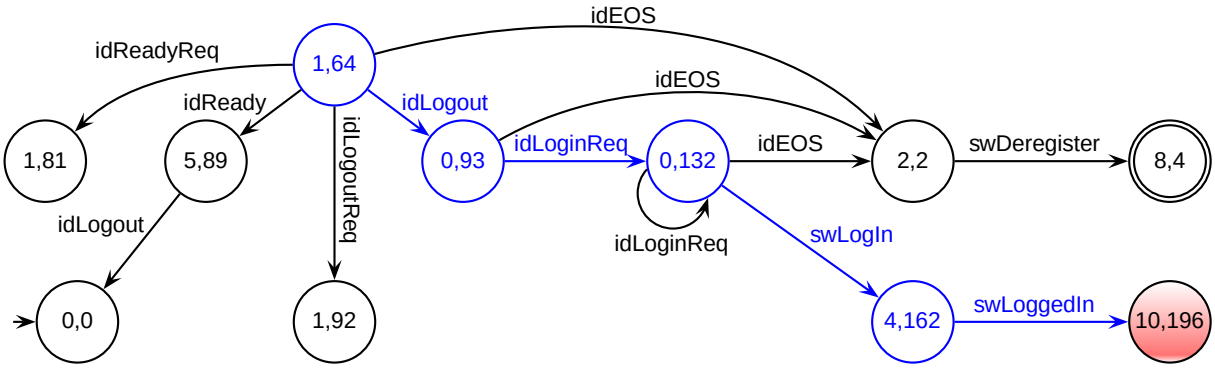


Abbildung 5.14: Teil des Automaten $\mathcal{A}_{\mathcal{P}} \times \mathcal{A}_{\mathcal{E}}$

Es folgt eine Analyse des Systemverhaltens unter einer hypothetischen Steuerung, die versuchen würde, die gegebene Spezifikation zu implementieren. Der Zustand (10,196) in $\mathcal{A}_{\mathcal{P}} \times \mathcal{A}_{\mathcal{E}}$ kann dann vom Initialzustand aus über die Ereignisfolge in Abbildung 5.15 erreicht werden. Das Zusammentreffen der Ereignisse idLogoutReq und swReady in dem Diagramm bedeutet, dass es irrelevant ist, welches davon zuerst an der Steuerung ankommt. In beiden Fällen wird nach beiden Transitionen derselbe Zustand erreicht. Diese Ereignisse lassen sich

auf den Automaten $\mathcal{A}_{\mathcal{P}}$ und $\mathcal{A}_{\mathcal{P}} \times \mathcal{A}_{\mathcal{E}}$ in den Abbildungen 5.13 und 5.14 verfolgen. Letztere ermöglicht dies nur ab dem Ereignis idLogout, was jedoch für die folgende Erläuterung ausreichend ist (alternativ dazu können alle Ereignisse auf den Abbildungen 5.6 bis 5.12 verfolgt werden). Das letzte Ereignis im Zeitdiagramm ist das von der Spezifikation nicht vorgesehene swReady.

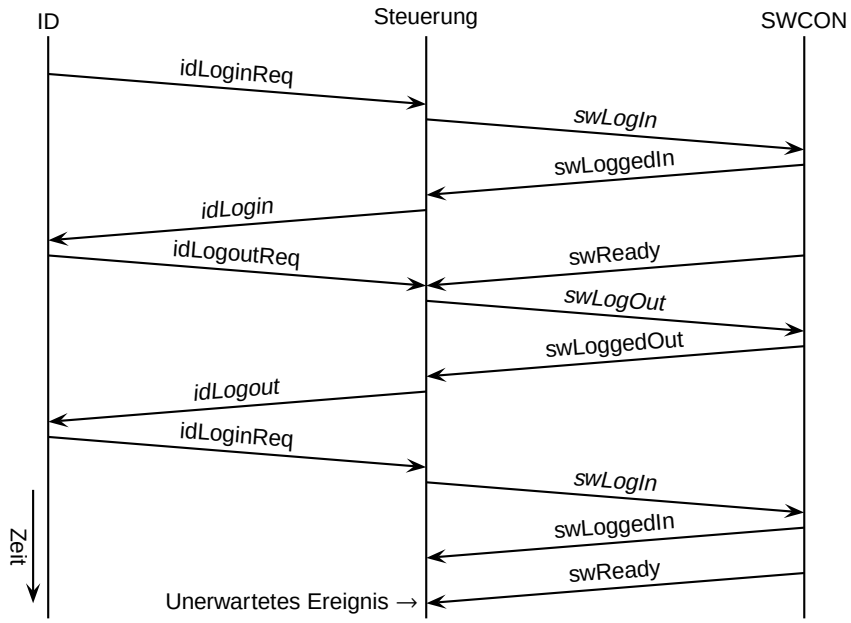


Abbildung 5.15: Gegenbeispiel für die Steuerbarkeit als Zeitdiagramm

Zunächst meldet sich ein Bearbeiter mit idLoginReq an, was die Steuerung dazu veranlasst, der Nebenstellenanlage den Befehl swLogIn zu schicken. Diese verarbeitet den Vorgang und bestätigt ihn mit swLoggedIn. Für den Bearbeiter ist aber auch die automatische Aktivierung eingerichtet, so dass die Nebenstellenanlage kurz darauf noch die Meldung swReady an die Steuerung schickt. Diese hat inzwischen den Arbeitsplatz mit idLogin in den angemeldeten Zustand versetzt. Der Bearbeiter möchte sich, aus welchem Grund auch immer, sofort wieder abmelden und schickt der Steuerung die Anforderung idLogoutReq. Die Steuerung reagiert mit dem Befehl swLogOut an die Nebenstellenanlage, die diesen wiederum mit swLoggedOut bestätigt. Zu diesem Zeitpunkt befindet sich das System auf Abbildung 5.13 in Zustand 1 und die Steuerung auf Abbildung 5.14 in Zustand (1,64). Die Steuerung hat die Wahl zwischen den Befehlen idLogout und idReady. In dem hier gezeigten Fall reagiert sie mit dem Befehl idLogout und versetzt damit das System in den Initialzustand zurück. Wie in Abbildung 5.14 zu sehen ist, kehrt jedoch die Steuerung nicht in ihren Initialzustand zurück, sondern verbleibt in Zustand (0,93). Der Grund dafür ist, dass sich die Steuerung merkt, dass zu einem früheren Zeitpunkt das Ereignis swReady von der Nebenstellenanlage eingetroffen ist, und dass der Bearbeiter daraufhin noch nicht aktiviert wurde. Dies ist auf den Automaten EA1 bis EA5 auf den Abbildungen 5.8 bis 5.12 leicht zu erkennen. Bis auf EA4 sind nun alle Automaten in ihren Initialzustand zurückgekehrt, EA4 verweilt jedoch noch in Zustand 1 und könnte erst nach dem Ereignis idReady zum Zustand 0 gelangen. Dieses ist jedoch im aktuellen Zustand des Systems nicht vorgesehen, und so wartet die Steuerung auf die nächste Anmeldeanforderung des Bearbeiters in einem Zustand, der nicht der Initialzustand ist.

Für den Bearbeiter sieht alles so aus, als wäre das System frisch gestartet worden. Beim nächsten Anmeldeversuch verhält sich das System aus seiner Sicht jedoch nicht korrekt. Nach der neuen Anmeldeanforderung mit idLoginReq folgen wie beim ersten Mal swLogIn und dessen Bestätigung swLoggedIn. Das System auf Abbildung 5.13 befindet sich dann im Zustand 10, die Steuerung auf Abbildung 5.14 in Zustand (10,196). Nun bekommt die Steuerung auf Grund der automatischen Aktivierung die Meldung swReady von der Nebenstellenanlage, die aber nicht vorgesehen ist. Die Folgen hängen davon ab, wie die Software der Steuerung mit Ausnahmesituationen umgeht. Der normale Ablauf ist aber nicht mehr möglich, was sich für den Bearbeiter im besten Fall durch das Ausbleiben der Anmeldung bemerkbar macht.

Diese Analyse zeigt, wie schwierig der Entwurf der Steuerung nur nach gesundem Menschenverstand sein kann. Ein solcher Fall ist äußerst schwierig vorherzusehen, und auf Grund der relativ langen Ereignisfolge, die notwendig ist, um den Fehler hervorzubringen, auch in einer Testphase nicht leicht zu finden. Wie kritisch die Reihenfolge der Ereignisse für die Reproduzierbarkeit des Fehlers ist, wird noch einmal deutlich, wenn angenommen wird, die Steuerung hätte sich nach dem Ereignis swLoggedOut in Zustand (1,64) nicht wie oben für den Befehl idLogout, sondern für die Befehle idReady, idLogout entschieden. Dann finden alle Automaten in ihren Initialzustand zurück und es tritt kein Fehler auf.

5.4.4 Synthese der Steuerung

An dieser Stelle kann das Beispiel gemäß der Quelle [77] wieder aufgenommen werden. Es folgt die Berechnung des Überwachers aus der Systembeschreibung und der Spezifikation. Das Ergebnis ist ein Automat mit 239 Zuständen und 969 Transitionen, der sich mit dem Standardverfahren zur Automatenminimierung auf 102 Zustände und 430 Transitionen reduzieren lässt[3]. Wie die Autoren bemerken, werden sämtliche Schnittstellen korrekt bedient, die Spezifikation wird eingehalten, Verklemmungen sind ausgeschlossen und das Systemverhalten wird nur minimal eingeschränkt. Die ersten Ergebnisse nach dem Einsatz eines Prototyps waren außerordentlich positiv. Die Implementierung der Steuerung lässt sich automatisieren, und damit können Änderungen an den Anforderungen beispielsweise für den Verbindungsablauf im Vergleich zu der konventionellen Programmierung sehr schnell umgesetzt werden. Die Berechnungszeit für die Steuerung liegt unter einer Sekunde, da die Automaten relativ klein sind.

Die gleiche Steuerung wurde anhand der hier gezeigten Automaten im Rahmen dieser Dissertation mit symbolischen Verfahren unter Einbindung des Programmpakets CUDD[4] für die Handhabung von binären Entscheidungsdiagrammen implementiert. Die Lösung kann wie in den Theoremen 4.8, 4.11 oder 5.5 vorgesehen berechnet werden. Zwar steht der originale Überwacher nicht zum Vergleich zur Verfügung, jedoch stimmen die Anzahl der Zustände und die der Transitionen in beiden Fällen überein. Dies kann als Bestätigung der hier vorgestellten Methoden gesehen werden. Interessant ist auch zu bemerken, dass dieses Problem trotz der vielen Schleifen VVK-frei ist und somit in der in Abschnitt 5.3 neu beschriebenen Problemklasse liegt.

Die Rechenzeit für die Lösung beträgt auf einem Standardrechner mit einem Athlon 1,8 GHz-Prozessor und 1GB RAM ca. 20ms. Damit sind die hier vorgestellten Methoden auch schneller als das Werkzeug CTCT[5], das die Autoren in [77] benutzten. Der Zeitunterschied spielt in diesem Beispiel keine große Rolle, weil die Zustandsräume klein bleiben und sich das Problem deshalb auch mit Hilfe einer expliziten Darstellung der Transitionsrelation lösen lässt. Das Beispiel in Abschnitt 5.5 wird jedoch zeigen, wie schnell sich diese Differenz bis ins Unermessliche steigern kann.

5.4.5 Ein Schönheitsfehler

Trotz aller Sorgfalt kann in der Lösung in [77] noch ein subtiler Fehler nachgewiesen werden. Dieser lässt sich mit Hilfe von Abbildung 5.16 erklären, die einen Teil des Überwachers zeigt. Hier sind die Zustände der Spezifikation, die durch den Synthesealgorithmus entfernt werden, nur noch mit gestrichelten Linien dargestellt (vgl. Abbildung 5.14). Die Synthese entfernt wie erwartet den verbotenen Zustand (10,196). Da das Ereignis swLoggedIn nicht steuerbar ist, wird dadurch auch (4,162) zum verbotenen Zustand. Das Ereignis swLogIn, das zu (4,162) führen würde, ist hingegen steuerbar. Es wird in (0,132) verboten, und dieser Zustand bleibt erhalten.

Angenommen, die ersten neun Ereignisse aus dem Zeitdiagramm in Abbildung 5.15 – idLoginReq bis idLogout – treten wieder in der dort angegeben Reihenfolge auf. Wie in Abschnitt 5.4.3 bereits gezeigt, befindet sich danach das System auf Abbildung 5.13 wieder in dem Initialzustand, der Überwacher auf Abbildung 5.16 jedoch in dem Zustand (0,93).

[3] Die Minimierung war für die Autoren der Anwendung sinnvoll, weil die Zustandsräume nicht symbolisch dargestellt wurden. Bei einer symbolischen Darstellung ist eine Minimierung der Automaten nicht unbedingt von Vorteil.

[4] Entwickelt von Fabio Somenzi an der University of Colorado at Boulder

[5] Entwickelt in der Gruppe von Prof. Wonham an der University of Toronto, Kanada.

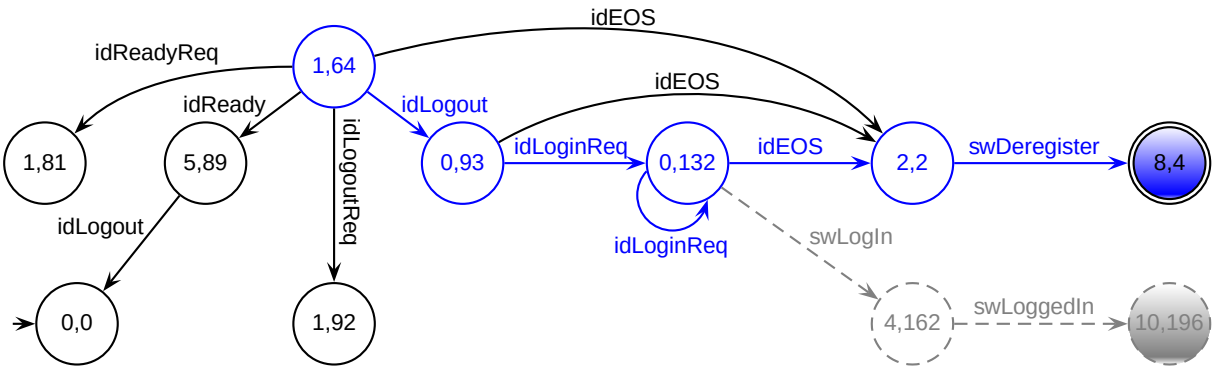


Abbildung 5.16: Teil des Überwachers $\mathcal{A}_{\mathcal{S}}$ für die Auskunftzentrale

Wenn der Bearbeiter sich nun erneut mit idLoginReq anmelden möchte, wechselt der Überwacher zu Zustand (0,132). Das zweite swLogIn in dem Zeitdiagramm in Abbildung 5.15 kann nun nicht auftreten, und damit auch nicht das darauf folgende swLoggedIn, das zu dem verbotenen Zustand führen würde. Allerdings schneidet die Synthese damit dem Zustand (0,93) auch den Weg zurück zum Initialzustand (0,0) ab. Da Zustand (0,93) trotzdem co-erreichbar bleibt (es gibt immer noch den Weg über idEOS und swDeregister zu dem markierten Zustand (8,4)), ist dies im Sinne der Überwachersynthese korrekt. Der Bearbeiter hat jedoch den Eindruck einer Verklemmung: Auch wenn er die Anforderung zur Anmeldung wiederholt, reagiert das System nicht darauf und bleibt in der Schleife um den Zustand (0,132) hängen. Um das Problem zu beheben, muss der Bearbeiter mit idEOS und swDeregister außer Betrieb genommen werden und kann seine Tätigkeit erst nach dem nicht modellierten Zurücksetzen des Systems wieder aufnehmen. Dies ist sicher nicht das von ihm erwartete Systemverhalten.

Um dieses Problem zu vermeiden, müssten Zustände wie (0,93), denen der Weg zurück zum Initialzustand während der Synthese abgeschnitten wird, ebenfalls entfernt werden. Mit dem bisher vorgestellten Ramadge-Wonham-Modell lassen sich jedoch solche Bedingungen nur sehr begrenzt spezifizieren. Dies ist möglich, wenn der Initialzustand der einzige markierte Zustand der Spezifikation ist, denn dann bedeutet die Co-Erreichbarkeit, dass immer zu diesem Zustand zurückgekehrt werden kann. Im Falle der hier verwendeten Spezifikation würde jedoch eine zusätzliche Markierung des Initialzustands die Synthese nicht dazu veranlassen, unerwünschte Zustände wie (0,93) zu entfernen, solange diese einen Pfad zu dem ebenfalls markierten Zustand (8,4) behalten und damit co-erreichbar bleiben. Was hier benötigt wird ist ein Synthesealgorithmus, der einen Überwacher berechnet, mit dem sich der Bearbeiter beliebig oft anmelden kann. Dies ist jedoch eine Fairnesseigenschaft. Wie bereits in Abschnitt 2.8 erwähnt, können solche Eigenschaften nicht in der konventionellen Überwachersynthese gehandhabt werden. In Kapitel 6 wird die Überwachersynthese unter anderem um Fairnesseigenschaften ergänzt. Das hier beschriebene Problem wird dann in Abschnitt 6.5 wieder aufgegriffen und mit Hilfe der neuen Techniken gelöst.

5.5 Das Nim-Spiel

Aus theoretischer Sicht kommt die Aufgabe, ein reaktives System so zu steuern, dass unter Berücksichtigung aller physikalisch möglichen Reaktionen ein gewünschtes Ergebnis erreicht wird, der Anwendung einer Gewinnstrategie für ein Spiel gleich, in dem es darum geht, einen bestimmten Zustand unter Berücksichtigung aller möglichen Züge des Gegners zu erreichen. Zweipersonenspiele sind deshalb Metaphern für reaktive Systeme, und die Ermittlung einer Gewinnstrategie kommt in diesem Sinne der Synthese einer Steuerung gleich. In diesem Abschnitt geht es darum, eine Gewinnstrategie für das Nim-Spiel zu synthetisieren.

5.5.1 Spielregeln und Gewinnstrategie

Das Nim-Spiel gibt es in verschiedenen Varianten. Allen gemeinsam ist eine Ansammlung von Steinen oder Hölzern, wie z.B. die Streichhölzer in Abbildung 5.17. Hier sind die Streichhölzer in vier Reihen angeordnet, wobei Reihe n (Reihennummern oben bei 1 angefangen) zu Beginn $2n - 1$ Hölzer enthält. Andere Varianten entstehen, indem die Anordnung der Hölzer bzw. die Anzahl der Reihen nach Belieben verändert werden. Das Spiel wird von zwei Spielern gespielt, die abwechselnd Hölzer aus dem Spielfeld entfernen. In jedem Zug wählt ein Spieler eine Reihe und nimmt dann aus dieser – und nur aus dieser – so viele Streichhölzer, wie er mag, jedoch mindestens eins. Das Ziel kann entweder sein, selbst das letzte Streichholz zu ziehen, oder den Gegner zu zwingen, dies zu tun.

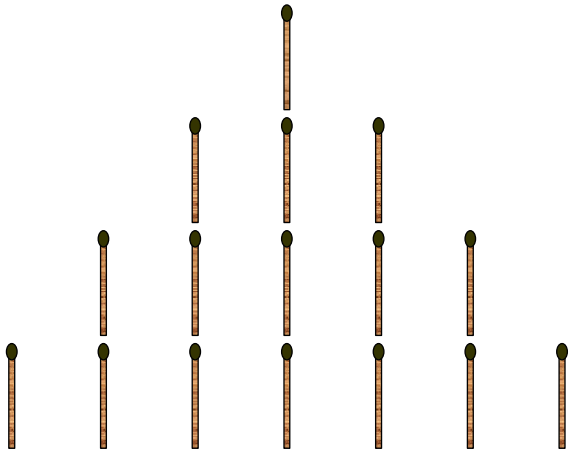


Abbildung 5.17: Die Anfangsposition für das Nim-Spiel mit vier Reihen

Laut Bewersdorff [5] ist Nim „wahrscheinlich das bekannteste Spiel, für das eine vollständige mathematische Theorie existiert“. Die Gewinnstrategie lässt sich mit Hilfe einer einfachen Formel beschreiben, die 1902 von Charles Bouton [8] gefunden wurde. Danach gibt es für eine Spielposition eine Gewinnstrategie, wenn die Modulo-2-Summe[6] der Anzahl der verbleibenden Hölzer ungleich 0 ist. Zunächst wird die Modulo-2-Summe s der Anzahl verbleibender Hölzer gebildet. Sei l_i die Anzahl der verbleibenden Hölzer in Reihe i. Eine Reihe, aus der im nächsten Zug gezogen werden darf, verrät sich dadurch, dass es möglich ist, so viele Hölzer zu entfernen, dass in dieser Reihe noch $l_i +_2 s$ Hölzer übrig bleiben, wobei $+_2$ die Modulo-2-Summe bezeichnet. Damit wird erreicht, dass die Modulo-2-Summe s nach dem Zug gleich Null wird. Unabhängig davon, welchen Zug der Gegner nun macht, muss danach die Summe wieder ungleich Null sein. Besteht nun das Ziel darin, selbst das letzte Streichholz zu nehmen, führt diese Strategie direkt zum Ziel. Soll andererseits der Gegner gezwungen werden, das letzte Streichholz zu ziehen, wird mit derselben Strategie gespielt, bis der nächste Zug zu einer Position führt, in der in allen noch nicht leeren Reihen nur ein Streichholz liegen bleibt. Dann ist dieser Zug so zu gestalten, dass danach die Anzahl der nicht leeren Reihen ungerade ist. Damit wird der Gegner das letzte Streichholz nehmen müssen, egal in welcher Reihenfolge die Reihen geleert werden.

Im Folgenden wird eine Gewinnstrategie für den zweiten dieser Fälle mit Hilfe der Überwachersynthese erzeugt, die andere Variante ist ähnlich zu berechnen. Dabei steht die Bestätigung von Boutons Ergebnissen nicht im Vordergrund, sondern vor allem die einfache Skalierbarkeit des Beispiels durch Hinzunahme neuer Streichholzreihen. Dadurch entsteht ein breites Spektrum an Zustandsräumen, mit denen das Problem der Zustandsexplosion und damit die Vorteile der in dieser Arbeit entwickelten Lösungen gegenüber der herkömmlichen expliziten Darstellung deutlich gemacht werden können.

[6] Durch die Darstellung der Summanden als Dualzahlen und dann mittels übertragsloser Addition zu berechnen.

5.5.2 Modellierung

Für die Modellierung des Spiels werden die einzelnen Reihen mit getrennten Automaten erfasst. Die Automaten für die ersten zwei Reihen sind in Abbildung 5.18 dargestellt.

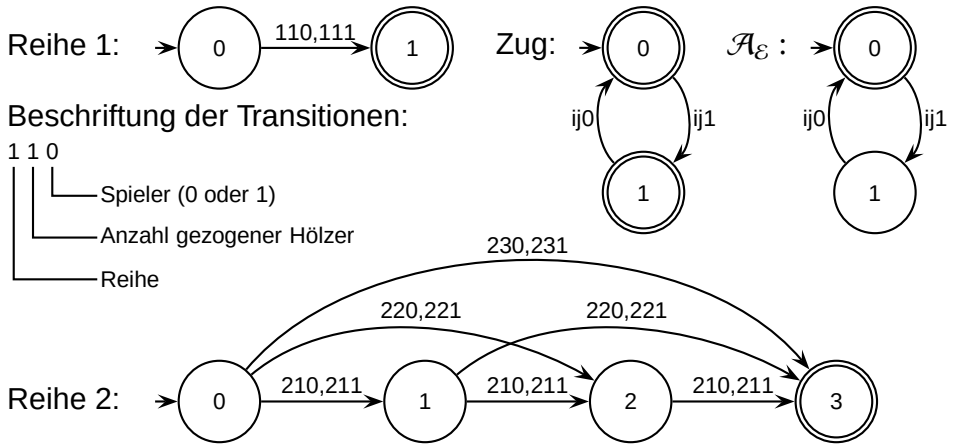


Abbildung 5.18: Die Automaten für die zwei ersten Reihen des Nim-Spiels

Der Automat für Reihe n hat $2n$ Zustände, einen für jede mögliche Anzahl der im Spielverlauf noch verbleibenden Hölzer. Im Initialzustand hat Reihe n wie in Abbildung 5.17 $2n - 1$ Streichhölzer, mit jedem Schritt nach rechts ein Streichholz weniger, bis im letzten Zustand die Reihe leer ist. Außerdem ist der letzte Zustand jeder Reihe markiert. Damit wird erreicht, dass nach der parallelen Komposition der einzelnen Reihenautomaten genau die Zustände markiert sind, in denen das Spiel zu Ende ist. Die Transitionen stellen die möglichen Züge dar. Aus ihrer Beschriftung gehen – wie in der Abbildung erläutert – die Reihennummer, die Anzahl der gezogenen Hölzer und der Spieler hervor. Da eine Gewinnstrategie für Spieler 1 gefunden werden soll, sind seine Züge (ungerade Transitionen) steuerbar, die von Spieler 0 (gerade Transitionen) nicht steuerbar.

Die Automaten für die einzelnen Reihen erfassen noch nicht die Regel, dass abwechselnd gespielt werden muss. Dazu dient der Hilfsautomat Zug. Die mit ij0 und ij1 gekennzeichneten Transitionen stehen für alle Züge des Spielers 0 bzw. 1. Durch die parallele Komposition der Reihen mit diesem Automat wird die gewünschte Abwechslung erzwungen. Darüber hinaus bestimmt auch die Wahl des Initialzustands von Zug, welcher Spieler beginnt. In Abbildung 5.18 ist dies Spieler 1. Die Zustände des Hilfsautomaten sind beide markiert, um die bereits beschriebene Markierung der Reihenautomaten nicht zu beeinflussen.

Die einzelnen Automaten werden ähnlich wie in Beispiel 4.2 mit booleschen Variablen beschriftet. Die Einzelheiten spielen in dem weiteren Verlauf des Beispiels keine besondere Rolle, weshalb die Kodierung in den Abbildungen nicht angegeben wird. Wichtig ist, dass es damit möglich wird, alle von nun an notwendigen Operationen symbolisch durchzuführen. Der Automat $\mathcal{A}_\mathcal{P}$ für die Systembeschreibung entsteht z.B. aus der parallelen Komposition Reihe 1 || Reihe 2 || Zug und ist in Abbildung 5.19 zu sehen. Die Zustände wurden dabei in einfacher aufsteigender Reihenfolge nummeriert. Der Automat beschreibt alle möglichen Spielverläufe, insbesondere stellen die Zustände 7 bzw. 11 die zwei möglichen Ausgänge des Spiels dar. Wird Zustand 7 erreicht, hat Spieler 0 das letzte Streichholz gezogen und somit Spieler 1 gewonnen, in Zustand 11 ist es genau umgekehrt. Um zu spezifizieren, dass Spieler 1 gewinnen soll, genügt es, die Markierung von Zustand 11 zu entfernen. Dadurch wird dieser zu einem nicht co-erreichbaren Zustand und folglich während der Synthese entfernt. Der Automat für die Spezifikation wird deshalb aus dem Produkt des Automaten $\mathcal{A}_\mathcal{P}$ in Abbildung 5.19 mit dem Hilfsautomaten $\mathcal{A}_\mathcal{E}$ in Abbildung 5.18 gebildet. Letzterer ist bis auf die Markierung identisch mit dem Automaten Zug. Damit wird sichergestellt, dass das Produkt die Transitionsrelation nicht verändert, gleichzeitig aber nur noch Zustände markiert sein können, die über einen Zug von Spieler 0 erreicht werden (s. Definition 2.10). Das Ergebnis ist in Abbildung 5.20 zu sehen, wobei die Zustände der Einfachheit halber wieder neu nummeriert wurden.

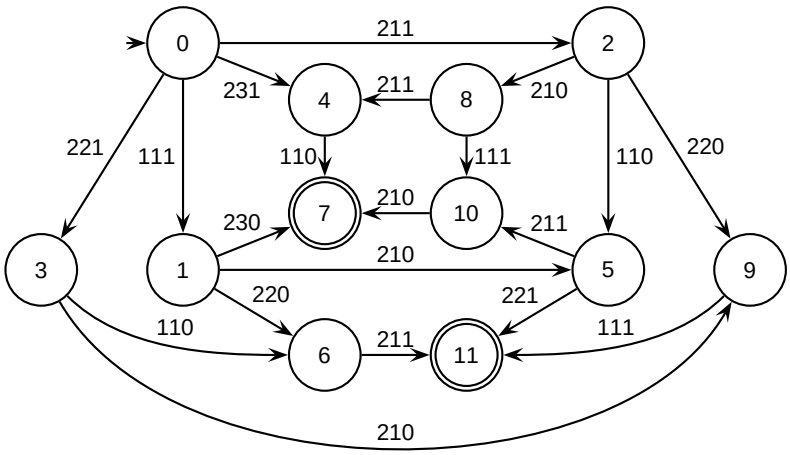


Abbildung 5.19: Der Automat $\mathcal{A}_{\mathcal{P}}$ für das Nim-Spiel mit zwei Reihen

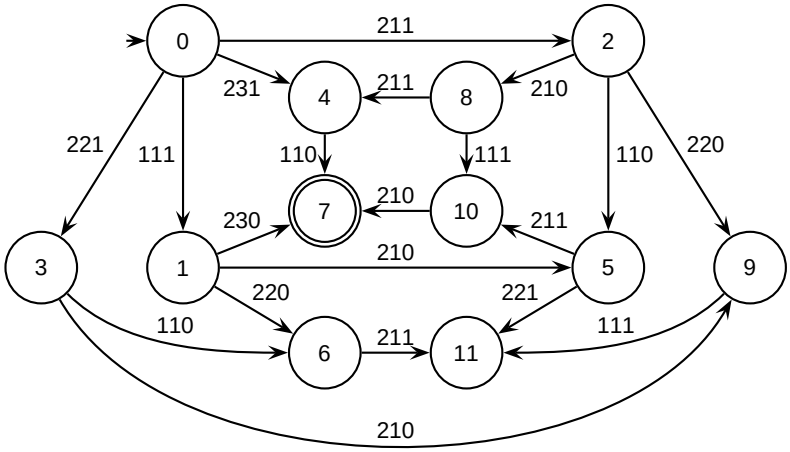


Abbildung 5.20: Der Automat $\mathcal{A}_{\mathcal{P}} \times \mathcal{A}_{\mathcal{E}}$ für das Nim-Spiel mit zwei Reihen

Als nächstes wird die Kripke-Struktur $\mathcal{K}_{\mathcal{A}_{\mathcal{P}} \times \mathcal{A}_{\mathcal{E}}}$ in Abbildung 5.21 erstellt, auf der die Gleichungssysteme für die Synthese zu lösen sind. Die Tatsache, dass in diesem Beispiel beide Spieler abwechselnd spielen, führt dazu, dass jeder Zustand entweder nur steuerbare oder nur nicht steuerbare Transitionen hat. Da auf der linken Hälfte der Kripke-Struktur nur nicht steuerbare Transitionen vorkommen, gibt es dort Zustände, die keine ausgehenden Transitionen haben.

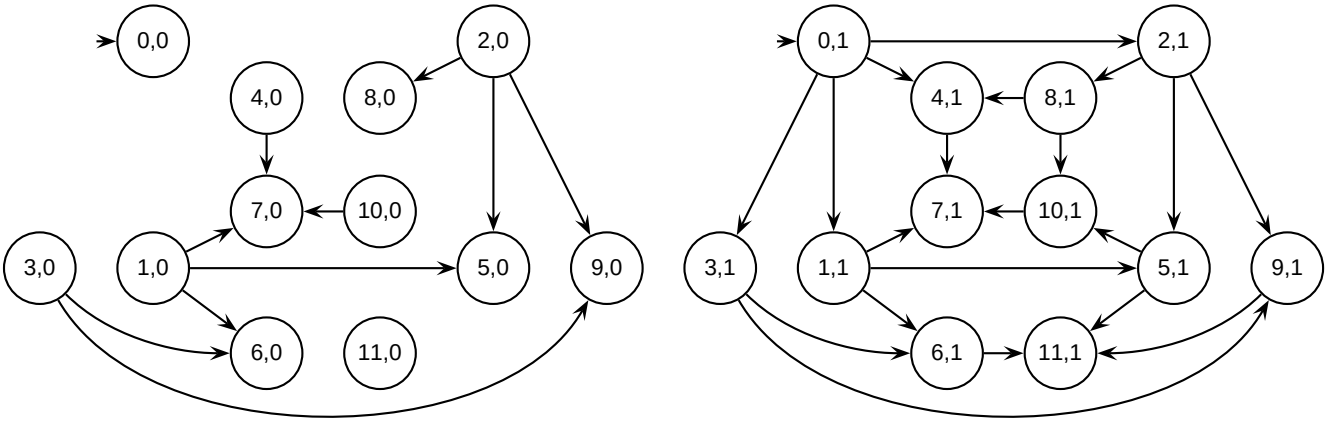


Abbildung 5.21: Die Kripke-Struktur für das Nim-Spiel mit zwei Reihen

5.5.3 Synthese der Gewinnstrategie

Der Überwacher für die gegebene Systembeschreibung $\mathcal{A}_\mathcal{P}$ und Spezifikation $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ steuert Spieler 1 so, dass erstens auf Grund der Co-Erreichbarkeit der unerwünschte Zustand 11 nicht mehr erreicht wird und zweitens auf Grund der Steuerbarkeit dabei keine Zustände betreten werden, in denen dem Gegner ein legitimer Zug verboten wird. Er stellt deshalb alle möglichen Gewinnstrategien für Spieler 1 dar.

Für die Berechnung des Überwachers kann im Prinzip jeder der in dieser Arbeit vorgestellten Algorithmen herangezogen werden. Zu bemerken ist jedoch, dass die verwendeten Automaten keine Schleifen aufweisen (weil ein genommenes Streichholz nie auf den Tisch zurückkehren kann). Damit fällt auch das Nim-Spiel in die neue Klasse der VVK-freien Probleme. Da sich das Beispiel durch Hinzunahme weiterer Reihen leicht skalieren lässt, eignet es sich besonders gut zum Vergleich zwischen den herkömmlichen Lösungsmöglichkeiten und dem neuen Ansatz in Theorem 5.5.

Als nächstes wird der Fall mit zwei Reihen anhand der Formeln 5.7 anschaulich berechnet. Die anfallenden Mengen werden wie in vorangegangenen Beispielen der Übersicht halber explizit dargestellt. Zunächst findet die Gleichung für u_n die nicht co-erreichbaren Zustände:

$$
\begin{aligned}
u_{bn}^{0} &= \{\} \\
\quad u_{v}^{0,0} &= \{\} \\
\quad\quad u_{b}^{0,0,0} &= \{\} \\
\quad\quad\quad u_{n}^{0,0,0,0} &= \mathcal{S} \\
\quad\quad\quad u_{n}^{0,0,0,1} &= \{(0,1),(1,1),(2,1),(3,1),(4,1),(5,1),(6,1),(8,1),(9,1),(10,1),(11,1)\} \\
\quad\quad\quad u_{n}^{0,0,0,2} &= \{(0,1),(2,1),(3,1),(5,1),(6,1),(8,1),(9,1),(11,1)\} \\
\quad\quad\quad u_{n}^{0,0,0,3} &= \{(2,1),(3,1),(6,1),(9,1),(11,1)\} \\
\quad\quad\quad u_{n}^{0,0,0,4} &= \{(3,1),(6,1),(9,1),(11,1)\} \\
\quad\quad\quad u_{n}^{0,0,0,\infty} &= \{(3,1),(6,1),(9,1),(11,1)\}.
\end{aligned}
$$

Nun kann der Fixpunkt $u_b^{0,0,\infty}$ berechnet werden. Dabei werden die gerade gefundenen Zustände auf die linke Hälfte der Kripke-Struktur gespiegelt. Dann werden ihnen die Zustände $(1,0)$ und $(2,0)$ hinzugefügt, weil diese durch das Entfernen der Zustände $(6,1)$ bzw. $(9,1)$ verboten geworden sind:

$$
\begin{aligned}
u_{b}^{0,0,0} &= \{\} \\
u_{b}^{0,0,1} &= \{(3,0),(6,0),(9,0),(11,0)\} \\
u_{b}^{0,0,2} &= \{(1,0),(2,0),(3,0),(6,0),(9,0),(11,0)\} \\
u_{b}^{0,0,\infty} &= \{(1,0),(2,0),(3,0),(6,0),(9,0),(11,0)\}.
\end{aligned}
$$

Der Fixpunkt $u_v^{0,\infty}$ findet keine neuen nicht co-erreichbaren Zustände:

$$
\begin{aligned}
u_{v}^{0,0} &= \{\} \\
u_{v}^{0,1} &= \{(1,1),(2,1),(3,1),(6,1),(9,1),(11,1)\} \\
u_{v}^{0,\infty} &= \{(1,1),(2,1),(3,1),(6,1),(9,1),(11,1)\}.
\end{aligned}
$$

Es folgen

$$
\begin{aligned}
u_{bn}^{1} &= \{(1,1),(2,1),(3,1),(6,1),(9,1),(11,1)\} \\
\quad u_{v}^{1,0} &= \{\} \\
\quad\quad u_{b}^{1,0,0} &= \{\} \\
\quad\quad\quad u_{n}^{1,0,0,0} &= \mathcal{S} \\
\quad\quad\quad u_{n}^{1,0,0,1} &= \{(0,1),(1,1),(2,1),(3,1),(4,1),(5,1),(6,1),(8,1),(9,1),(10,1),(11,1)\} \\
\quad\quad\quad u_{n}^{1,0,0,2} &= \{(0,1),((1,1),(2,1),(3,1),(5,1),(6,1),(8,1),(9,1),(11,1)\} \\
\quad\quad\quad u_{n}^{1,0,0,3} &= \{((1,1),(2,1),(3,1),(6,1),(9,1),(11,1)\} \\
\quad\quad\quad u_{n}^{1,0,0,\infty} &= \{(1,1),(2,1),(3,1),(6,1),(9,1),(11,1)\}.
\end{aligned}
$$

Damit ist das zweite Anhaltekriterium aus Lemma 5.6 bereits erreicht. Danach gilt $u_{bn}^{\infty} = u_n^{1,0,0,\infty} = \{(1,1),(2,1),(3,1),(6,1),(9,1),(11,1)\}$. Das Komplement davon, abgebildet auf den Automaten in Abbildung 5.20, ergibt die Zustandsmenge des gesuchten Überwachers. Die Zustände 5, 8 und 10 sind dabei nicht erreichbar und können weggelassen werden, so dass der Überwacher letztendlich – wie in Abbildung 5.22 zu sehen – aus den Zuständen 0, 4 und 7 besteht. Die Lösung besagt, dass Spieler 1 nur einen richtigen Zug machen kann, um das Spiel zu gewinnen. Dieser besteht darin, die zweite Reihe sofort zu räumen, so dass dem Gegner nur noch die Möglichkeit bleibt, das eine Streichholz in der ersten Reihe zu nehmen.

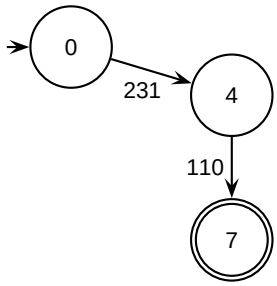


Abbildung 5.22: Der Überwacher für das Nim-Spiel mit zwei Reihen

5.5.4 Die Zustandsexplosion

Das vorangegangene Ergebnis mag trivial erscheinen, jedoch ist dies nur dem kleinen Zustandsraum des Spiels mit zwei Reihen zu verdanken. Werden mehr Reihen hinzugenommen, wächst der Zustandsraum überraschend schnell. Um die in dieser Arbeit entwickelten Verfahren zu testen, wurde ein Programm erstellt, das bei Eingabe der Zahl n ein Programm in C++ ausgibt, mit dem sich die Gewinnstrategie für das Spiel mit n Reihen ermitteln lässt. Es wurden damit Quellprogramme für unterschiedliche Werte von n generiert. Dabei konnten auch die Anzahl der Zustände und die der Transitionen des Systems ermittelt werden. Aus der Reihe der ersten Zahlen wurden dann folgende Formeln für die Anzahl der Zustände bzw. der Transitionen der erreichbaren Komponente der Spezifikation empirisch erstellt:

$$\left|Q_{\mathsf{Ac}(\mathcal{A}_{\mathcal{P}}\times\mathcal{A}_{\mathcal{E}})}\right| = 2^{n+1}n! - 2^n, \tag{5.8}$$

$$\left|\delta_{\mathsf{Ac}(\mathcal{A}_{\mathcal{P}}\times\mathcal{A}_{\mathcal{E}})}\right| = 2^n\left(n!n^2 - \frac{n(2n-1)}{2}\right). \tag{5.9}$$

Die von diesen Formeln gelieferten Werte sind in Tabelle 5.2 für $n = 1$ bis $n = 20$ aufgelistet und in Abbildung 5.23 graphisch dargestellt. Daraus wird deutlich, dass eine explizite Darstellung des Beispiels nach wenig mehr Reihen nicht mehr zu realisieren ist.

n	Zustände	Transitionen	n	Zustände	Transitionen
1	2	1	11	$1,63499210 \times 10^{11}$	$9,89170213 \times 10^{12}$
2	12	20	12	$3,92398110 \times 10^{12}$	$2,82526639 \times 10^{14}$
3	88	372	13	$1,02023508 \times 10^{14}$	$8,62098649 \times 10^{15}$
4	752	5920	14	$2,85665824 \times 10^{15}$	$2,79952508 \times 10^{17}$
5	7648	95280	15	$8,56997473 \times 10^{16}$	$9,64122158 \times 10^{18}$
6	92096	1656768	16	$2,74239191 \times 10^{18}$	$3,51026165 \times 10^{20}$
7	1290112	31605056	17	$9,32413251 \times 10^{19}$	$1,34733714 \times 10^{22}$
8	20643584	660587520	18	$3,35668770 \times 10^{21}$	$5,43783408 \times 10^{23}$
9	371588608	15049320192	19	$1,27554132 \times 10^{23}$	$2,30235209 \times 10^{25}$
10	7431781376	371589022720	20	$5,10216531 \times 10^{24}$	$1,02043306 \times 10^{27}$

Tabelle 5.2: Größe des Zustandsraums und der Transitionsrelation für das Nim-Spiel mit n Reihen

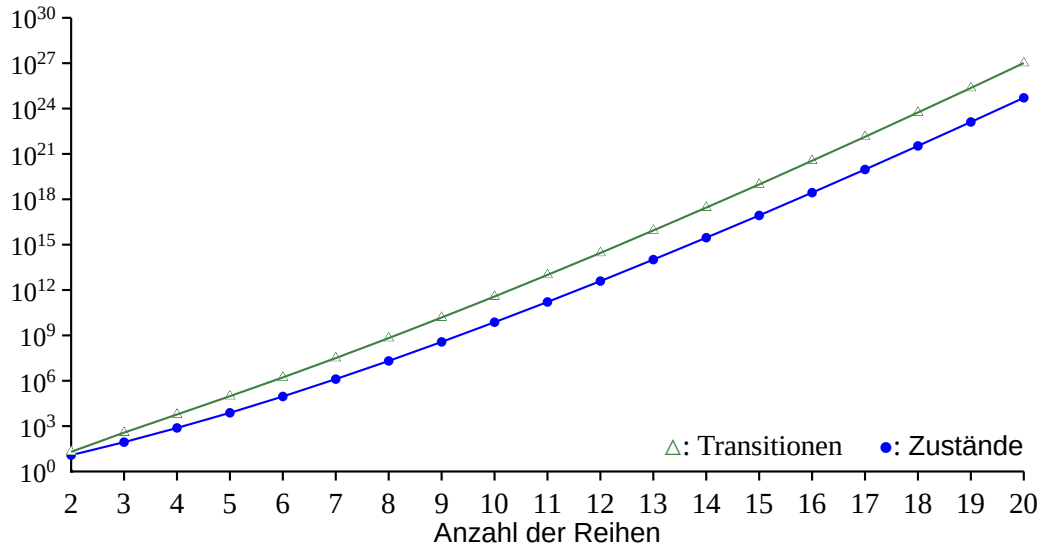


Abbildung 5.23: Größe des Zustandsraums und der Transitionsrelation für das Nim-Spiel

5.5.5 Testergebnisse

Die eigens entwickelten Testprogramme verwenden das in Abschnitt 5.4.4 bereits erwähnte Programmpaket CUDD. Es werden damit binäre Entscheidungsdiagramme für die symbolische Darstellung der Transitionsrelation und die Durchführung aller notwendigen Operationen benutzt. Implementiert wurden sowohl die Lösung nach dem Algorithmus von Kumar und Garg, d.h., nach Gleichungssystem 4.8, als auch der in diesem Kapitel vorgeschlagene neue Ansatz nach Gleichungssystem 5.7. Um die Vorteile der symbolischen Darstellung über die herkömmlichen Methoden zu verdeutlichen, wurde auch das Werkzeug LTCT[7] eingesetzt, das eine explizite Darstellung der Transitionsrelation verwendet.

Das Nim-Spiel wurde durch das Hinzufügen neuer Reihen schrittweise vergrößert, bis keiner der verfügbaren Ansätze mehr eine Lösung finden konnte. Die Zeitmessungen deuten jedoch darauf hin, dass bei den symbolischen Methoden weitere Fortschritte durch Optimierung des CUDD-Pakets zu erwarten sind. Da aber die erzielten Ergebnisse bereits ausreichen, um die Vorteile der neuen Methoden zu belegen, wurde diese Möglichkeit nicht weiter verfolgt.

Die Versuche wurden auf einem Standardrechner mit einem 1,8 GHz Athlon Prozessor und 1 GB RAM Hauptspeicher durchgeführt. Die verwendeten Programme und deren Versionen sind wie folgt: Betriebssystem SuSE Linux 8.1, GNU-Compiler g++ 3.2, CUDD-Paket 2.3.1, LTCT 2002.02.01. Die Laufzeit von LTCT konnte nur grob, die anderen Zeiten dagegen mit Unterstützung des Betriebssystems genauer gemessen werden. Tabelle 5.3 enthält Daten über die Laufzeiten, den Speicherbedarf und die Größe der Überwacher für die Spiele von 1 bis 12 Reihen. Die Laufzeiten werden in Sekunden angegeben, der Speicherbedarf nach der maximalen Anzahl der BDD-Knoten (1 Knoten = 16 Byte) gewertet.

In der ersten Spalte steht die Anzahl n der Reihen in dem Spiel, danach ist die Tabelle in vier Bereiche unterteilt. Der erste Bereich besteht aus einer Spalte, die die Laufzeiten für LTCT enthält. Auf Grund der Zustandsexplosion kam das Werkzeug mit der expliziten Darstellung nicht über fünf Reihen hinaus. Der nächste Bereich in der Tabelle umfasst die drei folgenden Spalten. Diese enthalten die Werte für die Lösung nach Kumar und Garg bzw. deren Variante aus Kapitel 2, die letztendlich zu Gleichungssystem 4.7 führt. Neben Laufzeit und BDD-Knoten wird auch die Anzahl der Iterationen angegeben, die für die Berechnung von u_{bn}^{∞} in den Formeln 4.8 notwendig sind. Diese Werte sind ein Maß dafür, wie oft zwischen dem größten und dem kleinsten Fixpunkt in Gleichungssystem 4.7 hin- und hergewechselt werden muss, weil neue verbotene Zustände immer wieder neue nicht co-erreichbare Zustände aufdecken und umgekehrt. Die Vorteile der symbolischen Darstellung werden deutlich, zumal das Beispiel nun bis zu elf Reihen gelöst werden kann, auch wenn der letzte Fall etwa drei Wo-

[7] Entwickelt in der Gruppe von Prof. Wonham an der University of Toronto

	LTCT	Gleichungssystem 4.7			Gleichungssystem 5.6		Größe des Überwachers	
n	Zeit	Zeit	Knoten	Iter.	Zeit	Knoten	Zustände	Transitionen
1	<< 1	$< 0,01$			$< 0,01$		2	1
2	<< 1	$< 0,01$			$< 0,01$		3	2
3	< 1	$< 0,01$			$< 0,01$		21	33
4	< 1	0,01	18396	7	0,01	17374	303	652
5	3	0,07	61320	11	0,04	48034	304	653
6		0,45	231994	17	0,10	114464	31936	107069
7		3,95	1234576	24	0,40	266742	442746	1524658
8		61,8	2498790	31	1,46	591738	9418751	38678520
9		1618	2587704	39	5,35	1211070	9418752	38678521
10		53106	6441666	49	49,2	2484482	2824840192	$1,59011.10^{10}$
11		$1,86.10^6$	32898180	58	1943	2474262	$8,17496.10^{10}$	$3,74686.10^{11}$
12					180262	3594374	$1,96199.10^{12}$	$9,45323.10^{12}$

Tabelle 5.3: Testergebnisse für das Nim-Spiel

chen Rechenzeit benötigt. Der dritte Bereich besteht aus den zwei nächsten Spalten und ist der neuen Lösung aus Abschnitt 5.2 gewidmet. Die neue Methode zur Suche nach nicht co-erreichbaren Zuständen verringert sowohl die Laufzeit als auch den Speicherbedarf. Der Fall mit elf Reihen kann nun in einer guten halben Stunde gelöst werden, darüber hinaus auch noch der Fall mit zwölf Reihen, also eine Größenordnung mehr bezüglich der Automaten für System, Spezifikation und Überwacher. Die Anzahl der Iterationen für die Berechnung von u_{bn}^{∞} ist stets gleich 1, weil das System keine verborgenen Verklemmungskomponenten hat. Der vierte Bereich enthält die Anzahl der erreichbaren Zustände und Transitionen für die jeweiligen Überwacher.

Mit Ausnahme des Überwachers für das Spiel mit drei Reihen sind alle Automaten zu groß, als dass sie hier wiedergegeben werden könnten. Die Gewinnstrategie für das Spiel mit drei Reihen ist jedoch in Abbildung 5.24 zu sehen und besagt z.B., dass der einzig richtige Zug zu Beginn des Spiels darin besteht, aus der dritten Reihe drei Streichhölzer zu entfernen.

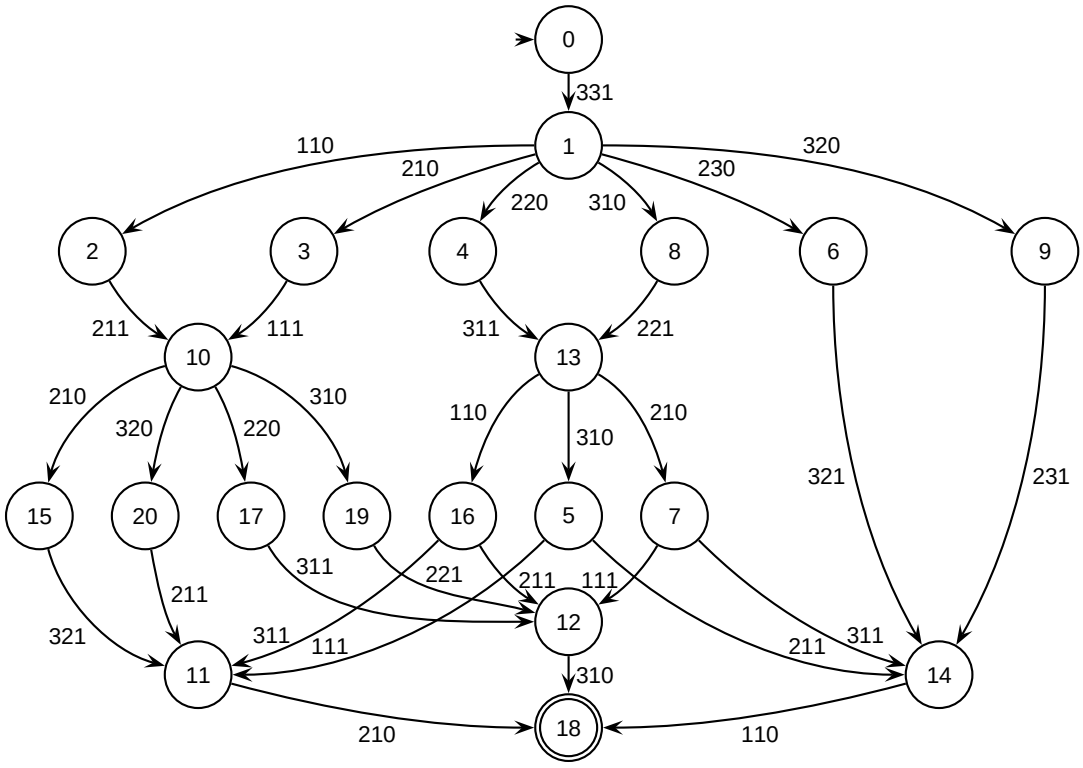


Abbildung 5.24: Der Überwacher für das Nim-Spiel mit drei Reihen

Kapitel 6

Verallgemeinerung der Überwachersynthese

In den Kapiteln 4 und 5 wurden bereits Verbesserungen hinsichtlich der Integration von Überwachersynthese und Modellprüfung erzielt. Durch die Einführung der symbolischen Verfahren in die Synthese konnte dem Problem der Zustandsexplosion entgegengewirkt werden, und die Effizienz der Berechnungen wurde durch einen neuen Algorithmus gesteigert. Unverändert blieb jedoch die Klasse der zugelassenen Spezifikationen, die nach wie vor auf Sicherheits- und Lebendigkeitseigenschaften beschränkt sind. Diese Einschränkung liegt daran, dass neben der immer notwendigen Steuerbarkeit die Co-Erreichbarkeit als festes Merkmal in die Formulierung des Überwachersyntheseproblems eingeht. In diesem Kapitel wird diese Forderung nach Co-Erreichbarkeit gelockert. An deren Stelle kann jede vom Anwender gewählte, im μ-Kalkül ausdrückbare Bedingung treten. Damit werden Fortdauer- und Fairnesseigenschaften in das Syntheseverfahren eingebracht, und zwar im Einklang mit den bisher erlaubten Sicherheits- und Lebendigkeitseigenschaften. Die Co-Erreichbarkeit selbst lässt sich als einfache μ-Kalkül-Formel ausdrücken, so dass das herkömmliche Verfahren zu einem Sonderfall des verallgemeinerten Ansatzes wird.

Wie in Abschnitt 2.7 erwähnt, haben Thistle und Wonham in [80, 81] einen Ansatz zur Behandlung von Fairnesseigenschaften vorgeschlagen, in dem sie die endlichen Automaten aus dem in Kapitel 2 beschriebenen Grundmodell durch ω-Automaten ersetzten. Beide Verfahren werden dadurch leider inkompatibel, denn einerseits kennen die endlichen Automaten keine Akzeptanzbedingungen für unendliche Wörter, andererseits haben die ω-Automaten keine endlichen Pfade. Eine weitere Hürde bei der Anwendung dieses Ansatzes liegt in der Schwierigkeit der Modellierung von Systemen mit ω-Automaten. In der hier vorgeschlagenen Verallgemeinerung werden nach wie vor die endlichen Automaten aus dem Grundmodell verwendet. Durch die Verallgemeinerung können jedoch sowohl endliche als auch unendliche Pfade gleichzeitig berücksichtigt werden, wie sie durchaus in realen Systemen vorkommen. Damit bildet auch das Verfahren von Thistle und Wonham einen Sonderfall des verallgemeinerten Ansatzes.

Als nächstes werden in Abschnitt 6.1 einige temporallogische Ausdrücke gegeben. Die hier vorgenommene Verallgemeinerung kommt zwar auch ohne den Einsatz von Temporallogiken aus, jedoch sind μ-Kalkül-Ausdrücke oft weniger gut lesbar als die äquivalenten temporallogischen Formulierungen. Da es Verfahren gibt (Emerson und Lei [32], Dam [23], Schneider [75]), mit denen temporallogische Ausdrücke in den μ-Kalkül übersetzt werden können, wird es der Anwender u.U. vorziehen, seine Spezifikationen zunächst in Temporallogik auszudrücken und dann in den μ-Kalkül zu übersetzen. Damit tragen temporallogische Ausdrücke dazu bei, die Kluft zwischen der menschlichen Intuition und der teilweise komplizierten Syntax der μ-Kalkül-Formeln und -Gleichungssysteme zu überbrücken. Abschnitt 6.2 stellt daraufhin das verallgemeinerte Überwachersyntheseproblem bzw. dessen Lösung vor. In den Abschnitten 6.3 und 6.4 wird gezeigt, dass sowohl das Grundmodell als auch der Ansatz von Thistle und Wonham Sonderfälle des verallgemeinerten Problems sind. Abschnitt 6.5 greift das Beispiel mit der Telefonnummernauskunft aus Abschnitt 5.4 nochmals auf und zeigt, wie der Fehler, der in Abschnitt 5.4.5 gefunden wurde, mit Hilfe des neuen Verfahrens behoben werden kann.

Weitere Arbeiten, die dem hier vorgestellten Ansatz nahe kommen, sind die von Jiang und Kumar [43], de Alfaro et al. [25] und Arnold et al. [2]. Diese Ansätze werden in Abschnitt 6.6 mit der hier vorgestellten Verallgemeinerung verglichen. Die Klasse der Probleme, die mit dem hier vorgestellten Verfahren gelöst werden

können, ist nahezu identisch mit der, die von den anderen Verfahren behandelt werden. Im Gegensatz zu diesen kann jedoch der hier vorgestellte Ansatz die Vorteile der symbolischen Darstellung nutzen und erreicht somit einen Reifegrad, der der Überwachersynthese bessere Chancen für die Anwendung im industriellen Bereich einräumt. Die Ergebnisse aus diesem Kapitel wurden in [92] und [95] veröffentlicht.

6.1 Temporallogische Ausdrücke

Temporallogische Ausdrücke beziehen sich auf Pfade einer Kripke-Struktur. Pfade können entweder endlich oder unendlich sein und werden gemäß folgender Definition unterschieden:

Definition 6.1 (Pfade) *Ein unendlicher Pfad der Kripke-Struktur* $\langle \mathcal{S}, \mathcal{I}, \mathcal{R}, \mathcal{L} \rangle$ *ist eine unendliche Folge von Zuständen* $(s_0, s_1, \ldots)$ *aus* $\mathcal{S}$*, so dass* $\forall i \geq 0.(s_i, s_{i+1}) \in \mathcal{R}$*.*
Ein endlicher Pfad der selben Struktur ist eine endliche Folge von Zuständen $(s_0, \ldots, s_n)$*, so dass* $\neg\exists s' \in \mathcal{S}.(s_n, s') \in \mathcal{R}$ *und, falls* $n > 0$*, dann* $\forall i \in \{0, \ldots, n-1\}.(s_i, s_{i+1}) \in \mathcal{R}$*.*

Aus dieser Definition folgen:

- Ein Pfad kann in jedem Zustand beginnen (der Zustand s_0 bezeichnet den Anfangszustand eines Pfades, und nicht etwa einen Initialzustand der Kripke-Struktur).
- Pfade sind immer maximal in dem Sinne, dass die Folge der Zustände nicht unterbrochen werden darf, solange von dem bisher letzten Zustand noch Transitionen ausgehen. Also sind endliche Präfixe eines unendlichen Pfades bzw. echte Präfixe eines endlichen Pfades *keine* Pfade.
- Hingegen ist jedes Suffix eines endlichen bzw. unendlichen Pfades auch ein Pfad.

Beispiel 6.2 Die Kripke-Struktur in Abbildung 3.1 auf Seite 32 hat endliche und unendliche Pfade. Endlich sind alle, die in Zustand 6 enden. Die Kripke-Struktur in Abbildung 5.21 auf Seite 95 hat nur endliche Pfade. □

In den hier vorgestellten Formeln kommen der existentielle Pfadquantor E und der universelle Pfadquantor A vor. Eine Formel, die einen dieser Quantoren enthält, wird von einem Zustand s einer Kripke-Struktur erfüllt, wenn der Ausdruck, der dem Quantor folgt, auf mindestens einem bzw. auf allen Pfaden, die in s beginnen, gilt. Die Pfadquantoren treten zusammen mit den Zeitoperatoren X, F, G, U und $\underline{\mathsf{U}}$ auf, die dazu dienen, Pfade zu beschreiben. Die drei ersten bedeuten, dass der darauf folgende Ausdruck entweder im nächsten (X), auf mindestens einem (F) Zustand oder auf allen (G) Zuständen der mit E bzw. A quantifizierten Pfade erfüllt sein muss. Der Operator U (aus dem Englischen *until*) verknüpft zwei Ausdrücke in der Form $[\varphi\ \mathsf{U}\ \psi]$ und gilt auf Pfaden, auf denen φ solange gilt, wie ψ nicht gilt. Hierbei muss ein Zustand, in dem ψ gilt, nicht unbedingt erreicht werden. Dann wird die Bedingung von Pfaden erfüllt, auf denen φ immer gilt. Dieses unendliche Warten auf das Erfüllen von ψ wird von der strengeren Variante $[\varphi\ \underline{\mathsf{U}}\ \psi]$ unterbunden, die verlangt, dass ein Zustand, in dem ψ gilt, auch irgendwann erreicht wird.

Die Definition einer Temporallogik legt fest, nach welchen Regeln die Pfadquantoren und Zeitoperatoren zusammengesetzt werden können. Daraus ergeben sich Logiken unterschiedlicher Mächtigkeit, wie z.B. CTL *(computer tree logic)* oder CTL*. Eine formale Beschreibung der Syntax und der Semantik von CTL, CTL* und anderen Logiken findet der Leser z.B. in den Werken von Clarke et al. [20] oder Schneider [75]. Weiterhin beschreiben diese Referenzen auch, wie einige der Operatoren mit Hilfe anderer ausgedrückt werden können. So gilt z.B. $\mathsf{EF}\psi \Leftrightarrow \mathsf{E}[1\ \underline{\mathsf{U}}\ \psi]$.

In Bezug auf endliche und unendliche Pfade wird noch folgende Notation für die Zeitoperatoren F und G vereinbart:

- F und G beziehen sich sowohl auf endliche als auch auf unendliche Pfade. Dies bedeutet, dass der gegebene temporallogische Ausdruck als erfüllt betrachtet wird, unabhängig davon, ob der Pfad, auf dem dies geschieht, endlich oder unendlich ist.

- F_∞ und G_∞ beziehen sich ausschließlich auf unendliche Pfade. Dadurch ergibt sich ein subtiler Unterschied zwischen existentiell und universell quantifizierten Ausdrücken. Steht vor F_∞ bzw. G_∞ der Pfadquantor E, so bedeutet dies, dass jeder Zustand, der von dem Ausdruck erfasst wird, einen unendlichen Pfad haben muss, auf dem die gegebene Eigenschaft zutrifft. Folglich sind Zustände, die nur endliche Pfade haben, niemals Teil der erfassten Zustandsmenge. Wird der Ausdruck andererseits durch den Pfadquantor A eingeleitet, erfüllen die Zustände, die keine unendlichen Pfade haben, die Quantifizierung „auf *allen* unendlichen Pfaden“ trivialerweise, und sind deshalb Teil der Lösungsmenge.
- $\mathsf{F}_\dashv$ und $\mathsf{G}_\dashv$ beziehen sich ausschließlich auf endliche Pfade. Auch hier ergibt sich ein Unterschied zwischen existentiell und universell quantifizierten Ausdrücken. Existentiell quantifizierte Ausdrücke schließen Zustände aus, von denen kein endlicher Pfad ausgeht, während universell quantifizierte Ausdrücke diese Zustände trivialerweise mit einbeziehen.

Es folgt eine Sammlung temporallogischer Ausdrücke aus der Modellprüfung mit Übersetzung in Gleichungssysteme des vektoriellen μ-Kalküls. Damit wird der Weg für die verallgemeinerte Überwachersynthese in Abschnitt 6.2 vorbereitet, die diese Gleichungssysteme verwendet. Die Übersetzungen stammen teilweise aus [75], andere wurden ohne großen Aufwand daraus abgeleitet. Die meisten μ-Kalkül-Ausdrücke in den Gleichungssystemen lassen sich intuitiv nachvollziehen. Dazu sind neben Lemma 3.25 auf Seite 39 die Gleichungen

$$u \overset{\mu}{=} \Box u \tag{6.1}$$

und

$$u \overset{\nu}{=} \Diamond u \tag{6.2}$$

hilfreich. Sie berechnen die Mengen der Zustände, von denen nur endliche bzw. nur unendliche Pfade ausgehen.

— $\mathsf{EG}\varphi$ beschreibt die Menge der Zustände, von denen mindestens ein Pfad ausgeht, auf dessen Zuständen der Ausdruck φ stets erfüllt ist. Diese Zustandsmenge ist die Lösung des Gleichungssystems

$$\left\{ \; u \overset{\nu}{=} \varphi \wedge (\Box 0 \vee \Diamond u). \right. \tag{6.3}$$

— $\mathsf{EG}_\infty\,\varphi$ ist die Einschränkung des letzten Ausdrucks auf unendliche Pfade. Dies entspricht dem Gleichungssystem

$$\left\{ \; u \overset{\nu}{=} \varphi \wedge \Diamond u. \right. \tag{6.4}$$

— $\mathsf{EG}_\dashv\,\varphi$ ist die Einschränkung des letzten Ausdrucks auf endliche Pfade. Dies entspricht dem Gleichungssystem

$$\left\{ \; u \overset{\mu}{=} \varphi \wedge (\Box 0 \vee \Diamond u). \right. \tag{6.5}$$

— $\mathsf{EF}\varphi$ beschreibt die Menge der Zustände, von denen mindestens ein Pfad ausgeht, der zu einem Zustand führt, auf dem φ gilt. Ein solcher Zustand wird *irgendwann* auf den beschriebenen Pfaden erreicht. Das entsprechende μ-Kalkül-Gleichungssystem ist

$$\left\{ \; u \overset{\mu}{=} \varphi \vee \Diamond u. \right. \tag{6.6}$$

— $\mathsf{EF}_\infty\varphi$ ist die Einschränkung des letzten Ausdrucks auf unendliche Pfade und entspricht dem Gleichungssystem

$$\begin{cases} u_2 \overset{\nu}{=} \Diamond u_2 \\ u_1 \overset{\mu}{=} \varphi \wedge u_2 \vee \Diamond u_1. \end{cases} \tag{6.7}$$

Hier steht u_2 für die Menge der Zustände, von denen gemäß Gleichung 6.2 ein unendlicher Pfad ausgeht, so dass u_1 auf die Menge dieser Zustände eingeschränkt wird.

— $\mathsf{EF}_{\dashv}\varphi$ ist die Einschränkung von $\mathsf{EF}\varphi$ auf endliche Pfade und entspricht dem Gleichungssystem

$$\begin{cases} u_2 & \overset{\mu}{=} & \Box u_2 \\ u_1 & \overset{\mu}{=} & u_2 \wedge (\varphi \vee \Diamond u_1). \end{cases} \tag{6.8}$$

Hier steht u_2 für die Menge der Zustände, von denen gemäß Gleichung 6.1 nur endliche Pfade ausgehen, so dass u_1 auf die Menge dieser Zustände eingeschränkt wird.

— $\mathsf{AG}\varphi$ beschreibt die Menge der Zustände, deren ausgehenden Pfade nur aus Zuständen bestehen, auf denen φ gilt. Das Gleichungssystem dazu ist

$$\{ \; u \overset{\nu}{=} \varphi \wedge \Box u. \tag{6.9}$$

— $\mathsf{AG}_\infty\,\varphi$ ist die Einschränkung des letzten Ausdrucks auf unendliche Pfade und führt zu dem Gleichungssystem

$$\begin{cases} u_2 & \overset{\nu}{=} & \Diamond u_2 \\ u_1 & \overset{\nu}{=} & \varphi \wedge u_2 \wedge \Box u_1. \end{cases} \tag{6.10}$$

— $\mathsf{AF}\varphi$ beschreibt die Menge der Zustände, deren ausgehenden Pfade alle mindestens einen Zustand aufweisen, in dem φ gilt. Auf allen Pfaden gilt also *irgendwann* φ. Das entsprechende Gleichungssystem ist

$$\{ \; u \overset{\mu}{=} \varphi \vee \Diamond 1 \wedge \Box u.$$

— $\mathsf{AF}_\infty\varphi$ ist die Einschränkung des letzten Ausdrucks auf unendliche Pfade und entspricht dem Gleichungssystem

$$\{ \; u \overset{\mu}{=} \varphi \vee \Box u.$$

— $\mathsf{E}[\varphi \,\underline{\mathsf{U}}\, \psi]$ gilt in allen Zuständen, die einen Pfad haben, auf dessen Zuständen φ solange gilt, bis irgendwann ein Zustand erreicht wird, in dem ψ gilt. In diesem Zustand muss φ dann nicht mehr gelten. Damit gilt der Ausdruck trivialerweise in allen Zuständen, in denen ψ gilt. Eine äquivalente Interpretation ist das φ gilt, solange ψ nicht gilt. Das Gleichungssystem dazu ist

$$\{ \; u \overset{\mu}{=} \psi \vee \varphi \wedge \Diamond u. \tag{6.11}$$

— $\mathsf{E}[\varphi \,\underline{\mathsf{U}}_\infty\, \psi]$ ist die Variante von $\mathsf{E}[\varphi \,\underline{\mathsf{U}}\, \psi]$ für unendliche Pfade:

$$\begin{cases} u_2 & \overset{\nu}{=} & \Diamond u_2 \\ u_1 & \overset{\mu}{=} & \psi \wedge u_2 \vee \varphi \wedge \Diamond u_1. \end{cases} \tag{6.12}$$

Wie bei Gleichungssystem 6.7 steht u_2 für die Menge der Zustände, von denen ein unendlicher Pfad ausgeht, so dass u_1 auf die Menge dieser Zustände eingeschränkt wird.

— $\mathsf{E}[\varphi \,\mathsf{U}\, \psi]$ ist die abgeschwächte Variante von $\mathsf{E}[\varphi \,\underline{\mathsf{U}}\, \psi]$, die auch solche Pfade zulässt, auf denen ψ nie erfüllt wird, dafür aber φ immer gilt. Die Gleichung dazu ist

$$\{ \; u \overset{\nu}{=} \psi \vee \varphi \wedge \Diamond u.$$

— $\mathsf{E}[\varphi \,\mathsf{U}_\infty \psi]$ ist die Einschränkung des letzten Ausdrucks auf unendliche Pfade. Das Gleichungssystem dazu lässt sich aus $\mathsf{E}[\varphi \,\mathsf{U}_\infty \psi] = \mathsf{E}[\varphi \,\underline{\mathsf{U}}_\infty\, \psi] \vee \mathsf{EG}_\infty\, \varphi$, also aus den Gleichungen 6.4 und 6.12 ableiten:

$$\begin{cases} u_4 & \overset{\nu}{=} & \varphi \wedge \Diamond u_4 \\ u_3 & \overset{\nu}{=} & \Diamond u_3 \\ u_2 & \overset{\mu}{=} & \psi \wedge u_3 \vee \varphi \wedge \Diamond u_2 \\ u_1 & \overset{\mu}{=} & u_2 \vee u_4. \end{cases}$$

— $\mathsf{EFG}\varphi$ gilt in Zuständen, von denen ein Pfad ausgeht, auf dem ab einem bestimmten Zustand φ immer gilt. Das entsprechende Gleichungssystem ist

$$\begin{cases} u_2 & \overset{\nu}{=} & \varphi \wedge (\Box 0 \vee \Diamond u_2) \\ u_1 & \overset{\mu}{=} & u_2 \vee \Diamond u_1. \end{cases}$$

Wie in Gleichung 6.3 berechnet u_2 die Menge der Zustände, die einen Pfad haben, auf dem φ immer gilt. u_1 berechnet dann die Zustände, die einen Pfad zu einem Zustand in u_2 haben.

— $\mathsf{EFG}_\infty\,\varphi$ ist die Einschränkung des letzten Ausdrucks auf unendliche Pfade. Das Gleichungssystem dazu entsteht indem die Gleichung für u_2 gemäß Gleichung 6.4 angepasst wird:

$$\begin{cases} u_2 & \overset{\nu}{=} & \varphi \wedge \Diamond u_2 \\ u_1 & \overset{\mu}{=} & u_2 \vee \Diamond u_1. \end{cases}$$

— Folgendes Gleichungssystem berechnet die Menge der Zustände, von denen aus mindestens zweimal ein Zustand erreicht werden kann, auf dem ψ gilt, wobei dazu nur Zustände durchlaufen werden dürfen, auf denen φ gilt. Dazu wird die Gleichung zu $\mathsf{E}[\varphi\,\underline{\mathsf{U}}\,\psi]$ zweimal verwendet:

$$\begin{cases} u_3 & \overset{\mu}{=} & \psi \vee \varphi \wedge \Diamond u_3 \\ u_2 & \overset{\mu}{=} & \psi \wedge \varphi \wedge \Diamond u_3 \\ u_1 & \overset{\mu}{=} & u_2 \vee \varphi \wedge \Diamond u_1. \end{cases}$$

Die Gleichung für u_3 berechnet $\mathsf{E}[\varphi\,\underline{\mathsf{U}}\,\psi]$. Danach werden in u_2 die existentiellen Vorgängerzustände dieser Menge festgehalten, auf denen $\varphi \wedge \psi$ gilt. Die Gleichung für u_1 berechnet dann $\mathsf{E}[\varphi\,\underline{\mathsf{U}}\,u_2]$ und damit die Menge der Zustände, von denen ein Pfad ausgeht, auf dem φ gilt, bis ein Zustand erreicht wird, auf dem $\varphi \wedge \psi$ gilt und der einen Pfad hat, auf dem φ weiterhin gilt, bis ψ ein zweites Mal erfüllt wird.

— Die Variante des letzten Gleichungssystems für unendliche Pfade kann mit Hilfe von Gleichung 6.2 erstellt werden:

$$\begin{cases} u_4 & \overset{\nu}{=} & \Diamond u_4 \\ u_3 & \overset{\mu}{=} & \psi \wedge u_4 \vee \varphi \wedge \Diamond u_3 \\ u_2 & \overset{\mu}{=} & \psi \wedge \varphi \wedge \Diamond u_3 \\ u_1 & \overset{\mu}{=} & u_2 \vee \varphi \wedge \Diamond u_1. \end{cases}$$

— $\mathsf{EGF}_\infty\varphi$ gilt in Zuständen, die einen Pfad haben, dessen Zustände alle unendliche Pfade haben, auf denen irgendwann φ gilt. Dies ist äquivalent zu der Aussage, dass der Ausdruck die Zustände beschreibt, von denen aus es unendlich oft möglich ist, zu einem Zustand zu gelangen, in dem φ gilt. Das Gleichungssystem dazu ist

$$\begin{cases} u_2 & \overset{\mu}{=} & u_1 \wedge \varphi \vee \Diamond u_2 \\ u_1 & \overset{\nu}{=} & \Diamond u_2. \end{cases}$$

— $\mathsf{E}[\mathsf{G}_\infty\,\beta \wedge \mathsf{GF}_\infty\varphi]$ erweitert den letzten Ausdruck mit der Bedingung, dass auf allen Zuständen des Pfades zusätzlich der Ausdruck β gilt. Das Gleichungssystem dazu ist

$$\begin{cases} u_2 & \overset{\mu}{=} & u_1 \wedge \varphi \vee \beta \wedge \Diamond u_2 \\ u_1 & \overset{\nu}{=} & \beta \wedge \Diamond u_2. \end{cases}$$

— $\mathsf{E}\left[\mathsf{G}_\infty\,\beta \wedge \bigwedge_{i=1}^{n} \mathsf{GF}_\infty\varphi_i\right]$ erweitert die vorangegangenen Ausdrücke um unterschiedliche Mengen φ_i, die alle unendlich oft besucht werden müssen. Die Übersetzung in den μ-Kalkül ergibt

$$\begin{cases} u_n & \stackrel{\mu}{=} & w \wedge \varphi_n \vee \beta \wedge \Diamond u_n \\ \vdots & & \\ u_1 & \stackrel{\mu}{=} & w \wedge \varphi_1 \vee \beta \wedge \Diamond u_1 \\ w & \stackrel{\nu}{=} & \beta \wedge \bigwedge_{i=1}^{n} \Diamond u_i. \end{cases} \tag{6.13}$$

— Die Akzeptanzbedingung für Rabin-Automaten [30, 33, 75] ist $\mathsf{E}\bigwedge_{j=1}^{n} \mathsf{GF}_\infty\varphi_j \vee \mathsf{FG}_\infty\,\psi_j$. Sie lässt sich in folgendes Gleichungssystem übersetzen:

$$\begin{cases} u_n & \stackrel{\mu}{=} & v \wedge \varphi_n \vee v \wedge \Diamond u_n \\ \vdots & & \\ u_1 & \stackrel{\mu}{=} & v \wedge \varphi_1 \vee v \wedge \Diamond u_1 \\ v & \stackrel{\nu}{=} & \bigwedge_{j=1}^{n} \Diamond u_j \vee \psi_j \wedge \Diamond v \\ w & \stackrel{\mu}{=} & v \vee \Diamond w. \end{cases} \tag{6.14}$$

Die Komplexität der Übersetzung temporallogischer Ausdrücke in den μ-Kalkül ist für CTL linear in der Größe der Formel, für CTL* exponentiell. Letztere liegt damit über der polynomiellen Komplexität des μ-Kalküls und könnte zu der Argumentation führen, dass das hier vorgestellte Verfahren nicht polynomieller, sondern exponentieller Natur ist. Da jedoch die Formeln in der Regel kurz sind, fällt die Übersetzung nicht besonders ins Gewicht. Der polynomielle Teil ist der schwierigste, weil die Transitionsrelationen oft sehr groß werden. Außerdem muss die Übersetzung nur einmal vorgenommen werden und ist unabhängig von dem Problem, das gelöst werden muss. Die zentrale Aufgabe bleibt deshalb nach wie vor die Lösung eines Gleichungssystems, unabhängig davon, mit welchen Hilfsmitteln dieses erstellt wird.

6.2 Das verallgemeinerte Überwachersyntheseproblem

In diesem Abschnitt wird das Überwachersyntheseproblem aus Kapitel 2 mit Hilfe des μ-Kalküls verallgemeinert. Der daraus entstehende Ansatz kann die Stärken beider Verfahren gleichzeitig nutzen und eröffnet dem Anwender dadurch neue Möglichkeiten für die Spezifikation und die Implementierung diskreter Systeme.

6.2.1 Motivation und Formulierung

In der konventionellen Überwachersynthese, die in den vergangenen Kapiteln behandelt wurde, kann mit Sicherheits- und Lebendigkeitseigenschaften gearbeitet werden, nicht aber mit Fortdauer- und Fairnesseigenschaften. Der Grund dafür ist, dass diese Eigenschaften nicht etwa durch eine geschickte Auswahl der Automaten spezifiziert werden können, sondern erst durch eine passende Berechnung der Zustände während der Synthese einzubringen sind. Die konventionellen Synthesealgorithmen sind jedoch fest auf die Steuerbarkeit und die Co-Erreichbarkeit eingestellt und lassen deshalb keine anderen Berechnungen zu.

Ein Beispiel für eine Fairnessbedingung ist die Forderung, dass sich ein Benutzer eines Systems an diesem immer wieder – also theoretisch unendlich oft – anmelden können soll. Mit der konventionellen Synthese könnte die Co-Erreichbarkeit sicherstellen, dass die Anmeldung mindestens einmal möglich ist. Die Spezifikation kann jedoch Pfade enthalten, auf denen die Anmeldung nur endlich viele Male möglich ist, oder es können während der Synthese solche Pfade entstehen. Die konventionellen Algorithmen können dies nicht feststellen und lassen auch solche Zustände zu, von denen aus es nur endlich viele Male möglich ist, sich anzumelden.

Die Co-Erreichbarkeit ist also nicht immer ausreichend, um das gewünschte Systemverhalten zu erzielen. Unter Umständen müssen an ihre Stelle strengere Forderungen treten. Letztendlich soll jede Spezifikationseigenschaft zugelassen werden, die wie jene in Abschnitt 6.1 als Gleichungssystem im μ-Kalkül ausgedrückt werden kann. Für die Berechnung der Zustände mit einer solchen Eigenschaft wird – wie bei der Co-Erreichbarkeit auch – die gesamte Transitionsrelation benötigt. Folglich werden die Zustände mit der neuen Eigenschaft auf der rechten Hälfte der Kripke-Struktur $\mathcal{K}_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}$ berechnet. Hingegen wird die Forderung nach Steuerbarkeit nach wie vor als Existenzbedingung für den Überwacher gesehen und bleibt unangetastet, womit sich an den Berechnungen auf der linken Hälfte der Kripke-Struktur prinzipiell nichts ändert.

Startpunkt für den verallgemeinerten Ansatz zur Überwachersynthese ist die Lösung des Überwachersyntheseproblems in Korollar 4.14 auf Seite 73, die zu dem Gleichungssystem

$$\begin{cases} u_c & \overset{\mu}{=} & (\Diamond u_c \vee x_m \overline{x_u}) \wedge u_{cg} \\ u_g & \overset{\nu}{=} & \Box u_g \wedge \overline{x_b} x_u \wedge \kappa(u_c) \\ u_{cg} & \overset{\nu}{=} & \kappa\left(u_g\right) \end{cases} \tag{6.15}$$

führt, in dem die Variablen u_c und u_g die Mengen der co-erreichbaren bzw. der guten (nicht verbotenen) Zustände berechnen.

Die Forderung nach Co-Erreichbarkeit wird nun durch eine beliebige, von dem Anwender wählbare Bedingung in Form eines Paares (E, ϕ) ersetzt. Hier ist E ein μ-Kalkül-Gleichungssystem und ϕ eine boolesche Formel, in der die Variablen auf der linken Seite von E nicht in gebundener Form vorkommen (s. Abschnitt 3.3.1). In der folgenden Definition wird $\phi := u_1$ gesetzt, wobei u_1 die Variable auf der linken Seite der untersten Gleichung in E ist. Damit wird festgelegt, dass die Menge der gewünschten Zustände durch diese Gleichung darzustellen ist. Dies ist jedoch keine Einschränkung des allgemeinen Falls mit der Form (E, ϕ). Sollte für die gewünschte Eigenschaft eine boolesche Formel wie $\phi = \psi(u_1, \ldots, u_n)$ notwendig sein, so genügt es, die Formeln für $u_1, \ldots, u_n$ in E neu zu nummerieren und die Formel ϕ als $u_1 \overset{\sigma}{=} \psi(u_1, \ldots, u_n)$ unter die anderen zu schreiben. Dabei kommt u_1 in den anderen Formeln nicht vor und kann auf Grund der Abhängigkeiten zuletzt und nur ein Mal berechnet werden.

Definition 6.3 (Verallgemeinertes Überwachersyntheseproblem) *Gegeben seien ein System $\mathcal{A}_\mathcal{P}$ und ein Automat $\mathcal{A}_\mathcal{E}$ zur Erstellung einer Spezifikation, so dass $K := \mathsf{L}_M(\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}) \subseteq \mathsf{L}_M(\mathcal{A}_\mathcal{P})$ das gewünschte Verhalten des überwachten Systems ausdrückt. Gegeben sei ebenfalls ein Paar (E, u_1), wobei E ein μ-Kalkül-Gleichungssystem der Form*

$$\begin{cases} u_n & \overset{\sigma_n}{=} & \Phi_n \\ & \vdots & \\ u_1 & \overset{\sigma_1}{=} & \Phi_1 \end{cases} \tag{6.16}$$

ist und zusätzliche temporallogische Eigenschaften für das überwachte System vorschreibt. Gesucht ist ein Überwacher $\mathcal{A}_\mathcal{S}$, der diese Eigenschaften gewährleistet, so dass $\mathsf{L}_M(\mathcal{A}_\mathcal{S}) \subseteq K$.

Definition 6.3 ist an die Formulierung des konventionellen Überwachersyntheseproblems in Definition 2.16 angelehnt. Wie dort, bilden auch hier die Automaten $\mathcal{A}_\mathcal{P}$ und $\mathcal{A}_\mathcal{E}$ den Ausgangspunkt. In dem konventionellen Syntheseproblem werden außerdem implizit zwei weitere Forderungen gestellt, nämlich nach Co-Erreichbarkeit, was dort gleichbedeutend mit der Verklemmungsfreiheit ist, und nach Steuerbarkeit, eine Existenzbedingung für Überwacher. Davon wird die erste nun vom Anwender bestimmt. Dazu muss er die gewünschte Bedingung als μ-Kalkül-Gleichungssystem formulieren, was z.B. unter Zuhilfenahme von temporallogischen Ausdrücken wie die in Abschnitt 6.1 geschehen kann. Die Forderung nach Steuerbarkeit bleibt unangetastet. In beiden Fällen gilt: Falls die Spezifikation $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ die gewünschten Eigenschaften bereits erfüllt, dient sie als Überwacher, ansonsten wird mit Hilfe der Syntheseprozeduren ein solcher berechnet.

Des Weiteren verzichtet der verallgemeinerte Ansatz aus den in Abschnitt 2.8 erläuterten Gründen auf die Spezifikation eines kleinsten akzeptablen Verhaltens A_{min}. Statt dessen eröffnen sich hier zwei neue Möglichkeiten für den Anwender. Einerseits kann er die Kripke-Struktur $\mathcal{K}_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}$ nutzen, um das Ergebnis der Synthese mit

Hilfe der Modellprüfung auf gewünschte Eigenschaften zu prüfen. Damit kommt die Modellprüfung der Synthese zu Hilfe, um zu verifizieren, ob die Einschränkungen, die der Überwacher dem System aufzwingt, akzeptabel sind. Die andere Möglichkeit besteht darin, die zu prüfenden Eigenschaften bereits in das vom Anwender formulierte Gleichungssystem einzubauen. Dann erfüllt das Syntheseergebnis entweder die Anforderungen, die ansonsten noch nachgeprüft werden müssten, oder es besteht aus der leeren Menge, falls die Anforderungen nicht einzuhalten sind. Diese Zusammenführung von Überwachersynthese und Modellprüfung stellt den wichtigsten Beitrag dieser Arbeit zu den bestehenden Spezifikations- und Implementierungstechniken für Systeme mit endlichen diskreten Zustandsräumen dar.

6.2.2 Lösung des verallgemeinerten Überwachersyntheseproblems

Die Lösung des verallgemeinerten Überwachersyntheseproblems besteht lediglich in der Anpassung des Gleichungssystems 6.15 an die in Abschnitt 6.2.1 erläuterten Änderungen. Die Menge der Zustände mit den vom Anwender gewünschten Eigenschaften tritt an die Stelle der co-erreichbaren Zustände. Damit tritt die Variable u_1 an die Stelle von u_c in der Gleichung für u_g. Die Gleichung für u_c wird durch das Gleichungssystem 6.16 ersetzt, wobei zu beachten ist, dass dessen Gleichungen nur gute Zustände sammeln dürfen und auf der rechten Hälfte der Kripke-Struktur zu berechnen sind. Sie werden deshalb um eine Konjunktion mit $\overline{x_u}$ und mit der Menge der guten Zustände ergänzt. Zuletzt wird noch die Notation angepasst, indem die Variable u_{cg} in z umbenannt wird, so dass die Lösung nun aus dem Fixpunkt z^∞ besteht. Aus diesen Überlegungen entsteht das Gleichungssystem 6.17 im folgenden Theorem:

Theorem 6.4 (Verallgemeinerte Überwachersynthese) *Gegeben seien die Automaten $\mathcal{A}_\mathcal{P}$ und $\mathcal{A}_\mathcal{E}$ und eine gewünschte Eigenschaft als Gleichungssystem der Form 6.16. Sei $\mathcal{A}_\mathcal{S} = (\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E})|_{Q_g}$ der Überwacher, unter dessen Steuerung die Menge der erreichbaren Zustände so eingeschränkt wird, dass die gewünschte Eigenschaft stets eingehalten wird. Dann ist $Q_g = [\![\exists x_u.z^\infty]\!]_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}$, wobei z^∞ die Lösung des Gleichungssystems*

$$\left\{\begin{array}{lcl} u_n & \overset{\sigma_n}{=} & \Phi_n \wedge \overline{x_u} \wedge z \\ & \vdots & \\ u_1 & \overset{\sigma_1}{=} & \Phi_1 \wedge \overline{x_u} \wedge z \\ u_g & \overset{\nu}{=} & \Box u_g \wedge \overline{x_b} x_u \wedge \kappa(u_1) \\ z & \overset{\nu}{=} & \kappa\left(u_g\right) \end{array}\right. \tag{6.17}$$

über den Variablen der Kripke-Struktur $\mathcal{K}_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}$ ist, die mit Hilfe der Formeln

$$\left[\begin{array}{lcl} z^{i_z+1} & := & \kappa\left(u_g^{i_z,\infty}\right) \\ u_g^{i_z,i_g+1} & := & \Box u_g^{i_z,i_g} \wedge \overline{x_b} x_u \wedge \kappa\left(u_1^{i_z,0,\infty}\right) \\ u_1^{i_z,i_g,i_1+1} & := & [\Phi_1]_{u_1,\ldots,u_n}^{u_1^{i_z,i_g,i_1},\ldots,u_n^{i_z,i_g,i_1,\ldots,i_n}} \wedge \overline{x_u} \wedge z \\ & \vdots & \\ u_n^{i_z,i_g,i_1,\ldots,i_n+1} & := & [\Phi_n]_{u_1,\ldots,u_n}^{u_1^{i_z,i_g,i_1},\ldots,u_n^{i_z,i_g,i_1,\ldots,i_n}} \wedge \overline{x_u} \wedge z \end{array}\right. \tag{6.18}$$

berechnet werden kann.

Beweis: Das Theorem geht aus Korollar 4.14 und der Argumentation in Abschnitt 6.2.1 hervor. □

Für die Berechnungen werden die Anfangswerte wie in Algorithmus 3.57 gewählt und nur bei Alternierung der Fixpunktart zurückgesetzt. Die Komplexität der Berechnungen hängt von den benutzerdefinierten Formeln Φ_i, $i = 1, \ldots, n$ ab und lässt sich mit den in Kapitel 3 diskutierten Methoden ermitteln. Die Notation $[\Phi_i]_{u_1,\ldots,u_n}^{u_1^{i_z,i_g,i_1},\ldots,u_n^{i_z,i_g,i_1,\ldots,i_n}}$ bedeutet lediglich, dass die Variablen $u_1, \ldots, u_n$, die in den Formeln Φ_i vorkommen, mit den entsprechenden Iterationszählern zu versehen sind.

Korollar 6.5 *Um Iterationen zu sparen, kann die Berechnung der Formeln 6.18 bereits dann abgebrochen werden, wenn für* $i_z \geq 1$ $u_1^{i_z,0,\infty} = \kappa\left(u_g^{i_z-1,\infty}\right)$. *Es gilt dann* $z^\infty = u_1^{i_z,0,\infty}$.

Beweis: Sei $i_z \geq 1$ der Wert für den gilt, dass $u_1^{i_z,0,\infty} = \kappa\left(u_g^{i_z-1,\infty}\right)$. Dann ist $u_g^{i_z,1} = \Box 1 \wedge \overline{x_b} x_u \wedge u_g^{i_z-1,\infty} = u_g^{i_z-1,\infty}$ und somit ein Fixpunkt, woraus folgt $u_g^{i_z,\infty} = u_g^{i_z-1,\infty}$. Damit ist auch $z^\infty = \kappa\left(u_g^{i_z-1,\infty}\right) = u_1^{i_z,0,\infty}$. □

Es folgen drei Anwendungsbeispiele der verallgemeinerten Überwachersynthese. Im ersten wird gezeigt, dass das Überwachersyntheseproblem von Ramadge und Wonham ein Sonderfall des verallgemeinerten Ansatzes ist. In dem zweiten Beispiel wird die Forderung nach Co-Erreichbarkeit durch eine Fairnessbedingung ersetzt. Das dritte greift das Beispiel aus Abschnitt 5.4 wieder auf und behebt den in Abschnitt 5.4.5 gefundenen Schönheitsfehler.

6.3 Die konventionelle Überwachersynthese als Sonderfall

Um das Überwachersyntheseproblem aus Definition 2.16 im verallgemeinerten Ansatz auszudrücken genügt es, die Co-Erreichbarkeit als die Bedingung zu wählen, die mit Hilfe eines μ-Kalkül-Gleichungssystems darzustellen ist. Diese kann mit Hilfe der temporallogischen Formel $\mathsf{EF}x_m$ dargestellt werden. Letztere berechnet die Menge der Zustände, die einen Pfad zu einem markierten Zustand haben. Da der Überwacher durch Einschränkung des Automaten $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ auf eine Teilmenge dieser Zustände entsteht, die gleichzeitig die Steuerbarkeit nicht verletzen, ist die Co-Erreichbarkeit gewährleistet. Das Gleichungssystem für $\mathsf{EF}x_m$ ist Gleichungssystem 6.6 mit $\varphi = x_m$ und $u = u_1$, also

$$\left\{ \; u_1 \stackrel{\mu}{=} x_m \vee \Diamond u_1 .$$

Wird diese Gleichung in die Vorlage 6.17 eingebaut, entsteht das Gleichungssystem

$$\left\{ \begin{array}{lcl} u_1 & \stackrel{\mu}{=} & (x_m \vee \Diamond u_1) \wedge \overline{x_u} \wedge z \\ u_g & \stackrel{\nu}{=} & \Box u_g \wedge \overline{x_b} x_u \wedge \kappa(u_1) \\ z & \stackrel{\nu}{=} & \kappa\left(u_g\right) \end{array} \right. \tag{6.19}$$

In der Gleichung für u_1 kann die Konjunktion von $\Diamond u_1$ mit $\overline{x_u}$ weggelassen werden, weil es sich um einen kleinsten Fixpunkt handelt. In der ersten Iteration ist dann $\Diamond 0 = 0$, und in den folgenden können nur noch Zustände eingesammelt werden, die in $\overline{x_u}$ liegen. Damit ist die erste Gleichung in 6.19 äquivalent zu

$$u_1 \stackrel{\mu}{=} (\Diamond u_1 \vee x_m \overline{x_u}) \wedge z.$$

Unter Berücksichtigung der Notationsänderungen $u_1 = u_c$ und $z = u_{cg}$ sind die Gleichungssysteme 6.19 und 6.15 also identisch.

6.4 Fairness

Fairnessbedingungen fordern, dass bestimmte Zustände während der gesamten Laufzeit eines Systems erreichbar bleiben. Sie spielen in reaktiven Systemen eine wichtige Rolle, zumal diese Systeme aus theoretischer Sicht unendlich lange laufen und ihre Dienste dem Benutzer uneingeschränkt zur Verfügung stellen sollen.

Die Spezifikation für das Verhalten einer Fertigungszelle, die in der Lage ist, eine Anzahl verschiedener Teile herzustellen, könnte z.B. die Forderung einschließen, dass es jederzeit möglich sein soll, jedes der genannten Teile zu fertigen. Dies ist eine Fairnessbedingung, die sich nicht in der konventionellen Überwachersynthese formulieren lässt. Es genügt nicht, die Zustände, in denen die jeweiligen Teile fertiggestellt werden, zu markieren, denn eine Lösung für das Syntheseproblem ist bereits gültig, wenn von allen Zuständen aus *ein* markierter Zustand zu erreichen ist. Die Synthese lässt sich also nicht daraufhin steuern, dass *alle* markierten Zustände erreichbar sind. Darüber hinaus ist es ein weiteres Problem, festzustellen, ob diese Zustände unendlich oft oder nur endlich viele Male erreichbar bleiben.

Ein solches Problem kann mit der verallgemeinerten Überwachersynthese gelöst werden, indem die gewünschte Fairnesseigenschaft als die vom Benutzer wählbare Bedingung eingesetzt wird. Dazu kann der temporallogische Ausdruck $\mathsf{E}\left[\bigwedge_{i=1}^{n} \mathsf{GF}\varphi_i\right]$ benutzt werden, in dem jedes der φ_i einen Zustand (oder falls gewünscht eine Zustandsmenge) darstellt, die stets erreichbar bleiben sollen. Ihre Übersetzung in den μ-Kalkül geschieht mit Hilfe von Gleichung 6.13, indem für den Parameter β der Wert 1 eingesetzt wird. Wird das so gewonnene Gleichungssystem in die Vorlage 6.17 eingesetzt, entsteht folgendes Gleichungssystem für die Synthese mit den gewünschten Fairnesseigenschaften:

$$\begin{cases} u_n & \overset{\mu}{=} & w \wedge \varphi_n \vee \Diamond u_n \\ \vdots & & \\ u_1 & \overset{\mu}{=} & w \wedge \varphi_1 \vee \Diamond u_1 \\ w & \overset{\nu}{=} & \bigwedge_{i=1}^{n} \Diamond u_i \\ u_g & \overset{\nu}{=} & \Box u_g \wedge \overline{x}_b x_u \wedge \kappa(w) \\ z & \overset{\nu}{=} & \kappa\left(u_g\right). \end{cases}$$

Ein weiteres Beispiel für die Einbeziehung von Fairness in die Überwachersynthese ist die Verwendung von Gleichungssystem 6.14, das die Akzeptanzbedingung von Rabin-Automaten darstellt. Diese wurden in Arbeiten von Thistle und Wonham [80, 81] verwendet mit dem Ziel, eine auf Fairness ausgelegte Variante des Modells von Ramadge und Wonham zu entwerfen. Allerdings werden in diesem speziellen Ansatz ausschließlich unendliche Pfade betrachtet, so dass er keine Verallgemeinerung des Grundmodells darstellt. Zudem ist die Übersetzung einer informalen Spezifikation in einen ω-Automaten in der Regel nicht einfach, wodurch theoretische Fähigkeiten und tatsächliche Anwendungsmöglichkeiten weit auseinanderrücken. Die hier vorgestellte Erweiterung ersetzt das Verfahren von Thistle und Wonham in einer zu all den anderen erlaubten Spezifikationen konsistenten Form.

6.5 Verbesserte Telefonnummernauskunft

In diesem Abschnitt wird das Beispiel aus Abschnitt 5.4 wieder aufgegriffen. Der folgende Text setzt die Kenntnis dieses Beispiels voraus.

In Abschnitt 5.4.5 wurde erläutert, dass trotz Einhaltung der Co-Erreichbarkeit Zustände erreichbar sind, die dem Bearbeiter den Weg zur Anmeldung versperren. Um dieses Problem zu beheben, müssen Zustände aus $\mathcal{A}_{\mathcal{P}} \times \mathcal{A}_{\mathcal{E}}$, denen der Weg zurück zum Initialzustand während der Synthese abschnitten wird, verboten werden. Es genügt allerdings nicht zu fordern, dass es von allen Zuständen stets möglich sein muss, zurück zum Initialzustand zu gelangen. Es gibt nämlich auch Zustände, die den Initialzustand von vorn herein nicht erreichen können und trotzdem erhalten bleiben sollen, weil sie die rasche Abmeldung ermöglichen, die zu dem markierten Zustand (8,4) in Abbildung 5.16 führt. Zeigt sich also während der Synthese, dass ein Zustand keinen Pfad zum Initialzustand hat, muss er dann und nur dann verboten werden, wenn er in $\mathcal{A}_{\mathcal{P}} \times \mathcal{A}_{\mathcal{E}}$ einen solchen Pfad hatte. Für die anderen Zustände genügt es, dass sie keine Verklemmungen bilden.

Diese informale Spezifikation muss nun in den μ-Kalkül umgesetzt werden. Zunächst müssen auf der Kripke-Struktur $\mathcal{K}_{\mathcal{A}_{\mathcal{P}} \times \mathcal{A}_{\mathcal{E}}}$ die Zustände identifiziert werden, die einen Pfad zum Initialzustand (0,0) – symbolisch $\varphi_{\{(0,0)\}}$ – haben. Diese Menge wird zu Beginn des Syntheseprozesses einmal berechnet und bleibt danach konstant. Die gesuchten Zustände erfüllen den Ausdruck $\mathsf{EF}\varphi_{\{(0,0)\}}$. Diese Menge ist laut Gleichung 6.6 die Lösung von

$$u_0 \overset{\mu}{=} \varphi_{\{(0,0)\}} \vee \Diamond u_0. \tag{6.20}$$

Die Spezifikation verlangt, dass Zustände aus dieser Menge einen Pfad zum Initialzustand beibehalten müssen, um den Syntheseprozess zu überstehen; alle anderen Zustände müssen lediglich co-erreichbar bleiben. Zustände, die während der Synthese einen Pfad zum Initialzustand behalten, erfüllen wie oben den Ausdruck $\mathsf{EF}\varphi_{\{(0,0)\}}$.

Es wird also eine weitere Gleichung wie 6.20 benötigt, die aber in Abhängigkeit von anderen Zustandsmengen neu berechnet wird. Dazu dient die Gleichung

$$u_3 \stackrel{\mu}{=} \varphi_{\{(0,0)\}} \vee \Diamond u_3. \tag{6.21}$$

Zustände, die co-erreichbar bleiben, werden wie in Abschnitt 6.3 durch die Gleichung

$$u_2 \stackrel{\mu}{=} x_m \vee \Diamond u_2 \tag{6.22}$$

erfasst, so dass die obige Bedingung sich als

$$u_1 \stackrel{\mu}{=} u_0 \wedge u_3 \vee \neg u_0 \wedge u_2 \tag{6.23}$$

darstellen lässt.

Das Gleichungssystem aus den Gleichungen 6.20 bis 6.23 wird nun zusammen mit den Automaten $\mathcal{A}_{\mathcal{P}}$ und $\mathcal{A}_{\mathcal{P}} \times \mathcal{A}_{\mathcal{E}}$ aus Abschnitt 5.4 als Eingabe für das verallgemeinerte Überwachersyntheseproblem benutzt. Damit entsteht laut Theorem 6.4 nach ähnlichen Vereinfachungen wie in Abschnitt 6.3 das Gleichungssystem

$$\left\{\begin{array}{lcl} u_0 & \stackrel{\mu}{=} & \varphi_{\{(0,0)\}} \vee \Diamond u_0 \\ u_3 & \stackrel{\mu}{=} & (\Diamond u_3 \vee \varphi_{\{(0,0)\}} \wedge \overline{x_u}) \wedge z \\ u_2 & \stackrel{\mu}{=} & (\Diamond u_2 \vee x_m \overline{x_u}) \wedge z \\ u_1 & \stackrel{\mu}{=} & (u_0 \wedge u_3 \vee \neg u_0 \wedge u_2) \wedge \overline{x_u} \wedge z \\ u_g & \stackrel{\nu}{=} & \Box u_g \wedge \overline{x_b} x_u \wedge \kappa(u_1) \\ z & \stackrel{\nu}{=} & \kappa\left(u_g\right). \end{array}\right. \tag{6.24}$$

Die Lösung dieses Gleichungssystems mit Algorithmus 3.57 ergibt nach Theorem 6.4 einen Überwacher mit 218 Zuständen und 894 Transitionen. Dieser ist erwartungsgemäß etwas kleiner als die Lösung in [77] (239 Zustände, 969 Transitionen), weil nun auch die Zustände vermieden werden, die den Bearbeiter wie in Abschnitt 5.4.5 beschrieben daran hindern, sich nach einem Abmeldevorgang wieder anzumelden.

Der Unterschied zur konventionellen Lösung aus Abschnitt 5.4.4 lässt sich mit Hilfe von Abbildung 6.1 erklären, die einen Teil des neuen Überwachers zeigt. Hier sind die Zustände der Spezifikation, die durch die verallgemeinerte Synthese entfernt werden, nur noch mit gestrichelten Linien dargestellt (vgl. Abbildungen 5.14 und 5.16).

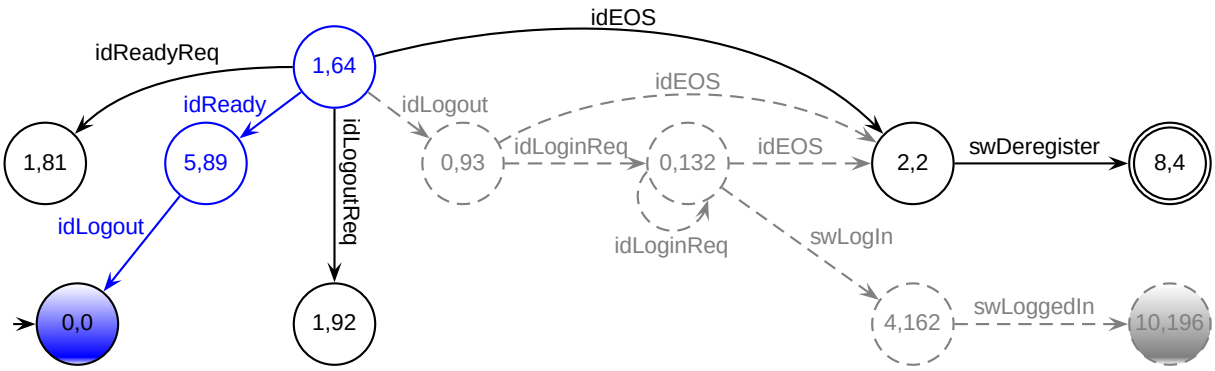


Abbildung 6.1: Teil des fairen Überwachers für die Auskunftzentrale

Die Korrektheit des Ergebnisses kann anhand der Ereignisse in Abbildung 5.15 verdeutlicht werden, die in Abschnitt 5.4.5 in den unerwünschten Zustand (0,93) führen. Nach den ersten acht Ereignissen befindet sich das überwachte System unter dem neuen Überwacher im Zustand (1,64) in Abbildung 6.1. In diesem Zustand ist nun das Ereignis idLogout nicht mehr erlaubt. Dies bedeutet, dass es mit dem neuen Überwacher nicht möglich ist, wie in Abschnitt 5.4.3 über (0,93) in den verhängnisvollen Zustand (0,132) zu gelangen. Statt

dessen muss die Steuerung im Zustand (1,64) das Ereignis idReady wählen, was die korrekte Antwort auf die vorangegangene automatische Anmeldung swReady seitens der Anlage ist. Der Befehl idLogout wird erst in Zustand (5,89) gegeben. So findet die Steuerung wieder in den Initialzustand (0,0) zurück und der Bearbeiter kann sich wie gewünscht erneut anmelden.

Um zu überprüfen, ob die zusätzlichen Einschränkungen, die der neue Überwacher dem System aufzwingt, harmlos sind, genügt es zu testen, ob der Zustand „angemeldet und aktiv“ für den Bearbeiter erreichbar ist. Die eben durchgespielte Folge von Ereignissen zeigt, dass dies möglich ist. Dies kann ebenso durch eine einfache Anwendung der Modellprüfung bestätigt werden, indem auf der rechten Hälfte der Kripke-Struktur die Menge der erreichbaren Zustände berechnet und dann geprüft wird, ob für die Automaten in den Abbildungen 5.6 bis 5.12 der Zustand (2,5,0,0,0,0,0) erreichbar ist. Auch dieser Test fällt positiv aus.

Dieses Beispiel verdeutlicht die Möglichkeiten, die die gleichzeitige Verwendung von Überwachersynthese und Modellprüfung eröffnet. Einerseits hätte die hier erzielte Lösung nicht allein mit der klassischen Überwachersynthese erzielt werden können, andererseits hätte die Modellprüfung allein keine Unterstützung für die Modellierung geliefert. Der Vorteil des neuen Verfahrens basiert auf der gemeinsam nutzbaren Kripke-Struktur, die es erlaubt, Methoden aus beiden Lagern zu beidseitigem Vorteil zu kombinieren. Mit diesem Beispiel endet die Beschreibung der in dieser Arbeit erzielten Fortschritte.

6.6 Ähnliche Ansätze

In diesem Abschnitt wird der in dieser Arbeit entwickelte Ansatz mit denen von Jiang und Kumar [43], de Alfaro et al. [25] und Arnold et al. [2] verglichen. Auf den ersten Blick versuchen all diese Ansätze, das Überwachersyntheseproblem mit Hilfe des μ-Kalküls oder zumindest von Temporallogiken zu lösen. Eine genauere Untersuchung fördert jedoch wichtige Unterschiede zutage. Um diese zu diskutieren ist es wichtig festzustellen, ob ein gegebener Ansatz das Überwachersyntheseproblem auf das *Erfüllbarkeits-* oder auf das *Modellprüfungsproblem* zurückführt.

Das μ-Kalkül-Erfüllbarkeitsproblem kann mit Hilfe von Abbildung 6.2(a) veranschaulicht werden. Gegeben sei eine μ-Kalkül-Formel φ bzw. das dazu äquivalente Gleichungssystem. Herauszufinden ist, ob es eine Kripke-Struktur $\mathcal{K}$ gibt, die φ erfüllt. Damit ist gemeint, dass φ auf der Menge der Initialzustände der Kripke-Struktur gelten muss. Um diese Frage zu beantworten, wird die Formel in einen Baumautomaten A_φ übersetzt. So wie herkömmliche Automaten Wörter akzeptieren, akzeptieren Baumautomaten Kripke-Strukturen. Die o.g. Übersetzung sorgt dafür, dass der Automat A_φ genau die Kripke-Strukturen akzeptiert, welche die Formel φ erfüllen. Damit wird das Erfüllbarkeitsproblem auf das Leerheitsproblem des Automaten A_φ zurückgeführt.

Diese Lösung des Erfüllbarkeitsproblems umfasst zwei schwierige Schritte. Der erste ist die Übersetzung der Formel in den Automaten. Wilke erläutert in [83], dass die Komplexität dieser Übersetzung, die gleichzeitig eine obere Grenze für die Größe des Automaten ist, exponentiell mit der Länge der μ-Kalkül-Formel wächst. Im Prinzip könnte an dieser Stelle argumentiert werden, dass auch die Übersetzung von CTL*-Formeln exponentiell ist, und dass dies bereits in Abschnitt 6.1 in Kauf genommen wurde, weil die Formeln in der Regel nicht besonders lang sind. Hier ist jedoch zu beachten, dass gerade weil die Übersetzung von CTL* in den μ-Kalkül bereits ein EXPTIME-Vollständiges Problem ist, μ-Kalkül-Formeln in der Regel wesentlich länger als CTL*-Formeln sind. Insgesamt ist also die Übersetzung einer CTL*-Formel in einen Baumautomaten doppelt exponentiell. Die zweite Hürde liegt in der Lösung des Leerheitsproblems für den Automaten. Die Komplexität des besten bekannten Algorithmus für diese Übersetzung, gegeben von Emerson und Jutla [29], wächst exponentiell mit der Länge der μ-Kalkül-Formel, also ebenfalls doppelt exponentiell mit der Länge der CTL*-Formel. Hinzu kommt noch, dass die o.g. Algorithmen nicht gut geeignet für eine Implementierung mit symbolischer Darstellung sind.

Das μ-Kalkül-Modellprüfungsproblem wird in Abbildung 6.2(b) verdeutlicht. Gegeben seien eine μ-Kalkül-Formel φ bzw. das äquivalente Gleichungssystem und eine Kripke-Struktur $\mathcal{K}$. Es gilt, herauszufinden, ob $\mathcal{K}$ die Formel φ erfüllt. Der wesentliche Vorteil gegenüber dem Erfüllbarkeitsproblem besteht darin, dass die Übersetzung der Formel in den Baumautomaten in diesem Fall nicht notwendig ist. Da die Kripke-Struktur bekannt ist, können Modellprüfungsalgorithmen eingesetzt werden um zu verifizieren, ob φ auf den Initialzuständen

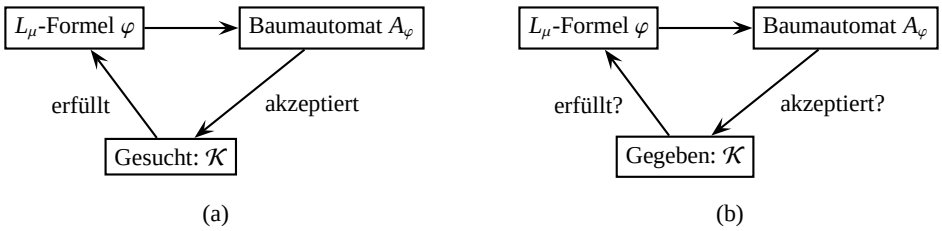


Abbildung 6.2: a) Das μ-Kalkül-Erfüllbarkeitsproblem; b) Das μ-Kalkül-Modellprüfungsproblem

von $\mathcal{K}$ gilt. Theorem 3.62 besagt, dass die Komplexität dieses Problems polynomiell mit der Länge der Formel und der Größe der Kripke-Struktur wächst. Wie in Kapitel 3 erläutert, sind Modellprüfungsalgorithmen auch für die symbolische Implementierung gut geeignet.

Ein weiteres Problem des Erfüllbarkeitsansatzes ist, dass der Baumautomat alle Kripke-Strukturen, die die gegebene Formel erfüllen, gleichermaßen behandelt. Die Auswahl einer unter vielen Möglichkeiten bleibt dem Anwender überlassen, und auch diese Aufgabe kann einen nicht vernachlässigbaren Aufwand verursachen. Im Gegensatz ist die Lösung des Modellprüfungsproblems immer genau eine Zustandsmenge, also eine Kripke-Struktur. In Anbetracht dieser Tatsachen scheint es angemessen, zu erwarten, dass Lösungen, die die Überwachersynthese auf das Modellprüfungsproblem zurückführen, im Allgemeinen leichter zu handhaben sind als jene, die auf dem Erfüllbarkeitsproblem basieren.

Die in dieser Arbeit vorgestellten Lösungen für das Überwachersyntheseproblem unterscheiden sich von der Modellprüfung nur in der Art und Weise, in der das Ergebnis des Modellprüfungsalgorithmus verwendet wird. Dieses Ergebnis ist die Zustandsmenge, auf der die gegebene Formel gilt. Während in der konventionellen Modellprüfung geprüft wird, ob die Initialzustände der Kripke-Struktur Teil dieser Menge sind, wird hier der Überwacher daraus abgeleitet. Dies ändert die Komplexität der Lösung jedoch nicht. Folglich fallen alle ab Kapitel 4 vorgestellten Lösungsvarianten in die Klasse der Modellprüfungsprobleme. Im Gegensatz dazu beruhen die Ansätze, die im Folgenden diskutiert werden, auf dem Erfüllbarkeitsproblem oder benutzen Übersetzungsprozeduren mit ähnlicher Komplexität.

Jiang und Kumar [43] gehen von den gleichen endlichen Automaten aus wie der hier vorgestellte Ansatz. Die Forderung nach Co-Erreichbarkeit wird ähnlich wie in Abschnitt 6.2 gelockert und durch eine temporallogische Formel in CTL ersetzt. Diese Formel wird dann in einen Rabin-Baumautomaten übersetzt, wodurch die Synthese auf das Erfüllbarkeitsproblem zurückgeführt wird. Es ist zwar möglich, die Größe des Baumautomaten nicht exponentiell, sondern linear in der Größe der Systembeschreibung zu halten, jedoch wächst sie doppelt exponentiell mit der Länge der temporallogischen Formel an. Außerdem kann, wie in den Beispielen in Kapitel 5 verdeutlicht, bereits der Automat für die Systembeschreibung unter der Zustandsexplosion leiden, was den Vorteil des linearen Wachstums wieder relativiert. Da die Baumautomaten keine effiziente symbolische Darstellung der Transitionsrelation zulassen, bleibt das Verfahren auf relativ kleine Anwendungen beschränkt. Durch die Einführung der CTL-Formel ist der Ansatz allgemeiner als der ursprüngliche von Ramadge und Wonham, wird aber von dem in Kapitel 6 vorgestellten subsumiert, zumal die Logik CTL eine strikte Untermenge des μ-Kalküls ist.

In dem Ansatz von de Alfaro et al.[25] wird das zu steuernde System nicht als Automat, sondern als Spielgraph dargestellt. Die Spezifikation ist eine LTL-Formel und die Lösung des Syntheseproblems besteht darin, die Menge der Zustände des Spielgraphen zu finden, welche die Formel erfüllen. Insofern wird das Syntheseproblem auf ein Modellprüfungsproblem zurückgeführt. Allerdings muss die LTL-Formel in den μ-Kalkül übersetzt werden, bevor die Modellprüfung beginnen kann. Während der hier vorgestellte Ansatz für die Übersetzung der temporallogischen Formeln wie in Abschnitt 6.1 beschrieben auf bewährte Methoden zurückgreifen kann, ist der Ansatz von de Alfaro et al. auf eine spezielle Übersetzung angewiesen, die auf Safras Prozedur zur Erstellung deterministischer ω-Automaten basiert [74]. Die Komplexität dieses Algorithmus ist doppelt exponentiell in der Länge der LTL-Formel und kann die Vorteile einer symbolischen Darstellung nicht nutzen.

Ein weiterer Ansatz, der die Überwachersynthese auf das Erfüllbarkeitsproblem zurückführt, ist der von Arnold et al. [2]. Ähnlich wie in der Verallgemeinerung in Abschnitt 6.2 wird die Forderung nach Co-Erreichbarkeit durch beliebige μ-Kalkül-Formeln ausgetauscht, die in alternierende Baumautomaten übersetzt werden. Damit wählt der Ansatz die Lösung über das Erfüllbarkeitsproblem. Dieses wird weiter auf die Suche nach Gewinnstrategien für Paritätsspiele zurückgeführt [31, 45]. Der Ansatz enthält auch eine Erweiterung für die Behandlung beschränkt sichtbarer Ereignisse, eine der in Abschnitt 2.7 erwähnten Varianten des Ramadge-Wonham-Modells, die im Rahmen dieser Arbeit aber nicht näher behandelt werden.

Auch wenn von dieser Erweiterung abgesehen wird, kann die Interpretation der μ-Kalkül-Spezifikation in beiden Verfahren unterschiedlich ausfallen, wie das folgende Beispiel zeigt. Angenommen, das System wird durch den Automaten in Abbildung 6.3 dargestellt, in dem das Ereignis α steuerbar ist. Der Automat für die Spezifikation ist identisch mit dem der Systembeschreibung und die temporallogische Bedingung lautet „es soll keine unendlichen Pfade geben“, was sich im μ-Kalkül mit Hilfe der Gleichung 6.1 ausdrücken lässt (s. Seite 103).

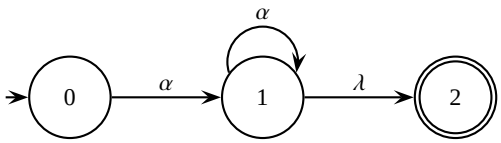


Abbildung 6.3: Eine Systembeschreibung

In der Interpretation von Arnold et al. sollte diese Bedingung zu einem Überwacher führen, der dem Automaten in Abbildung 6.3 ohne die Schleife um Zustand 1 gleicht. Um dies zu erreichen, muss der Synthesealgorithmus in der Lage sein, einzelne Transitionen aus der Transitionsrelation zu entfernen, ohne dass die betroffenen Zustände ebenfalls entfernt werden. Dies ist allerdings nicht das, was in der hier vorgestellten Verallgemeinerung passiert. Die Lösung in Abschnitt 6.2 kann nur ganze Zustände (zusammen mit allen Transitionen, die an diese Zustände gekoppelt sind) entfernen. Insofern ist Gleichung 6.1 in diesem Fall als „es soll keine Zustände mit unendlichen Pfaden geben“ zu lesen. Die Lösung des verallgemeinerten Überwachersyntheseproblems würde dementsprechend die Zustände 1 und 2 (letzteren wegen Unerreichbarkeit) entfernen, so dass nur der Initialzustand zurückbliebe. Allerdings kann der von Arnold et al. gewünschte Überwacher auch mit dem hier vorgestellten Ansatz berechnet werden, indem ein geeigneter Automat als Spezifikation gewählt wird. Während Arnold et al. den gleichen Automaten für die Systembeschreibung und für die Spezifikation verwenden, würde der Anwender bei dem Ansatz aus Abschnitt 6.2 die Schleife um Zustand 1 bereits aus der Spezifikation entfernen. Da das Ereignis α steuerbar ist, entsteht nun der gleiche Überwacher wie bei Arnold et al. Das Beispiel zeigt, dass mit einer angemessenen Spezifikation beide Ansätze zu denselben Ergebnissen führen können. Ob beide Ansätze exakt die gleiche Problemklasse behandeln, ist zur Zeit noch unklar. Auf jeden Fall aber hat der hier vorgestellte Ansatz die Vorteile, die durch die Verwendung der Modellprüfung an Stelle der Erfüllbarkeit entstehen.

Kapitel 7

Zusammenfassung und Ausblick

Die vorliegende Arbeit geht auf die Problematik des Entwurfs diskreter Systeme ein. Abstrakt gesehen, besteht diese Aufgabe lediglich darin, in jedem Zustand eines Systems für jedes mögliche Ereignis eine angemessene Reaktion vorzusehen. Diese prinzipiell einfache Aufgabe wird in der Praxis oft durch die sehr große Anzahl der Systemzustände erschwert. Andererseits können gerade bei komplexen Systemen fehlerhafte Lösungen hohe Investitionen oder gar Menschenleben gefährden. Deshalb sind formale Ansätze für die Handhabung dieser Systeme besonders wichtig.

Von den vielen Ansätzen, die in den letzten Jahrzehnten mehr oder weniger erfolgreich versucht haben, diese Systeme zu meistern, heben sich zwei Verfahren ab, die in dieser Arbeit näher behandelt und letztendlich zu einem neuen, vorteilhafteren Verfahren zusammengefügt werden. Es handelt sich dabei um die Überwachersynthese und die μ-Kalkül-basierte Modellprüfung, die im ersten Teil der Arbeit beschrieben werden. Am Ende der Kapitel 2 bzw. 3 werden bereits Verbesserungsmöglichkeiten auf beiden Seiten aufgelistet, die im zweiten Teil der Arbeit schrittweise realisiert werden.

Entscheidend für den Erfolg dieses Unterfangens sind die Ergebnisse in Kapitel 4, in dem die herkömmlichen Synthesealgorithmen in μ-Kalkül-Gleichungssysteme übersetzt werden. Dies ermöglicht nicht nur die Zusammenführung von Überwachersynthese und Modellprüfung, sondern auch eine Weiterentwicklung der Synthesealgorithmen in den zwei folgenden Kapiteln. Die aus den Algorithmen gewonnenen Gleichungssysteme dienen in Kapitel 5 als Ausgangspunkt für eine Verbesserung der Effizienz der Überwachersynthese. Diese geht mit einer Verringerung der Komplexität von quadratisch auf linear einher und kann, wie das Beispiel in Abschnitt 5.5 zeigt, große Verbesserungen der Laufzeit erzielen. In diesem Zusammenhang wird auch die für die Praxis wichtige Problemklasse, bei der diese Verringerung eintritt, genau beschrieben. Auch solche Probleme, bei denen sich die Komplexität nicht verringern lässt, werden von dem neuen Ansatz in der Regel schneller gelöst, zumal die Berechnungen automatisch auf dem möglichst effizienten Weg geschehen. Das Kapitel 5 enthält ein weiteres Beispiel, das aus einer kommerziellen Anwendung der Überwachersynthese zur Steuerung einer Telefonnummernauskunftzentrale stammt und hier genauer analysiert wird. Dabei wird auch ein subtiler Fehler gefunden, der sich mit der konventionellen Überwachersynthese nicht beheben lässt.

In Kapitel 6 werden die Gleichungssysteme aus Kapitel 4 als Ausgangspunkt für eine Verallgemeinerung der Überwachersynthese verwendet. Um dies zu erreichen, werden an Stelle der festen Forderung nach Co-Erreichbarkeit, die Teil des ursprünglichen Verfahrens ist, vom Anwender frei wählbare Eigenschaften zugelassen. Die Folge ist eine deutliche Steigerung der Spezifikationsmöglichkeiten, insbesondere die Einbeziehung von Fairnesseigenschaften in den Syntheseprozess. Unter anderem wird es dadurch möglich, den Fehler in dem Beispiel mit der Telefonnummernauskunft zu beheben.

Abschließend werden die erzielten Ergebnisse mit ähnlichen Verfahren verglichen. Entscheidend ist, ob ein gegebenes Verfahren die Überwachersynthese auf das Erfüllbarkeits- oder auf das Modellprüfungsproblem im μ-Kalkül zurückführt. Die erste dieser Möglichkeiten ist sicherlich die schwierigste, nicht zuletzt was die Implementierung von Synthesewerkzeugen angeht. Trotzdem beruhen alle analysierten Verfahren entweder auf dem Erfüllbarkeitsproblem oder benötigen Übersetzungsprozeduren ähnlicher Komplexität. Der hier vorge-

stellte Ansatz ist nach dem aktuellen Wissensstand des Autors momentan das einzige, das alle Vorteile der Modellprüfung für die Überwachersynthese nutzen kann.

Die neuen Ergebnisse werfen auch neue Fragen auf, denen in einer Weiterführung der Forschungsarbeiten nachgegangen werden kann. Die vorliegende Arbeit musste sich bereits in Kapitel 2 auf das Grundmodell von Ramadge und Wonham beschränken. Es ist anzunehmen, dass ein Großteil, wenn nicht die Gesamtmenge der in Abschnitt 2.7 aufgelisteten Varianten in ähnlicher Weise von den Vorteilen der Übersetzung in den μ-Kalkül profitieren kann.

Auch im Bereich der Algorithmen für die Lösung von Gleichungssystemen stehen noch Fragen offen, besonders nach Optimierungen, die erst bei höheren Alternierungstiefen greifen. Bis jetzt sind solche Gleichungssysteme in der Praxis nicht oft aufgetreten, da die gängigen Temporallogiken in Gleichungssysteme mit höchstens Alternierungstiefe 2 übersetzt werden. Bei der Überwachersynthese wird die Alternierungstiefe im Allgemeinen durch die Gleichung für die Steuerbarkeit um eins erhöht. Weiterhin erlauben die vom Anwender frei erstellbaren Gleichungssysteme beliebige Alternierungstiefen. Für diese Fälle kann es sich lohnen, nach Algorithmen zu suchen, die effizienter als die hier verwendeten sind.

Die Übersetzung der Überwachersynthese in den μ-Kalkül in Theorem 4.8 zeigt, dass die Abschätzung der Komplexität eines Gleichungssystems anhand seiner Alternierungstiefe zu hoch geraten kann. Theorem 3.62 liefert für die Lösung von Gleichungssystem 4.5 die obere Grenze $O\left(|\mathcal{S}|^2\,|\mathcal{R}|\right)$, woraus sich $O\left(|Q|^3\,|\Sigma|\right)$ ergibt. Wie in dem Beweis von Theorem 4.8 gezeigt, kann die obere Grenze auf Grund der Abhängigkeiten jedoch auf $O\left(|Q|^2\,|\Sigma|\right)$ verringert werden. Dieser Einfluss der Abhängigkeiten auf die Komplexitätsabschätzung ist in der Literatur über den μ-Kalkül bisher – wenn überhaupt – nur sehr wenig berücksichtigt worden. Es sollte möglich sein, genauere Abschätzungen für ein Gleichungssystem anhand seines Abhängigkeitsgraphen zu machen.

Von den in Abschnitt 2.8 erwähnten Verbesserungsmöglichkeiten ist sicherlich die Forderung nach einer höheren Bereitschaft zur Anwendung der Überwachersynthese im industriellen Bereich am schwersten zu erfüllen. Dafür gibt es mehrere Gründe. Zum Teil konnten bisher nur relativ kleine Systeme behandelt werden, ein Problem das aber mit Hilfe der hier verwendeten symbolischen Darstellung deutlich an Gewicht verliert. Andererseits ist der Lernaufwand bei formalen Verfahren relativ groß. Dies hat besonders in der modernen Industrie eine abschreckende Wirkung auf potentielle Anwender. Aus dieser Sicht ist die Entwicklung von Werkzeugen sinnvoll, die den Zugang zur Modellierung und Synthese mit endlichen Automaten und temporallogischen Formeln erleichtern.

Die Umstellung eines gewohnten Entwicklungsablaufs auf formale Methoden erfordert ein tiefgreifendes Umdenken, das nicht schlagartig zu erreichen ist. Eine schrittweise Einführung der Überwachersynthese ist jedoch möglich, indem die formal erzeugten Spezifikationen zunächst nur als Hilfsmittel herangezogen und wie gehabt umgesetzt werden. Dies ist auch deshalb angebracht, weil sich nicht jedes Problem mit formalen Methoden lösen lässt. Es sollte sich für jeden Anwender Schritt für Schritt herausstellen, wie die Überwachersynthese im Einklang mit anderen Methoden einzusetzen ist.

Teil III

Anhänge

Anhang A

Das Tarski-Knaster-Theorem

Dieses Kapitel enthält einen Beweis für das das Tarski-Knaster-Theorem [78]. Der hier erbrachte Beweis basiert auf Material aus [37, 46, 75, 81]. Es werden Kenntnisse über Ordnungsrelationen und deren Darstellung als Hasse-Diagramme [siehe z.B. 46] vorausgesetzt. Eine Erläuterung der Begriffe in deutscher Sprache findet der Leser z.B. in [36].

A.1 Extrema in Ordnungsrelationen

Definition A.1 (Ordnungsrelationen) *Gegeben seien eine Menge $\mathcal{D}$ und eine Relation $\sqsubseteq$ in $\mathcal{D}$. $(\mathcal{D}, \sqsubseteq)$ ist eine* reflexive Halbordnungsrelation *g.d.w. folgende Gesetze gelten:*

- *Reflexivität:* $\forall x \in \mathcal{D}.x \sqsubseteq x$;
- *Antisymmetrie:* $\forall x, y \in \mathcal{D}.x \sqsubseteq y \wedge y \sqsubseteq x \rightarrow x = y$;
- *Transitivität:* $\forall x, y, z \in \mathcal{D}.x \sqsubseteq y \wedge y \sqsubseteq z \rightarrow x \sqsubseteq z$.

$(\mathcal{D}, \sqsubseteq)$ *ist eine* reflexive Vollordnungsrelation *g.d.w. zusätzlich alle Elemente von $\mathcal{D}$ miteinander vergleichbar sind, also wenn* $\forall x, y \in \mathcal{D}.x \sqsubseteq y \vee y \sqsubseteq x$.

Im Folgenden kommen nur noch reflexive Halbordnungsrelationen vor, die einfach *Ordnungsrelationen* genannt werden.

Definition A.2 (Maximale und minimale Elemente) *Gegeben seien eine Ordnungsrelation $(\mathcal{D}, \sqsubseteq)$ und eine Menge $M \subseteq \mathcal{D}$. Ein Element $a \in M$ ist ein* maximales Element von M, *falls* $\forall x \in M.a \sqsubseteq x \rightarrow a = x$; *a ist ein* minimales Element von M, *falls* $\forall x \in M.x \sqsubseteq a \rightarrow x = a$.

Beispiel A.3 Abbildung A.1 zeigt das Hasse-Diagramm einer Ordnungsrelation in der Menge $\mathcal{D} = \{a_0, a_1, \ldots, a_{12}\}$. Die Menge $M \subseteq \mathcal{D}$ hat maximale Elemente a_5, a_6 und minimale Elemente a_{11}, a_{12}. Letztere sind gleichzeitig minimale Elemente von $\mathcal{D}$. Die Menge $\mathcal{D}$ hat ein maximales Element, a_0. □

Definition A.4 (Maxima und Minima) *Gegeben seien eine Ordnungsrelation $(\mathcal{D}, \sqsubseteq)$ und eine Menge $M \subseteq \mathcal{D}$. Ein Element $a \in M$ ist das* Maximum von M, *falls* $\forall x \in M.x \sqsubseteq a$; *a ist das* Minimum von M, *falls* $\forall x \in M.a \sqsubseteq x$.

Lemma A.5 (Eindeutigkeit der Maxima und Minima) *Eine Menge $M \subseteq \mathcal{D}$ hat höchstens ein Maximum und ein Minimum.*

Beweis: Angenommen, M hätte zwei Maxima, a_1 und a_2. Per Definition müssten dann gelten $a_1 \sqsubseteq a_2$ und $a_2 \sqsubseteq a_1$ und, wegen der Antisymmetrie, $a_1 = a_2$. Die Argumentation für das Minimum ist ähnlich. □

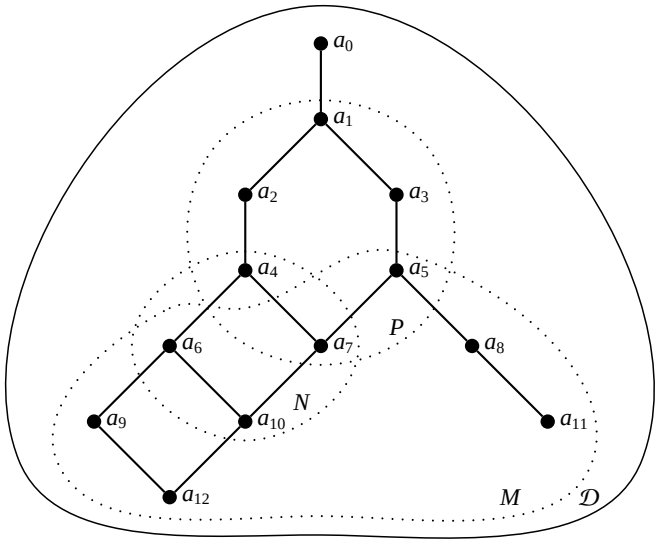


Abbildung A.1: Eine Menge $\mathcal{D}$ mit Untermengen M, N und P

Beispiel A.6 Die Menge M in Abbildung A.1 hat kein Maximum und kein Minimum (a_5 ist kein Maximum von M, da a_5 und a_6 nicht vergleichbar sind). Die Menge N das Maximum a_4 und das Minimum a_{10}; die Menge P hat das Maximum a_1 und das Minimum, a_7; die Menge $\mathcal{D}$ hat das Maximum a_0 und kein Minimum, da a_{11} und a_{12} nicht vergleichbar sind. □

Die Definitionen A.2 und A.4 beziehen sich auf Elemente einer Untermenge von $\mathcal{D}$. Die folgenden Definitionen beschreiben Verhältnisse zwischen Elementen einer solchen Untermenge und Elementen, die irgendwo in $\mathcal{D}$ liegen können.

Definition A.7 (Obere und untere Schranken) *Gegeben seien eine Ordnungsrelation* $(\mathcal{D}, \sqsubseteq)$ *und eine Menge* $M \subseteq \mathcal{D}$*. Ein Element* $a \in \mathcal{D}$ *ist eine* obere Schranke von M*, falls* $\forall x \in M.(x \sqsubseteq a)$*; a ist eine* untere Schranke von M*, falls* $\forall x \in M.(a \sqsubseteq x)$.

Beispiel A.8 In Abbildung A.1 hat die Menge P obere Schranken a_1 und a_0 und untere Schranken a_7, a_{10} und a_{12}; die Menge N hat obere Schranken a_4, a_2, a_1 und a_0 und untere Schranken a_{10} und a_{12}; die Menge M hat obere Schranken a_1 und a_0 und keine untere Schranke; die Menge $\mathcal{D}$ hat die obere Schranke a_0 und ebenfalls keine untere Schranke. □

Definition A.9 (Supremum und Infimum) *Das Element* $a \in \mathcal{D}$ *ist die* kleinste obere Schranke *oder* Supremum von M*, geschrieben* $\sup(M)$*, falls a das Minimum aller oberen Schranken von M ist, d.h.,*

$$(\forall x \in M.x \sqsubseteq a) \wedge (\forall y \in \mathcal{D}.(\forall x \in M.x \sqsubseteq y).a \sqsubseteq y)\,;$$

a ist die größte untere Schranke *oder* Infimum von M*, geschrieben* $\inf(M)$*, falls a das Maximum aller unteren Schranken von M ist, d.h.,*

$$(\forall x \in M.a \sqsubseteq x) \wedge (\forall y \in \mathcal{D}.(\forall x \in M.y \sqsubseteq x).y \sqsubseteq a)\,.$$

Lemma A.10 (Eindeutigkeit der Suprema und Infima) *Eine Menge* $M \subseteq \mathcal{D}$ *hat höchstens ein Supremum und ein Infimum.*

Beweis: Da Supremum bzw. Infimum als Maximum bzw. Minimum einer Menge definiert sind, müssen sie nach Lemma A.5 eindeutig sein. □

Beispiel A.11 In Abbildung A.1 gilt $\sup(P) = a_1$, $\inf(P) = a_7$, $\sup(N) = a_4$, $\inf(N) = a_{10}$, $\sup(M) = a_1$ und $\sup(\mathcal{D}) = a_0$. Die Mengen M und $\mathcal{D}$ haben kein Infimum. □

Lemma A.12 (Identität der Extrema) *Folgende Aussagen sind äquivalent:*

(i) *Eine Menge $M \subseteq \mathcal{D}$ besitzt ein Maximum x_{max} bzw. ein Minimum x_{min}.*
(ii) $\sup(M) = x_{max}$ *bzw.* $\inf(M) = x_{min}$.
(iii) $\sup(M) \in M$ *bzw.* $\inf(M) \in M$.

Beweis: Die Äquivalenz der Aussagen wird für das Maximum bzw. das Supremum gezeigt. Für das Minimum bzw. das Infimum ist die Vorgehensweise analog.

($i \rightarrow ii$): x_{max} erfüllt die Bedingungen von Definition A.9.

($ii \rightarrow iii$): Falls $\sup(M) = x_{max}$, dann gilt $\sup(M) \in M$, weil x_{max} stets ein Element von M ist.

($iii \rightarrow i$): Falls $\sup(M) \in M$, dann erfüllt $\sup(M)$ die Bedingung von Definition A.4. □

Lemma A.13 (Eigenschaften von Ordnungsrelationen) *Gegeben seien eine Ordnungsrelation $(\mathcal{D}, \sqsubseteq)$ und folgende Untermengen von $\mathcal{D}$: A, B, und, für eine gegebene Indexmenge I, A_i mit $i \in I$. Haben diese Untermengen Suprema und Infima, dann gelten folgende Eigenschaften:*

(i) $\sup(A) = \sup(B)$ *g.d.w.* $\sup(A)$ *eine obere Schranke von B und* $\sup(B)$ *eine obere Schranke von A ist;*

(ii) $\inf(A) = \inf(B)$ *g.d.w.* $\inf(A)$ *eine untere Schranke von B und* $\inf(B)$ *eine untere Schranke von A ist;*

(iii) $\sup(\bigcup_{i \in I} A_i) = \sup(\bigcup_{i \in I} \sup(A_i))$;

(iv) $\inf(\bigcap_{i \in I} A_i) = \inf(\bigcap_{i \in I} \inf(A_i))$;

(v) $A \subseteq B \rightarrow \sup(A) \sqsubseteq \sup(B)$;

(vi) $A \subseteq B \rightarrow \inf(B) \sqsubseteq \inf(A)$.

Beweis: Es werden hier die Beweise für die Suprema gegeben, die für die Infima sind ähnlich.

(i) ($\rightarrow$): trivial.
($\leftarrow$): da $\sup(A)$ eine obere Schranke für B ist, gilt $\sup(B) \sqsubseteq \sup(A)$; umgekehrt gilt $\sup(A) \sqsubseteq \sup(B)$. Wegen der Antisymmetrie von $\sqsubseteq$ folgt $\sup(A) = \sup(B)$.

(iii) Der Beweis besteht aus zwei Teilen:

(a) Das Element $\sup(\bigcup_{i \in I} A_i)$ ist eine obere Schranke für jedes Element der A_i, d.h. $\forall i \in I . \forall x \in A_i . x \sqsubseteq \sup(\bigcup_{i \in I} A_i)$. Da jedes $\sup(A_i)$ jeweils die *kleinste* obere Schranke von A_i ist, gilt $\forall i \in I . \sup(A_i) \sqsubseteq \sup(\bigcup_{i \in I} A_i)$, also $\forall x \in \bigcup_{i \in I} \sup(A_i) . x \sqsubseteq \sup(\bigcup_{i \in I} A_i)$. Damit ist $\sup(\bigcup_{i \in I} A_i)$ eine obere Schranke für $\bigcup_{i \in I} \sup(A_i)$.

(b) Laut Definition A.9 gilt $\forall i \in I . \forall x \in A_i . x \sqsubseteq \sup(A_i)$. Fasst man nun alle $\sup(A_i)$ zu einer Menge zusammen, dann enthält diese Mindestens eine obere Schranke für jedes Element der A_i, d.h. $\forall x \in \bigcup_{i \in I} A_i . \exists y \in \bigcup_{i \in I} \sup(A_i) . x \sqsubseteq y$. Wegen der Transitivität von $\sqsubseteq$ gilt dann $\forall x \in \bigcup_{i \in I} A_i . x \sqsubseteq \sup(\bigcup_{i \in I} \sup(A_i))$. Damit ist $\sup(\bigcup_{i \in I} \sup(A_i))$ eine obere Schranke von $\bigcup_{i \in I} A_i$. Der Beweis folgt nun unmittelbar aus Punkt (i).

(v) Laut Definition A.9 gilt $\forall x \in B . x \sqsubseteq \sup(B)$. Da $A \subseteq B$ gilt ebenfalls $\forall x \in A . x \sqsubseteq \sup(B)$. Also ist $\sup(B)$ eine obere Schranke von A. Da $\sup(A)$ aber die *kleinste* obere Schranke von A ist, folgt $\sup(A) \sqsubseteq \sup(B)$. □

A.2 Verbände

Definition A.14 (Verband) *Eine Ordnungsrelation $(\mathcal{D}, \sqsubseteq)$ ist ein* Verband *g.d.w. es für beliebige Elemente $x, y \in \mathcal{D}$ stets eine größte untere und eine kleinste obere Schranke gibt, d.h.,*

$$\forall x, y \in \mathcal{D}.\exists a, b \in \mathcal{D}.a = \sup(\{x, y\}) \wedge b = \inf(\{x, y\}).$$

Ein Verband ist vollständig *wenn jede Menge $M \subseteq \mathcal{D}$ ein Supremum und ein Infimum in $\mathcal{D}$ hat, d.h.,*

$$\forall M \subseteq \mathcal{D}.\exists a, b \in \mathcal{D}.a = \sup(M) \wedge b = \inf(M).$$

Existieren die Schranken der Menge $\mathcal{D}$, werden für diese die Symbole $\top := \sup(\mathcal{D})$ und $\bot := \inf(\mathcal{D})$ verwendet.

Zur Vereinfachung der Notation schreibt man für $x, y \in \mathcal{D}$:

$$\begin{aligned} x \sqcup y &:= \sup(\{x, y\}) \\ x \sqcap y &:= \inf(\{x, y\}). \end{aligned}$$

Lemma A.15 (Gesetze der Verbände) *Gegeben sei ein Verband $(\mathcal{D}, \sqsubseteq)$. Dann gelten für $x, y, z \in \mathcal{D}$ folgende Gesetze:*

- *Idempotenz:*
  $x \sqcup x = x$;
  $x \sqcap x = x$;
- *Kommutativität:*
  $x \sqcup y = y \sqcup x$;
  $x \sqcap y = y \sqcap x$;
- *Assoziativität:*
  $x \sqcup (y \sqcup z) = x \sqcup (y \sqcup z)$;
  $x \sqcap (y \sqcap z) = x \sqcap (y \sqcap z)$;
- *Absorption:*
  $x \sqcup (x \sqcap y) = x$;
  $x \sqcap (x \sqcup y) = x$.

Beweis:

- Idempotenz: Die Gesetze folgen aus Definition A.9.
- Kommutativität: Die Gesetze folgen ebenfalls aus Definition A.9, die die Elemente x und y symmetrisch behandelt.
- Assoziativität: Laut Definition A.9 gelten $x \sqsubseteq x \sqcup (y \sqcup z)$ und $(y \sqcup z) \sqsubseteq x \sqcup (y \sqcup z)$. Ebenfalls gelten $y \sqsubseteq y \sqcup z$ und $z \sqsubseteq y \sqcup z$. Wegen der Transitivität schließt man daraus $y \sqsubseteq x \sqcup (y \sqcup z)$ und $z \sqsubseteq x \sqcup (y \sqcup z)$. Deshalb ist $x \sqcup (y \sqcup z)$ eine obere Schranke von x und y, also gilt $x \sqcup y \sqsubseteq x \sqcup (y \sqcup z)$. Nun ist $x \sqcup (y \sqcup z)$ eine obere Schranke von $x \sqcup y$ und von z, also folgt $(x \sqcup y) \sqcup z \sqsubseteq x \sqcup (y \sqcup z)$. Analog dazu lässt sich zeigen, dass $x \sqcup (y \sqcup z) \sqsubseteq (x \sqcup y) \sqcup z$. Wegen der Antisymmetrie von $\sqsubseteq$ folgt nun $x \sqcup (y \sqcup z) = (x \sqcup y) \sqcup z$. Der Beweis für die Infima ist ähnlich.
- Absorption: Da $x \sqsubseteq x$ und $x \sqcap y \sqsubseteq x$, ist x eine obere Schranke von x und von $x \sqcap y$. Also gilt $x \sqcup (x \sqcap y) \sqsubseteq x$. Andererseits gilt laut Definition A.9 $x \sqsubseteq x \sqcup (x \sqcap y)$. Wegen der Antisymmetrie folgt $x \sqcup (x \sqcap y) = x$. Der Beweis für den letzten Punkt ist ähnlich. □

Lemma A.16 (Suprema und Infima für besondere Mengen) *Für Mengen, die nur ein Element enthalten, gilt $\forall a \in \mathcal{D}.\sup(\{a\}) = \inf(\{a\}) = a$.*

Falls die Schranke $\bot$ existiert, gilt $\sup(\emptyset) = \bot$. Falls die Schranke $\top$ existiert, gilt $\inf(\emptyset) = \top$.

Beweis:

- Da $\forall a \in \mathcal{D}.a \sqsubseteq a$, ist a gleichzeitig das Maximum und das Minimum der Menge $\{a\}$. Der Beweis folgt nun unmittelbar aus Lemma A.12.
- Supremum und Infimum der leeren Menge: Jedes $a \in \mathcal{D}$ erfüllt die Bedingungen in Definition A.9 trivialerweise. Da das Supremum (Infimum) das kleinste (größte) Element aus dieser Menge ist, folgen $\sup(\emptyset) = \bot$ und $\inf(\emptyset) = \top$. □

Anmerkung: $\sup(\emptyset) \sqsubseteq \inf(\emptyset)$. □

Theorem A.17 (Eigenschaften von Verbänden) *Sei $(\mathcal{D}, \sqsubseteq)$ ein Verband und seien $a_1, \ldots, a_{n+1}$ Elemente von $\mathcal{D}$. Dann gelten für $n \geq 1$ folgende Eigenschaften:*

$$\begin{aligned} \sup(\{a_1, \ldots, a_{n+1}\}) &= \sup(\{\sup(\{a_1, \ldots, a_n\}), a_{n+1}\}) \\ \inf(\{a_1, \ldots, a_{n+1}\}) &= \inf(\{\inf(\{a_1, \ldots, a_n\}), a_{n+1}\}). \end{aligned}$$

Beweis: Hier wird der Beweis für das Supremum gegeben. Für das Infimum ist die Vorgehensweise analog. Der Beweis benutzt das Induktionsprinzip.

Induktionsbasis ($n = 1$): Die Existenz von $\sup(\{a_1, a_2\})$ folgt aus Definition A.14. Es muss gezeigt werden, dass $\sup(\{a_1, a_2\}) = \sup(\{\sup(\{a_1\}), a_2\})$. Laut Lemma A.16 ist $\sup(\{a_1\}) = a_1$, also folgt $\sup(\{\sup(\{a_1\}), a_2\}) = \sup(\{a_1, a_2\})$.

Induktionsschritt ($n \geq 2$): Angenommen, es existiert das Element $\sup(\{a_1, \ldots, a_n\})$. Sei $a = \sup(\{\sup(\{a_1, \ldots, a_n\}), a_{n+1}\})$. Dann gelten laut Definition A.9 $\sup(\{a_1, \ldots, a_n\}) \sqsubseteq a$ und $a_{n+1} \sqsubseteq a$. Es gilt auch $\forall i \leq n.a_i \sqsubseteq \sup(\{a_1, \ldots, a_n\})$. Wegen der Transitivität von $\sqsubseteq$ folgt $\forall i \leq n.a_i \sqsubseteq a$. Somit ist a eine obere Schranke von $\{a_1, \ldots, a_{n+1}\}$. Nun muss gezeigt werden, dass alle oberen Schranken mit a vergleichbar sind. Zu diesem Zweck sei b eine obere Schranke von $\{a_1, \ldots, a_{n+1}\}$. Insbesondere ist dann b eine obere Schranke von $\{a_1, \ldots, a_n\}$, also gilt $\sup(\{a_1, \ldots, a_n\}) \sqsubseteq b$, also ist b eine obere Schranke von $\sup(\{a_1, \ldots, a_n\})$ und von a_{n+1}. Daraus folgt wiederum $\sup(\{\sup(\{a_1, \ldots, a_n\}), a_{n+1}\}) \sqsubseteq b$, d.h., $a \sqsubseteq b$. Also ist a stets vergleichbar mit b und gleichzeitig die gesuchte kleinste obere Schranke von $\{a_1, \ldots, a_{n+1}\}$. □

Korollar A.18 (Eigenschaften endlicher Verbände) *Gegeben sei ein Verband $(\mathcal{D}, \sqsubseteq)$. Ist die Menge $\mathcal{D}$ endlich, so gelten folgende Eigenschaften:*

- *Für jede Menge $M \subseteq \mathcal{D}$ existieren die Elemente $\sup(M)$ und $\inf(M)$;*
- *$(\mathcal{D}, \sqsubseteq)$ ist vollständig.*

Beweis: Die Elemente $\sup(M)$ und $\inf(M)$ können für jedes $M \subseteq \mathcal{D}$ nach Lemma A.16 bzw. nach Theorem A.17 berechnet werden. Damit ist $(\mathcal{D}, \sqsubseteq)$ laut Definition A.14 vollständig. □

A.3 Funktionen

Definition A.19 (Gerichtete Menge) *Gegeben seien eine Ordnungsrelation $(\mathcal{D}, \sqsubseteq)$ und eine Menge $M \subseteq \mathcal{D}$. M ist* gerichtet *g.d.w. für alle $a_1, a_2 \in M$ gelten $\exists a \in M.a_1 \sqsubseteq a \wedge a_2 \sqsubseteq a$ und $\exists b \in M.b \sqsubseteq a_1 \wedge b \sqsubseteq a_2$.*

Beispiel A.20 Die Menge $\mathcal{D}$ in Abbildung A.1 ist nicht gerichtet: Um die erste Bedingung zu erfüllen kann man zwar $a = a_0$ wählen, aber für die zweite Bedingung gibt es kein Element $b \in \mathcal{D}$ so dass $b \sqsubseteq a_{11}$ und $b \sqsubseteq a_{12}$. Hingegen sind N und P gerichtet. □

Lemma A.21 (Endliche gerichtete Mengen) *Gegeben seien eine Ordnungsrelation $(\mathcal{D}, \sqsubseteq)$ und eine endliche, nicht leere Menge $M \subseteq \mathcal{D}$. M ist gerichtet g.d.w. $\sup(M) \in M$ und $\inf(M) \in M$.*

Beweis: Es genügt zu zeigen, dass M unter den gegebenen Umständen ein Maximum und ein Minimum hat. Das Ergebnis folgt dann aus Lemma A.12.

($\rightarrow$): Angenommen, M wäre endlich und gerichtet und hätte kein Maximum. Dann müsste es zwei Elemente $a_1, a_2 \in M$ geben, so dass weder $a_1 \sqsubseteq a_2$ noch $a_2 \sqsubseteq a_1$, und es dürfte kein $a \in M$ geben, so dass $a_1 \sqsubseteq a \wedge a_2 \sqsubseteq a$. Dies widerspricht der Annahme, M sei gerichtet. Das duale gilt für das Minimum.

($\leftarrow$): Angenommen, M hat ein Maximum a und ein Minimum b. Dann gelten $\forall x \in M.x \sqsubseteq a$ und $\forall x \in M.b \sqsubseteq x$, also erfüllen a und b die Bedingungen in Definition A.19. □

Beispiel A.22 *Hier ist ein Gegenbeispiel das zeigt, dass Lemma A.21 tatsächlich nur auf endlichen Mengen gilt: Sei $\mathcal{D} = \mathbb{Q}$ (die Menge der rationalen Zahlen) und $M = \{x \in \mathbb{Q}.0 \leq x < \sqrt{2}\}$. Dann ist M gerichtet, denn für jedes Zahlenpaar a, b ist entweder $a \leq b$ oder $b \leq a$. M hat aber kein Maximum.*

Definition A.23 (Monotone und stetige Funktionen) *Gegeben seien die Ordnungsrelationen $(\mathcal{D}, \sqsubseteq_{\mathcal{D}})$ und $(\mathcal{E}, \sqsubseteq_{\mathcal{E}})$. Die Funktion $f : \mathcal{D} \to \mathcal{E}$ ist* monoton *g.d.w.*

$$\forall x, y \in \mathcal{D}. x \sqsubseteq_{\mathcal{D}} y \to f(x) \sqsubseteq_{\mathcal{E}} f(y).$$

f ist stetig *wenn für jede nicht leere*[1] *gerichtete Menge $M \subseteq \mathcal{D}$ gelten*

$$f(\sup(M)) = \sup(\{f(x) | x \in M\}) \text{ und } f(\inf(M)) = \inf(\{f(x) | x \in M\}).$$

Lemma A.24 (Stetigkeit impliziert Monotonie) *Gegeben sei eine Ordnungsrelation $(\mathcal{D}, \sqsubseteq)$. Jede stetige Funktion $f : \mathcal{D} \to \mathcal{D}$ ist monoton.*

Beweis: Seien $a, b \in \mathcal{D}$ zwei Elemente so dass $a \sqsubseteq b$. Es muss gezeigt werden, dass die Stetigkeit von f verlangt, dass dann $f(a) \sqsubseteq f(b)$.

$a \sqsubseteq b$ impliziert $\sup(\{a, b\}) = b$. Wendet man hier f auf beiden Seiten an, folgt (1) $f(\sup(\{a, b\})) = f(b)$. Die Stetigkeit von f bedeutet, dass (2) $f(\sup(\{a, b\})) = \sup(\{f(a), f(b)\})$. Vergleicht man (1) und (2) folgt $\sup(\{f(a), f(b)\}) = f(b)$, also ist $f(a) \sqsubseteq f(b)$. □

Definition A.25 (Fixpunkt, Prä-Fixpunkt und Post-Fixpunkt) *Gegeben seien eine Ordnungsrelation $(\mathcal{D}, \sqsubseteq)$ und eine Funktion $f : \mathcal{D} \to \mathcal{D}$. Ein Element $x \in \mathcal{D}$ ist ein* Fixpunkt *von f g.d.w. $f(x) = x$. x ist ein* Prä-Fixpunkt *von f g.d.w. $x \sqsubseteq f(x)$ und ein* Post-Fixpunkt *von f g.d.w. $f(x) \sqsubseteq x$.*

Theorem A.26 (Tarski-Knaster-Theorem) *Seien $(\mathcal{D}, \sqsubseteq)$ ein vollständiger Verband und $f : \mathcal{D} \to \mathcal{D}$ eine monotone Funktion. Dann hat f Fixpunkte, und diese bilden einen Verband $(\mathcal{F}_f, \sqsubseteq)$ mit folgendem Minimum bzw. Maximum:*

$$\begin{array}{rcccl} \check{x} & = & \inf(\{x \in \mathcal{D} | f(x) = x\}) & = & \inf(\{x \in \mathcal{D} | f(x) \sqsubseteq x\}) \\ \hat{x} & = & \sup(\{x \in \mathcal{D} | f(x) = x\}) & = & \sup(\{x \in \mathcal{D} | x \sqsubseteq f(x)\}). \end{array}$$

Ist f außerdem stetig, so ist $(\mathcal{F}_f, \sqsubseteq)$ vollständig.

Seien ferner $p, q \in \mathcal{D}$ Elemente, die folgende Bedingungen erfüllen:

$$\begin{array}{llll} (\mu_1) & p \sqsubseteq f(p) & (\nu_1) & f(q) \sqsubseteq q \\ (\mu_2) & p \sqsubseteq \check{x} & (\nu_2) & \hat{x} \sqsubseteq q. \end{array}$$

Die Folge $f^0(p), f^1(p), \ldots$, definiert durch $f^0(p) := p$ und $\forall i \in \mathbb{N}. f^{i+1}(p) := f(f^i(p))$, konvergiert dann gegen $\check{x}$. Entsprechend konvergiert die Folge $f^0(q), f^1(q), \ldots$ gegen $\hat{x}$.

Beweis: Der Beweis wird in zwei Teile gegliedert. In Teil I wird gezeigt, dass die Fixpunkte von f einen Verband mit Minimum $\check{x}$ und Maximum $\hat{x}$ bilden, und dass dieser Verband vollständig ist, wenn f stetig ist. In Teil II wird die Konvergenz der angegebenen Iterationen gegen das Minimum bzw. Maximum bewiesen. Da die Beweisschritte für das Minimum und das Maximum dual sind, werden hier nur erstere ausführlich behandelt.

[1] Die leere Menge wird von der Bedingung ausgeklammert, da sonst aus der Definition folgen würde, dass $f(\bot) = \bot$ und $f(\top) = \top$, was eine unnötige Einschränkung wäre.

Teil I

Seien $\mathcal{P}_f := \{x \in \mathcal{D} | f(x) \sqsubseteq x\}$ die Menge aller Post-Fixpunkte und $\mathcal{F}_f := \{x \in \mathcal{D} | f(x) = x\}$ die Menge aller Fixpunkte von f. Wegen der angenommenen Vollständigkeit von $(\mathcal{D}, \sqsubseteq)$ existieren die Elemente $\check{x}_{\mathcal{P}} := \mathsf{inf}\left(\mathcal{P}_f\right)$ und $\check{x}_{\mathcal{F}} := \mathsf{inf}\left(\mathcal{F}_f\right)$. Es muss nun folgendes gezeigt werden:

(*i*): Die Menge $\mathcal{P}_f$ hat ein Minimum, und dieses ist gleich $\check{x}_{\mathcal{P}}$.
(*ii*): $\check{x}_{\mathcal{P}}$ ist gleichzeitig das Minimum von $\mathcal{F}_f$.

Zu (*i*): Sei x ein Element der Menge $\mathcal{P}_f$. Dann gilt (1) $f(x) \sqsubseteq x$ und, laut Definition A.9, $\check{x}_{\mathcal{P}} \sqsubseteq x$. Da f per Annahme monoton ist, folgt daraus (2) $f(\check{x}_{\mathcal{P}}) \sqsubseteq f(x)$. Aus (2) und (1) folgt nun wegen der Transitivität von $\sqsubseteq$ $f(\check{x}_{\mathcal{P}}) \sqsubseteq x$. Da dies für jedes $x \in \mathcal{P}_f$ gilt, ist $f(\check{x}_{\mathcal{P}})$ eine untere Schranke von $\mathcal{P}_f$. Da $\check{x}_{\mathcal{P}}$ die *größte* untere Schranke von $\mathcal{P}_f$ ist, gilt auch (3) $f(\check{x}_{\mathcal{P}}) \sqsubseteq \check{x}_{\mathcal{P}}$. Dies bedeutet aber, dass $\check{x}_{\mathcal{P}}$ selbst ein Element von $\mathcal{P}_f$ ist, und somit ist laut Lemma A.12 das Infimum $\check{x}_{\mathcal{P}}$ gleichzeitig das Minimum von $\mathcal{P}_f$.

Zu (*ii*): Da f per Annahme monoton ist, folgt aus (3): $f(f(\check{x}_{\mathcal{P}})) \sqsubseteq f(\check{x}_{\mathcal{P}})$. Also gilt $f(\check{x}_{\mathcal{P}}) \in \mathcal{P}_f$ und somit (4) $\check{x}_{\mathcal{P}} \sqsubseteq f(\check{x}_{\mathcal{P}})$. Aus (3) und (4) folgt nun (5) $\check{x}_{\mathcal{P}} = f(\check{x}_{\mathcal{P}})$. Dies bedeutet, dass $\check{x}_{\mathcal{P}}$ ein Fixpunkt von f ist, also (6) $\check{x}_{\mathcal{P}} \in \mathcal{F}_f$ und deshalb (7) $\check{x}_{\mathcal{F}} \sqsubseteq \check{x}_{\mathcal{P}}$. Andererseits ist aus den Definitionen von $\mathcal{P}_f$ und $\mathcal{F}_f$ ersichtlich, dass $\mathcal{F}_f \subseteq \mathcal{P}_f$. Aus Lemma A.13, Punkt (*vi*) folgt dann (8) $\check{x}_{\mathcal{P}} \sqsubseteq \check{x}_{\mathcal{F}}$. Aus (7) und (8) folgt (9) $\check{x}_{\mathcal{F}} = \check{x}_{\mathcal{P}}$. Aus (6) folgt nun $\check{x}_{\mathcal{F}} \in \mathcal{F}_f$, und deshalb ist laut Lemma A.12 das Infimum $\check{x}_{\mathcal{F}}$ gleichzeitig das Minimum von $\mathcal{F}_f$.

Zusammenfassend gilt also: $\mathcal{F}_f$ und $\mathcal{P}_f$ haben ein gemeinsames Infimum, das gleichzeitig ihr Minimum ist:

$$\check{x} = \mathsf{inf}\left(\{x \in \mathcal{D} | f(x) = x\}\right) = \mathsf{inf}\left(\{x \in \mathcal{D} | f(x) \sqsubseteq x\}\right).$$

Nun muss noch bewiesen werden, dass die Stetigkeit von f die Vollständigkeit von $(\mathcal{F}_f, \sqsubseteq)$ impliziert. Zu zeigen ist, dass dann Infimum und Supremum jeder Menge $M_f \subseteq \mathcal{F}_f$ in $\mathcal{F}_f$ enthalten sind. Wenn f stetig ist, gilt $f\left(\mathsf{inf}\left(M_f\right)\right) = \mathsf{inf}\left(\{f(x) | x \in M_f\}\right) = \mathsf{inf}\left(\{x | x \in M_f\}\right) = \mathsf{inf}\left(M_f\right)$. Also ist $\mathsf{inf}\left(M_f\right)$ selbst ein Fixpunkt und damit in $\mathcal{F}_f$. Das gleiche gilt für $\mathsf{sup}\left(M_f\right)$.

Teil II

Sei $C_p := \{f^n(p) | n \in \mathbb{N}\}$ die Menge der Elemente, die bei der Iteration $f^0(p), f^1(p), \ldots$ durchlaufen werden. Dann folgt aus der Vollständigkeit von $(\mathcal{D}, \sqsubseteq)$, dass $\mathsf{inf}\left(C_p\right)$ und $\mathsf{sup}\left(C_p\right)$ existieren (nebenbei bemerkt, $\mathsf{inf}\left(C_p\right) = p$).

Nun wird folgendes Zwischenresultat benötigt: $\mathsf{sup}\left(\{f^n(p) | n \in \mathbb{N}\}\right) = \mathsf{sup}\left(\{f^{n+1}(p) | n \in \mathbb{N}\}\right)$. Um dies zu zeigen, werden die Abkürzungen $m_1 := \mathsf{sup}\left(C_p\right) = \mathsf{sup}\left(\{f^n(p) | n \in \mathbb{N}\}\right)$ und $m_2 := \mathsf{sup}\left(\{f^{n+1}(p) | n \in \mathbb{N}\}\right)$ eingeführt. Aus Definition A.9 folgen:

$$\begin{array}{llll} (10) & \forall n \in \mathbb{N}. f^n(p) \sqsubseteq m_1 & (11) & \forall m \in \mathcal{D}. \forall n \in \mathbb{N}. f^n(p) \sqsubseteq m. m_1 \sqsubseteq m \\ (12) & \forall n \in \mathbb{N}. f^{n+1}(p) \sqsubseteq m_2 & (13) & \forall m \in \mathcal{D}. \forall n \in \mathbb{N}. f^{n+1}(p) \sqsubseteq m. m_2 \sqsubseteq m \end{array}$$

Da $n \in \mathbb{N} \rightarrow (n+1) \in \mathbb{N}$ folgt aus (10) $\forall n \in \mathbb{N}. f^{n+1}(p) \sqsubseteq m_1$. Durch *modus ponens* mit (13) folgt (14) $m_2 \sqsubseteq m_1$. Andererseits bedeutet (12) (für $n = 0$) $f(p) \sqsubseteq m_2$. Wegen der Transitivität von $\sqsubseteq$ folgt aus der Annahme (μ_1) des Theorems (15) $p \sqsubseteq m_2$. (12) und (15) bedeuten zusammen, dass $\forall n \in \mathbb{N}. f^n(p) \sqsubseteq m_2$. Durch *modus ponens* mit (11) folgt (16) $m_1 \sqsubseteq m_2$. Aus (14) und (16) folgt nun (17) $m_1 = m_2$.

Als nächstes wird mit Hilfe des Induktionsprinzips bewiesen, dass C_p eine gerichtete Menge ist. Als Induktionsbasis gilt die Annahme (μ_1), also $f^0(p) \sqsubseteq f^1(p)$. Der Induktionsschritt folgt: Angenommen, $f^i(p) \sqsubseteq f^{i+1}(p)$. Da f monoton ist (siehe Lemma A.24), folgt $f(f^i(p)) \sqsubseteq f(f^{i+1}(p))$, d.h., $f^{i+1}(p) \sqsubseteq f^{i+2}(p)$. Damit ist die Folge $p \sqsubseteq f(p) \sqsubseteq f^2(p) \sqsubseteq \ldots$ monoton. Wegen der Transitivität von $\sqsubseteq$ sind die Elemente einer solchen Folge stets paarweise vergleichbar, d.h., gegeben zwei Zahlen i, j gilt $i \leq j \rightarrow f^i(p) \sqsubseteq f^j(p)$. Die Elemente $f^j(p)$ bzw. $f^i(p)$ erfüllen dann die Bedingungen in Definition A.19, also ist C_p eine gerichtete Menge.

Damit lässt sich die Definition von Stetigkeit (in der folgenden Gleichung an der mit S gekennzeichneten Stelle) auf die Berechnung von $f\left(\sup\left(C_p\right)\right)$ anwenden. Es folgt:

$$f\left(\sup\left(C_p\right)\right) = f\left(\sup\left(\{f^n(p)|n \in \mathbb{N}\}\right)\right) \overset{(S)}{=} \sup\left(\{f(f^n(p))|n \in \mathbb{N}\}\right) =$$
$$= \sup\left(\{f^{n+1}(p)|n \in \mathbb{N}\}\right) \overset{(17)}{=} \sup\left(\{f^n(p)|n \in \mathbb{N}\}\right) = \sup\left(C_p\right).$$

Also ist $\sup\left(C_p\right)$ ein Fixpunkt von f, und damit gilt (18) $\check{x} \sqsubseteq \sup\left(C_p\right)$.

Als letztes wird mit Hilfe des Induktionsprinzips gezeigt, dass auch die Umkehrung von (18) gilt. Als Induktionsbasis gilt die Annahme (μ_2), also $f^0(p) \sqsubseteq \check{x}$. Angenommen, es gilt $f^i(p) \sqsubseteq \check{x}$. Da f Monoton ist, folgt $f^{i+1}(p) \sqsubseteq f(\check{x})$. Da $\check{x}$ selbst ein Fixpunkt von f ist, folgt daraus $f^{i+1}(p) \sqsubseteq \check{x}$, also (19) $\forall n \in \mathbb{N}.f^n(p) \sqsubseteq \check{x}$. Damit ist $\check{x}$ eine obere Schranke von C_p. Da $\sup\left(C_p\right)$ die *kleinste* obere Schranke von C_p ist, gilt (20) $\sup\left(C_p\right) \sqsubseteq \check{x}$. Aus (18) und (20) folgt nun (21) $\sup\left(C_p\right) = \check{x}$.

Da $\check{x}$ der kleinste Fixpunkt von f ist, kann die Folge der $f^i(p)$ nicht stehen bleiben, bevor sie $\check{x}$ erreicht hat. Andererseits kann sie wegen (19) dieses Element auch nicht überspringen, und somit konvergiert sie gegen $\check{x}$. □

Korollar A.27 *Ist $(\mathcal{D}, \sqsubseteq)$ endlich, können $\check{x}$ bzw. $\hat{x}$ mit Hilfe der angegebenen Iterationen berechnet werden.*

Beweis: Ist $\mathcal{D}$ endlich, muss die Folge der f^i den Wert $\sup\left(C_p\right) = \check{x}$ nach einer begrenzten Anzahl Iterationen erreichen. □

Zum Schluss wird noch gezeigt, dass auf endlichen Verbänden jede monotone Funktion auch stetig ist. Somit reicht für die Berechnung der Fixpunkte bei endlichen Verbänden aus, dass f monoton ist. Zunächst wird gezeigt, unter welcher Bedingung eine monotone Funktion auch stetig ist.

Lemma A.28 (Monotonie impliziert Stetigkeit – ausreichende Bedingung) *Gegeben sei eine Ordnungsrelation $(\mathcal{D}, \sqsubseteq)$. Falls für jede nicht leere gerichtete Menge $M \subseteq \mathcal{D}$ gilt, dass $\sup(M) \in M$ und $\inf(M) \in M$, dann ist jede monotone Funktion $f : \mathcal{D} \to \mathcal{D}$ stetig.*

Beweis: Sei $M \subseteq \mathcal{D}$ eine gerichtete Menge. Es muss gezeigt werden, dass die Monotonie von f unter der gegebenen Bedingung verlangt, dass $f(\sup(M)) = \sup(\{f(x)|x \in M\})$ und $f(\inf(M)) = \inf(\{f(x)|x \in M\})$.

Laut Definition A.9 gilt $\forall x \in M.x \sqsubseteq \sup(M)$. Da f monoton ist, folgt $\forall x \in M.f(x) \sqsubseteq f(\sup(M))$. Also ist $f(\sup(M))$ eine obere Schranke von $M' = \{f(x)|x \in M\}$. Wenn nun zusätzlich gilt, dass $\sup(M) \in M$, dann ist $f(\sup(M))$ kleinste obere Schranke von M', also ist $f(\sup(M)) = \sup(\{f(x)|x \in M\})$.

Der Beweis für das Infimum ist ähnlich. □

Beispiel A.29 Die ausreichende Bedingung in Lemma A.28 kann durch folgendes Gegenbeispiel verdeutlicht werden: Gegeben sei ein unendlicher Verband $(\mathcal{D} \sqsubseteq)$, wobei $\mathcal{D} = \{\mathbb{N} \cup \{a, b, c\}\}$, $\forall n \in \mathbb{N}.n \sqsubseteq c$ und $c \sqsubseteq b \sqsubseteq a$. $M \subseteq \mathcal{D}$ sei die Menge der geraden Zahlen. Dann ist M gerichtet (siehe Definition A.19), hat aber kein Maximum. Es gilt $\sup(M) = c$, zumal c die Bedingungen von Definition A.9 erfüllt. Die Elemente der Menge M werden auf $M' = \{f(x)|x \in M\}$ abgebildet. Sei nun f wie folgt definiert:

$$\begin{cases} \forall x \in \mathcal{D} \setminus \{c\} & f(x) = x \\ & f(c) = b. \end{cases}$$

Das Verhältnis zwischen den genannten Elementen zeigt Abbildung A.2.

Dann ist f monoton, aber nicht stetig, weil $f(\sup(M)) = f(c) = b$, während $\sup(M') = \sup(\{f(x)|x \in M\}) = \sup(M) = c$. Das heißt, wenn $\sup(M) \notin M$, dann kann f das Element $\sup(M)$ auf ein Element abbilden, das strikt über $\sup(\{f(x)|x \in M\})$ liegt (in diesem Fall b). Allerdings muss dies nicht passieren, denn es gibt natürlich auch monotone Funktionen, bei denen $\sup(M)$ auf sich selbst abgebildet wird.

Anders ist es, falls $\sup(M) \in M$. Dann gilt $f(\sup(M)) \in M'$, und wegen der Monotonie von f muss $f(\sup(M))$ das Maximum (und damit auch das Supremum) von M' sein. □

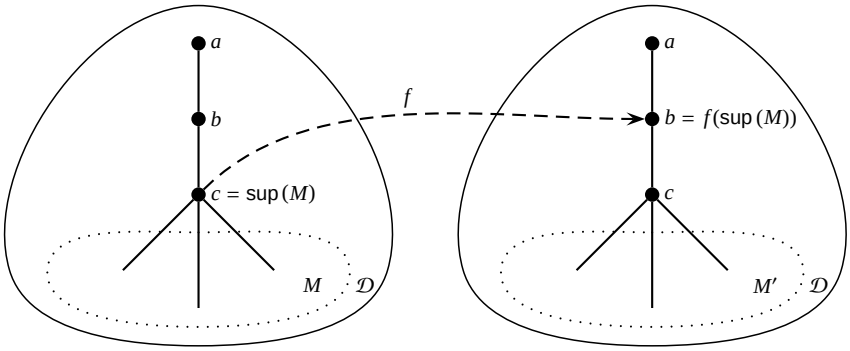


Abbildung A.2: Monotonie impliziert nicht unbedingt Stetigkeit

Theorem A.30 (Äquivalenz zwischen Monotonie und Stetigkeit) *Gegeben sei eine Ordnungsrelation* $(\mathcal{D}, \sqsubseteq)$ *. Falls für jede nicht leere gerichtete Menge* $M \subseteq \mathcal{D}$ *gilt, dass* $\sup(M) \in M$ *und* $\inf(M) \in M$*, dann sind Monotonie und Stetigkeit einer gegebenen Funktion* $f : \mathcal{D} \to \mathcal{D}$ *äquivalent.*

Beweis:
f stetig $\to$ f monoton: Folgt bedingungslos aus Lemma A.24.
f monoton $\to$ f stetig: Folgt unter der gegebenen Bedingung aus Lemma A.28. □

Korollar A.31 (Monotonie und Stetigkeit bei endlichen Ordnungsrelationen) *Gegeben sei eine* endliche *Ordnungsrelation* $(\mathcal{D}, \sqsubseteq)$*. Dann sind Monotonie und Stetigkeit jeder Funktion* $f : \mathcal{D} \to \mathcal{D}$ *äquivalent.*

Beweis: Die ausreichende Bedingung für die Äquivalenz zwischen Monotonie und Stetigkeit folgt aus Lemma A.21. □

A.4 Einschränkung auf Potenzmengen

Lemma A.32 (Potenzmengen bilden Verbände) *Sei* $\mathcal{P}$ *eine Menge, und seien* $A, B \subseteq \mathcal{P}$*. Dann ist* $(2^{\mathcal{P}}, \subseteq)$ *ein Verband mit den Operationen* $\sup(\{A, B\}) = A \cup B$ *und* $\inf(\{A, B\}) = A \cap B$*.*

Beweis: Es muss gezeigt werden, dass (*i*) $\subseteq$ eine Ordnungsrelation auf $2^{\mathcal{P}}$ ist, und (*ii*) dass $A \cup B$ bzw. $A \cap B$ das Supremum bzw. das Infimum von $\{A, B\}$ ist.

Zu (*i*): $\subseteq$ erfüllt die Bedingungen von Definition A.1 und bildet somit eine Ordnungsrelation auf $2^{\mathcal{P}}$.

Zu (*ii*): Für $A \cup B$ gelten $A \subseteq A \cup B$ und $B \subseteq A \cup B$, also ist $A \cup B$ eine obere Schranke von $\{A, B\}$. Außerdem gilt $\forall M.A \subseteq M \wedge B \subseteq M.A \cup B \subseteq M$, also ist $A \cup B$ die kleinste obere Schranke von $\{A, B\}$.

Der Beweis für $A \cap B$ ist ähnlich. □

Lemma A.33 (Supremum und Infimum für Potenzmengen-Verbände) *Seien* $(2^{\mathcal{P}}, \subseteq)$ *ein Verband und, für eine gegebene Indexmenge* I*, die Mengen* $A_i \subseteq \mathcal{P}$*,* $i \in I$*. Dann gilt* $\sup(\{A_i | i \in I\}) = \bigcup_{i \in I} A_i$ *und* $\inf(\{A_i | i \in I\}) = \bigcap_{i \in I} A_i$*.*

Beweis: Der Beweis benutzt das Induktionsprinzip: Gegeben zwei Mengen $A_1, A_2 \subseteq \mathcal{P}$, gilt laut Lemma A.32 $\sup(\{A, B\}) = A \cup B$. Angenommen, es gilt $\sup(\{A_1, \ldots, A_n\}) = \bigcup_{i=1}^{n} A_i$. Aus Theorem A.17 folgt $\sup(\{A_1, \ldots, A_{n+1}\}) = \sup\left(\left\{\bigcup_{i=1}^{n} A_i, A_{n+1}\right\}\right) = \bigcup_{i=1}^{n} A_i \cup A_{n+1} = \bigcup_{i=1}^{n+1} A_i$. Der Beweis für das Infimum ist analog. □

Korollar A.34 (Tarski-Knaster-Theorem für endliche Potenzmengen-Verbände) *Seien $\mathcal{P}$ eine endliche Menge und $f : 2^{\mathcal{P}} \to 2^{\mathcal{P}}$ eine monotone Funktion auf dem Verband $(2^{\mathcal{P}}, \subseteq)$. Dann hat f Fixpunkte, und diese bilden einen vollständigen Verband mit folgendem Minimum bzw. Maximum:*

$$\begin{array}{rclcl} \check{x} & = & \bigcap\{X \subseteq \mathcal{P} | f(X) = X\} & = & \bigcap\{X \subseteq \mathcal{P} | f(X) \subseteq X\} \\ \hat{x} & = & \bigcup\{X \subseteq \mathcal{P} | f(X) = X\} & = & \bigcup\{X \subseteq \mathcal{P} | X \subseteq f(X)\}. \end{array}$$

Seien ferner $P, Q \in \mathcal{P}$ Elemente, die folgende Bedingungen erfüllen:

$$\begin{array}{llll} (\mu_1) & P \sqsubseteq f(P) & (\nu_1) & f(Q) \sqsubseteq Q \\ (\mu_2) & P \sqsubseteq \check{x} & (\nu_2) & \hat{x} \sqsubseteq Q. \end{array}$$

Dann können $\check{x}$ bzw. $\hat{x}$ iterativ berechnet werden, indem $X_0 := P$ bzw. $X_0 := Q$ als Ausgangspunkt gewählt wird. Die Folge $X_{i+1} := f(X_i)$ erreicht dann den entsprechenden Fixpunkt.

Beweis: Die Ausdrücke für $\check{x}$ und $\hat{x}$ folgen unmittelbar aus Theorem A.26 und Lemma A.33. Da $\mathcal{P}$ endlich ist, folgt aus der Monotonie von f auch die Stetigkeit (siehe Korollar A.31). Die Endlichkeit der Menge $\mathcal{P}$ sorgt auch dafür, dass die Fixpunkte erreicht werden. □

In den Fällen, in denen keine Annäherung der Fixpunkte bekannt ist, die als Anfangswert für die Iteration dienen könnte, werden als Anfangswerte $\emptyset$ bzw. $\mathcal{P}$ gewählt. Diese erfüllen auf jeden Fall die Bedingungen μ_1 und μ_2 bzw. ν_1 und ν_2.

Lemma A.35 (Kleinstes und größtes Element in Potenzmengen-Verbänden) *Sei $(2^{\mathcal{P}}, \subseteq)$, ein Verband, und $\mathcal{P}$ endlich. Dann sind $\bot = \emptyset$ und $\top = \mathcal{P}$.*

Beweis: Diese Elemente lassen sich mit Hilfe von Lemma A.33 berechnen:
$\top = \sup\left(\{X | X \in 2^{\mathcal{P}}\}\right) = \bigcup\{X | X \in 2^{\mathcal{P}}\} = \mathcal{P}$ bzw.
$\bot = \inf\left(\{X | X \in 2^{\mathcal{P}}\}\right) = \bigcap\{X | X \in 2^{\mathcal{P}}\} = \emptyset$. □

Anmerkung: Es gelten laut Lemma A.16 $\sup(\emptyset) = \sup(\{\}) = \emptyset$ und $\inf(\emptyset) = \inf(\{\}) = \mathcal{P}$.
Achtung! $\sup(\emptyset) = \sup(\{\})$ bzw. $\inf(\emptyset) = \inf(\{\})$ dürfen nicht mit $\sup(\{\emptyset\})$ bzw. $\inf(\{\emptyset\})$ verwechselt werden! Es gilt $\sup(\{\emptyset\}) = \inf(\{\emptyset\}) = \emptyset$ (siehe Lemma A.16). □

A.5 Zusammenfassung

Die hier behandelten Grundlagen und die wichtigsten Erkenntnisse zu dem Tarski-Knaster-Theorem lassen sich wie folgt zusammenfassen:

- Extrema in Halbordnungsrelationen sind Maximum, Minimum, Supremum und Infimum; falls es diese gibt, sind sie eindeutig.
- Ein Verband ist eine Halbordnungsrelation in der es für je zwei Elemente ein Supremum und ein Infimum gibt.
- Ein Verband ist vollständig, falls es für jede Menge ein Supremum und ein Infimum gibt.
- Endliche Verbände sind vollständig.
- Eine Menge ist gerichtet g.d.w. es für je zwei Elemente ein Maximum und ein Minimum gibt.
- Endliche gerichtete Mengen haben ein Maximum und ein Minimum;.
- Monotone Funktionen auf Verbänden sind stetig.

- Stetige Funktionen auf Verbänden sind monoton, wenn jede gerichtete Menge ihr eigenes Supremum und Infimum enthält; dies ist insbesondere bei endlichen gerichteten Mengen der Fall.
- Damit sind Monotonie und Stetigkeit auf endlichen Verbänden äquivalent.
- Das Tarski-Knaster-Theorem gilt auf vollständigen Verbänden, die endlich oder unendlich sein können.
- Ist eine Funktion f monoton, so hat sie Fixpunkte, die einen Verband bilden. Ist f darüber hinaus stetig, so ist dieser Verband vollständig und die Iteration aus Theorem A.26 konvergiert gegen $\check{x}$ bzw. $\hat{x}$.
- Nur wenn der Verband endlich ist, muss die Iteration die Werte $\check{x}$ bzw. $\hat{x}$ tatsächlich erreichen.
- Ist der Verband endlich, so genügt für die Berechnung von $\check{x}$ und $\hat{x}$, dass f monoton ist (denn unter dieser Bedingung sind Monotonie und Stetigkeit äquivalent).

Anhang B

Theoreme zur Lösung von Gleichungssystemen

In diesem Anhang werden die Theoreme bewiesen, mit deren Hilfe die Korrektheit und die Komplexität der Algorithmen in Kapitel 3 nachgewiesen wird. Da auf die Algorithmen 3.54 und 3.57 häufig Bezug genommen wird, werden diese hier als Algorithmen B.1 bzw. B.2 wiedergegeben:

Algorithmus B.1 (1. Lösung eines Gleichungssystems aus einer maximalen Komponente) *Sei*

$$E := \left\{ \begin{array}{lcl} u_n & \stackrel{\sigma_n}{=} & \varphi_n(u_1, \ldots, u_n) \\ & \vdots & \\ u_1 & \stackrel{\sigma_1}{=} & \varphi_1(u_1, \ldots, u_n) \end{array} \right.$$

ein Gleichungssystem, dessen Abhängigkeitsgraph aus einer maximalen streng zusammenhängenden Komponente besteht. Dann kann E wie folgt gelöst werden:

$$\left[\begin{array}{lcl} u_1^{i_1+1} & := & \varphi_1(u_1^{i_1}, \ldots, u_n^{i_1,\infty,\ldots,\infty}) \\ & \vdots & \\ u_n^{i_1,\ldots,i_n+1} & := & \varphi_n(u_1^{i_1}, \ldots, u_n^{i_1,\ldots,i_n}), \end{array} \right. \tag{B.1}$$

wobei die Initialwerte $u_k^{i_1,\ldots,i_{k-1},0}$ *bei* $\sigma_k = \mu$ *gleich* 0 *und bei* $\sigma_k = \nu$ *gleich* 1 *gesetzt werden.*

Algorithmus B.2 (2. Lösung von Gleichungssystemen aus einer maximalen Komponente) *Sei*

$$E := \left\{ \begin{array}{lcl} u_n & \stackrel{\sigma_n}{=} & \varphi_n(u_1, \ldots, u_n) \\ & \vdots & \\ u_1 & \stackrel{\sigma_1}{=} & \varphi_1(u_1, \ldots, u_n) \end{array} \right.$$

ein Gleichungssystem, dessen Abhängigkeitsgraph aus einer maximalen streng zusammenhängenden Komponente besteht. Dann kann E wie folgt gelöst werden:

$$\left[\begin{array}{lcl} u_1^{i_1+1} & := & \varphi_1(u_1^{i_1}, \ldots, u_n^{i_1,\infty,\ldots,\infty}) \\ & \vdots & \\ u_n^{i_1,\ldots,i_n+1} & := & \varphi_n(u_1^{i_1}, \ldots, u_n^{i_1,\ldots,i_n}). \end{array} \right. \tag{B.2}$$

Die Anfangswerte $u_k^{0,\ldots,0}$ *für die erste Berechnung von* u_k *werden bei* $\sigma_k = \mu$ *gleich* 0 *und bei* $\sigma_k = \nu$ *gleich* 1 *gesetzt. Danach werden die Anfangswerte* $u_{k+1}^{i_1,\ldots,i_k,0}$ *dann und nur dann auf* 0 *bzw.* 1 *zurückgesetzt, wenn* $\sigma_k \neq \sigma_{k+1}$.

Folgende Definition ist notwendig, um Ergebnisse verschiedener Algorithmen zu vergleichen.

Definition B.3 (Ordnungsrelation über boolesche Ausdrücke) *Gegeben eine Kripke-Struktur $\mathcal{K}$ und Ausdrücke $\varphi_P, \varphi_Q \in \mathcal{L}_\mu$, wird die Halbordnungsrelation $\sqsubseteq$ so definiert, dass*

$$\varphi_P \sqsubseteq \varphi_Q \Leftrightarrow [\![\varphi_P]\!]_{\mathcal{K}} \subseteq [\![\varphi_Q]\!]_{\mathcal{K}}.$$

Definition B.3 erlaubt es außerdem, den Begriff der Monotonie auf symbolischer Ebene anzuwenden. Da die Ausdrücke φ_k in den Gleichungssystemen monoton sind, gilt z.B.

$$v_g^i \sqsubseteq v_g^j \Rightarrow \varphi_k(v_1^i, \ldots, v_g^i, \ldots, v_n^i) \sqsubseteq \varphi_k(v_1^i, \ldots, v_g^j, \ldots, v_n^i).$$

Das folgende Lemma wird in Kapitel 3 benötigt, um die Verringerung der Komplexität der Modellprüfung von Emerson und Lei zu erklären.

Lemma B.4 (Übertragung der Monotonie auf verschachtelte Fixpunkte) *Sei das Gleichungssystem*

$$E := \begin{cases} u_n & \overset{\sigma_n}{=} & \varphi_n(u_1, \ldots, u_n) \\ & \vdots & \\ u_1 & \overset{\sigma_1}{=} & \varphi_1(u_1, \ldots, u_n), \end{cases}$$

das aus einer maximalen Komponente besteht und in dem $\sigma_k \in \{\mu, \nu\}$ für $1 \leq k \leq n$. Gegeben die Formeln B.1, gilt

$$\begin{aligned} &\text{für } \sigma_k = \mu\text{:} \quad u_{k+1}^{i_1,\ldots,i_k-1,\infty} \sqsubseteq u_{k+1}^{i_1,\ldots,i_k,\infty} \\ &\text{für } \sigma_k = \nu\text{:} \quad u_{k+1}^{i_1,\ldots,i_k,\infty} \sqsubseteq u_{k+1}^{i_1,\ldots,i_k-1,\infty}. \end{aligned}$$

Beweis: Der Beweis wird für $\sigma_k = \mu$ gegeben, der Fall $\sigma_k = \nu$ ist ähnlich.

Basis: Aus dem Tarski-Knaster-Theorem folgt, dass $u_k^{i_1,\ldots,i_k-1} \sqsubseteq u_k^{i_1,\ldots,i_k}$, also gilt für φ_n:

$$\begin{aligned} &\varphi_n(u_1^{i_1}, \ldots, u_k^{i_1,\ldots,i_k-1}, u_{k+1}^{i_1,\ldots,i_k-1,0}, \ldots, u_n^{i_1,\ldots,i_k-1,0,\ldots,0}) \quad \sqsubseteq \\ &\varphi_n(u_1^{i_1}, \ldots, u_k^{i_1,\ldots,i_k}, u_{k+1}^{i_1,\ldots,i_k,0}, \ldots, u_n^{i_1,\ldots,i_k,0,\ldots,0}). \end{aligned}$$

Werden die Iterationen auf beiden Seiten fortgesetzt bis die Fixpunkte erreicht sind, folgt unabhängig von σ_n

$$u_n^{i_1,\ldots,i_k-1,0,\ldots,0,\infty} \sqsubseteq u_n^{i_1,\ldots,i_k,0,\ldots,0,\infty}.$$

Für φ_{n-1} gilt nun

$$\begin{aligned} &\varphi_{n-1}(u_1^{i_1}, \ldots, u_k^{i_1,\ldots,i_k-1}, u_{k+1}^{i_1,\ldots,i_k-1,0}, \ldots, u_{n-1}^{i_1,\ldots,i_k-1,0,\ldots,0,0}, u_n^{i_1,\ldots,i_k-1,0,\ldots,0,\infty}) \quad \sqsubseteq \\ &\varphi_{n-1}(u_1^{i_1}, \ldots, u_k^{i_1,\ldots,i_k}, u_{k+1}^{i_1,\ldots,i_k,0}, \ldots, u_{n-1}^{i_1,\ldots,i_k,0,\ldots,0,0}, u_n^{i_1,\ldots,i_k,0,\ldots,0,\infty}), \end{aligned}$$

also, unabhängig von σ_{n-1},

$$u_{n-1}^{i_1,\ldots,i_k-1,0,\ldots,0,1} \sqsubseteq u_{n-1}^{i_1,\ldots,i_k,0,\ldots,0,1}.$$

Unter Wiederholung der Iterationen für φ_n und φ_{n-1} nach B.1 folgt

$$\begin{aligned} u_{n-1}^{i_1,\ldots,i_k-1,0,\ldots,0,\infty} &\sqsubseteq u_{n-1}^{i_1,\ldots,i_k,0,\ldots,0,\infty} \\ u_n^{i_1,\ldots,i_k-1,0,\ldots,0,\infty,\infty} &\sqsubseteq u_n^{i_1,\ldots,i_k,0,\ldots,0,\infty,\infty} \end{aligned}.$$

Die Berechnungen können nun für φ_{n-2} fortgesetzt werden, woraus folgt

$$\begin{aligned} u_{n-2}^{i_1,\ldots,i_k-1,0,\ldots,0,\infty} &\sqsubseteq u_{n-2}^{i_1,\ldots,i_k,0,\ldots,0,\infty} \\ u_{n-1}^{i_1,\ldots,i_k-1,0,\ldots,0,\infty,\infty} &\sqsubseteq u_{n-1}^{i_1,\ldots,i_k,0,\ldots,0,\infty,\infty} \\ u_n^{i_1,\ldots,i_k-1,0,\ldots,0,\infty,\infty,\infty} &\sqsubseteq u_n^{i_1,\ldots,i_k,0,\ldots,0,\infty,\infty,\infty} \end{aligned}.$$

In dieser Weise werden die Nullen in den Iterationszählern von rechts nach links schrittweise durch ∞ ersetzt, bis für alle u_ℓ mit $k < \ell \leq n$ gilt

$$u_\ell^{i_1,\ldots,i_k-1,\infty,\ldots,\infty} \sqsubseteq u_\ell^{i_1,\ldots,i_k,\infty,\ldots,\infty}.$$

Insbesondere gilt für $\ell = k+1$

$$u_{k+1}^{i_1,\ldots,i_k-1,\infty} \sqsubseteq u_{k+1}^{i_1,\ldots,i_k,\infty},$$

was zu beweisen war. □

Es folgt das Theorem über die Komplexität von Algorithmus B.2.

Theorem B.5 (Komplexität der μ-Kalkül-Modellprüfung nach Algorithmus B.2) *Gegeben eine Kripke-Struktur $\mathcal{K} = \langle \mathcal{S}, \mathcal{I}, \mathcal{R}, \mathcal{L} \rangle$ und ein Gleichungssystem E der der Größe $|E|$ und Alternierungstiefe ℓ über den Variablen von $\mathcal{K}$, wird die Lösung von E von Algorithmus B.2 innerhalb der Zeit $O\left(\left(\frac{|E||\mathcal{S}|}{\ell}\right)^\ell |\mathcal{R}|\,|E|\right)$ berechnet.*

Beweis: Algorithmus B.2 sieht vor, dass die Anfangswerte der Iterationen dann und nur dann auf 0 bzw. 1 zurückgesetzt werden, wenn eine Teilblockgrenze überschritten wird. Jedes Mal, wenn die Berechnung der ersten Gleichung eines Teilblocks i zurückgesetzt wird, können die anderen Gleichungen im selben Teilblock jeweils höchstens $|\mathcal{S}|$ Iterationsschritte bis zum Fixpunkt machen. Für einen Teilblock mit g_i Gleichungen fallen deshalb höchstens $g_i\,|\mathcal{S}|$ Iterationen an, wenn die erste Gleichung im Teilblock einen Schritt macht. Durch die Verschachtelung entstehen maximal $g_1\,|\mathcal{S}|\,.g_2\,|\mathcal{S}|\ldots g_\ell\,|\mathcal{S}| = \Pi_{i=1}^{\ell}(g_i\,|\mathcal{S}|)$ Schritte. Die Ungleichung zwischen dem arithmetischen und dem geometrischen Mittel besagt, dass

$$\sqrt[l]{\Pi_{i=1}^{\ell}(g_i\,|\mathcal{S}|)} \leq \frac{\Sigma_{i=1}^{\ell}(g_i\,|\mathcal{S}|)}{\ell} = \frac{|\mathcal{S}|\,\Sigma_{i=1}^{\ell} g_i}{\ell} = \frac{n\,|\mathcal{S}|}{\ell}.$$

Wird $|E|$ als obere Grenze für n eingesetzt folgt, dass die Anzahl der Iterationsschritte für den größten Teilblock unterhalb der Grenze $\left(\frac{|E||\mathcal{S}|}{\ell}\right)^\ell$ liegt.

Eine genauere Abschätzung der Dauer der Berechnungen müsste auch die Darstellungsform der Transitionsrelation berücksichtigen. Im einfachsten Fall ist diese direkt proportional zu $|R|$ und zu der Anzahl der Operationen, die durchgeführt werden müssen. Für letztere ist $|E|$ eine obere Grenze. Daraus ergibt sich für die Berechnungszeit die obere Grenze

$$O\left(\left(\frac{|E|\,|\mathcal{S}|}{\ell}\right)^\ell |\mathcal{R}|\,|E|\right).$$

Die Aussage des Theorems gilt auch für Gleichungssystemen mit mehreren maximalen Komponenten, denn laut Lemma 3.53 zerfällt deren Berechnung in die der einzelnen Komponenten. Die Alternierungstiefe ℓ des Gleichungssystems ist laut Definition 3.59 das Maximum der Alternierungstiefen der einzelnen Komponenten. Zudem ist $|E|$ eine obere Grenze für die Anzahl der Gleichungen in jeder Komponente mit Alternierungstiefe ℓ, also ist auch im Allgemeinen Fall die Berechnungszeit in $O\left(\left(\frac{|E||\mathcal{S}|}{\ell}\right)^\ell |\mathcal{R}|\,|E|\right)$. □

Folgendes Lemma wird für den Beweis von Theorem B.7 benötigt.

Lemma B.6 (Anordnung der Iterationswerte in gleichförmigen Gleichungssystemen) *Sei das Gleichungssystem*

$$E := \begin{cases} u_n & \overset{\sigma}{=} & \varphi_n(u_1,\ldots,u_n) \\ & \vdots & \\ u_1 & \overset{\sigma}{=} & \varphi_1(u_1,\ldots,u_n), \end{cases}$$

das aus einer maximalen Komponente besteht und in dem $\sigma \in \{\mu,\nu\}$. Gegeben die Formeln B.1 aus Algorithmus B.1 und Iterationszähler $i_1,\ldots,i_n$ bzw. $j_1,\ldots,j_n$, so dass $\forall a.i_a \leq j_a$, gilt für $1 \leq k \leq n$

$$\text{für } \sigma = \mu\text{:} \quad u_k^{i_1,\ldots,i_k} \sqsubseteq u_k^{j_1,\ldots,j_k} \tag{B.3}$$

$$\text{für } \sigma = \nu\text{:} \quad u_k^{j_1,\ldots,j_k} \sqsubseteq u_k^{i_1,\ldots,i_k}. \tag{B.4}$$

Beweis: Der Beweis wird für $\sigma = \mu$ gegeben, der Fall $\sigma = \nu$ ist ähnlich. Als Induktionsbasis dient

$$u_1^{i_1} \sqsubseteq u_1^{j_1},$$

was bei $i_1 \leq j_1$ aus dem Tarski-Knaster-Theorem folgt. Angenommen, bis zu dem Indexpaar (i_{k-1}, j_{k-1}) gilt

$$u_{k-1}^{i_1,\ldots,i_{k-1}} \sqsubseteq u_{k-1}^{j_1,\ldots,j_{k-1}},$$

folgt für φ_n:

$$\begin{array}{l} \varphi_n(u_1^{i_1}, \ldots, u_k^{i_1,\ldots,i_{k-1},0}, \ldots, u_n^{i_1,\ldots,i_{k-1},0,\ldots,0,0}) \quad \sqsubseteq \\ \varphi_n(u_1^{j_1}, \ldots, u_k^{j_1,\ldots,j_{k-1},0}, \ldots, u_n^{j_1,\ldots,j_{k-1},0,\ldots,0,0}) \end{array}$$

und, wenn die Iterationen auf beiden Seiten fortgesetzt werden, bis die Fixpunkte erreicht sind,

$$u_n^{i_1,\ldots,i_{k-1},0,\ldots,0,\infty} \sqsubseteq u_n^{j_1,\ldots,j_{k-1},0,\ldots,0,\infty}.$$

Für φ_{n-1} gilt nun

$$\begin{array}{l} \varphi_{n-1}(u_1^{i_1}, \ldots, u_k^{i_1,\ldots,i_{k-1},0}, \ldots, u_{n-1}^{i_1,\ldots,i_{k-1},0,\ldots,0,0}, u_n^{i_1,\ldots,i_{k-1},0,\ldots,0,0,\infty}) \quad \sqsubseteq \\ \varphi_{n-1}(u_1^{j_1}, \ldots, u_k^{j_1,\ldots,j_{k-1},0}, \ldots, u_{n-1}^{j_1,\ldots,j_{k-1},0,\ldots,0,0}, u_n^{j_1,\ldots,j_{k-1},0,\ldots,0,0,\infty}), \end{array}$$

also

$$u_{n-1}^{i_1,\ldots,i_{k-1},0,\ldots,0,1} \sqsubseteq u_{n-1}^{j_1,\ldots,j_{k-1},0,\ldots,0,1}.$$

Unter Wiederholung der Iterationen für φ_n und φ_{n-1} nach B.1 folgt

$$\begin{array}{rcl} u_{n-1}^{i_1,\ldots,i_{k-1},0,\ldots,0,\infty} & \sqsubseteq & u_{n-1}^{j_1,\ldots,j_{k-1},0,\ldots,0,\infty} \\ u_n^{i_1,\ldots,i_{k-1},0,\ldots,0,\infty,\infty} & \sqsubseteq & u_n^{j_1,\ldots,j_{k-1},0,\ldots,0,\infty,\infty}. \end{array}$$

Die Berechnungen können nun für φ_{n-2} fortgesetzt werden, woraus folgt

$$\begin{array}{rcl} u_{n-2}^{i_1,\ldots,i_{k-1},0,\ldots,0,\infty} & \sqsubseteq & u_{n-2}^{j_1,\ldots,j_{k-1},0,\ldots,0,\infty} \\ u_{n-1}^{i_1,\ldots,i_{k-1},0,\ldots,0,\infty,\infty} & \sqsubseteq & u_{n-1}^{j_1,\ldots,j_{k-1},0,\ldots,0,\infty,\infty} \\ u_n^{i_1,\ldots,i_{k-1},0,\ldots,0,\infty,\infty,\infty} & \sqsubseteq & u_n^{j_1,\ldots,j_{k-1},0,\ldots,0,\infty,\infty,\infty}. \end{array}$$

In dieser Form werden die Nullen in den Iterationszählern von rechts nach links schrittweise durch ∞ ersetzt, bis nur noch eine an der k-ten Stelle steht. Dann gilt

$$\begin{array}{l} \varphi_k(u_1^{i_1}, \ldots, u_k^{i_1,\ldots,i_{k-1},0}, u_{k+1}^{i_1,\ldots,i_{k-1},0,\infty}, \ldots, u_n^{i_1,\ldots,i_{k-1},0,\infty,\ldots,\infty}) \quad \sqsubseteq \\ \varphi_k(u_1^{j_1}, \ldots, u_k^{j_1,\ldots,j_{k-1},0}, u_{k+1}^{j_1,\ldots,j_{k-1},0,\infty}, \ldots, u_n^{j_1,\ldots,j_{k-1},0,\infty,\ldots,\infty}), \end{array}$$

also

$$u_k^{i_1,\ldots,i_{k-1},1} \sqsubseteq u_k^{j_1,\ldots,j_{k-1},1}.$$

Die oben beschriebenen Iterationen können beliebig oft durchgeführt werden, so dass

$$u_k^{i_1,\ldots,i_{k-1},\ell} \sqsubseteq u_k^{j_1,\ldots,j_{k-1},\ell}$$

und, für $i_k \leq j_k$,

$$u_k^{i_1,\ldots,i_{k-1},i_k} \sqsubseteq u_k^{j_1,\ldots,j_{k-1},j_k}.$$

□

Theorem B.7 (Vektorielle Berechnung von Gleichungssystemen mit Alternierungstiefe 1)
Sei

$$E := \left\{ \begin{array}{lcl} u_n & \stackrel{\sigma}{=} & \varphi_n(u_1, \ldots, u_n) \\ & \vdots & \\ u_1 & \stackrel{\sigma}{=} & \varphi_1(u_1, \ldots, u_n) \end{array} \right.$$

ein Gleichungssystem mit Alternierungstiefe 1 über den Variablen der Kripke-Struktur $\mathcal{K} = \langle \mathcal{S}, \mathcal{I}, \mathcal{R}, \mathcal{L} \rangle$, dessen Abhängigkeitsgraph aus einer maximalen streng zusammenhängenden Komponente besteht und in dem $\sigma \in \{\mu, \nu\}$. Gegeben die Anfangswerte $v_k^0 := 0$ für $\sigma = \mu$ bzw. $v_k^0 := 1$ für $\sigma = \nu$, $1 \leq k \leq n$, berechnen die Formeln

$$\left[\begin{array}{lcl} v_1^{i+1} & := & \varphi_1(v_1^i, \ldots, v_{n-1}^i, v_n^i) \\ & \vdots & \\ v_n^{i+1} & := & \varphi_n(v_1^{i+1}, \ldots, v_{n-1}^{i+1}, v_n^i) \end{array} \right. \tag{B.5}$$

innerhalb der Zeit $O\left(|E|^2 \, |\mathcal{S}| \, |\mathcal{R}|\right)$ die gleichen Fixpunkte wie Algorithmus B.2.

Beweis: Es wird gezeigt, dass (1) jede Gleichung aus B.5 in höchstens $|\mathcal{S}|$ Schritten zu einem Fixpunkt v_k^∞ konvergiert, und dass (2) $\forall k \in \{1, \ldots, n\}. v_k^\infty = u_k^{\infty,\ldots,\infty}$, wobei letztere die Fixpunkte aus der Formeln B.1 sind.

1. Die Existenz der Fixpunkte v_k^∞ folgt aus der Monotonie der Funktionen φ_k und aus der Endlichkeit der Menge $\mathcal{S}$. Da jede Gleichung maximal $|\mathcal{S}|$ Iterationsschritte machen kann bis ein Fixpunkt erreicht wird, ist die Berechnung von B.5 in maximal $n\,|\mathcal{S}|$ Schritten beendet. Wie in Theorem B.5 wird angenommen, dass die Berechnungsdauer eines Schrittes im einfachsten Fall direkt proportional zu $|E|$ und zu $|\mathcal{R}|$ ist. Außerdem ist $|E|$ eine obere Grenze für n, so dass sich die Zeitkomplexität $O\left(|E|^2 \, |\mathcal{S}| \, |\mathcal{R}|\right)$ ergibt.

2. Um zu zeigen, dass für $1 \leq k \leq n$

   $$v_k^\infty = u_k^{\infty,\ldots,\infty}, \tag{B.6}$$

   muss zwischen den Fällen $\sigma = \mu$ und $\sigma = \nu$ unterschieden werden. Im Folgenden wird nur der Beweis für $\sigma = \mu$ gegeben, der für $\sigma = \nu$ ist ähnlich.

   Aus dem Tarski-Knaster-Theorem (Theorem 3.27) folgt, dass die $u_k^{\infty,\ldots,\infty}$ stets die kleinsten Fixpunkte der φ_k sind. Somit genügt es, die Ungleichung

   $$v_k^\infty \sqsubseteq u_k^{\infty,\ldots,\infty} \tag{B.7}$$

   zu beweisen. Wenn diese gilt, dann muss auch Gleichung B.6 gelten, weil die v_k^∞ nicht strikt kleiner als die kleinsten Fixpunkte sein können.

   Um B.7 zu beweisen, wird induktiv gezeigt, dass

   $$\forall i. v_k^i \sqsubseteq u_k^{i,\ldots,i}. \tag{B.8}$$

   Als Basis dient auf Grund der angenommenen Anfangswerte

   $$v_k^0 \sqsubseteq u_k^{0,\ldots,0}.$$

   Induktion: Angenommen, $v_k^{i-1} \sqsubseteq u_k^{i-1,\ldots,i-1}$, folgt:

   $$\varphi_1\left(v_1^{i-1}, v_2^{i-1}, \ldots, v_n^{i-1}\right) \sqsubseteq \varphi_1\left(u_1^{i-1}, u_2^{i-1,i-1}, \ldots, u_n^{i-1,i-1,\ldots,i-1}\right) \stackrel{(B.3)}{\sqsubseteq} \varphi_1\left(u_1^{i-1}, u_2^{i-1,\infty}, \ldots, u_n^{i-1,\infty,\ldots,\infty}\right)$$
   $$v_1^i \sqsubseteq u_1^i. \tag{B.9}$$

   Ungleichung B.9 dient nun als Basis für folgenden Induktionsschritt: Angenommen, für $1 \leq g \leq k-1$ gilt $v_g^i \sqsubseteq u_g^{i,\ldots,i}$ und für $k \leq g \leq n$ gilt $v_g^{i-1} \sqsubseteq u_g^{i-1,\ldots,i-1}$, folgt:

   $$\begin{array}{rcl} v_k^i & := & \varphi_k(v_1^i, \ldots, v_{k-1}^i, v_k^{i-1}, \ldots, v_n^{i-1}) \sqsubseteq \\ & & \varphi_k(u_1^i, \ldots, u_{k-1}^{i,\ldots,i}, u_k^{i-1,\ldots,i-1,i-1}, \ldots, u_n^{i-1,\ldots,i-1}) \stackrel{(B.3)}{\sqsubseteq} \\ & & \varphi_k(u_1^i, \ldots, u_{k-1}^{i,\ldots,i}, u_k^{i,\ldots,i,i-1}, \ldots, u_n^{i,\ldots,i,i-1,\infty,\ldots,\infty}) = u_k^{i,\ldots,i}. \end{array}$$

Letztere ist Ungleichung B.8, aus der Ungleichung B.7 und dann die Aussage des Theorems unmittelbar folgen. □

Folgendes Theorem besagt, dass die Lösung gleichförmiger Gleichungssysteme gegenüber Vertauschungen der Gleichungen invariant ist.

Theorem B.8 (Vertauschbarkeit der Gleichungen gleichförmiger Gleichungssysteme) *Sei*

$$E := \begin{cases} u_n & \overset{\sigma}{=} & \varphi_n(u_1, \ldots, u_n) \\ & \vdots & \\ u_1 & \overset{\sigma}{=} & \varphi_1(u_1, \ldots, u_n) \end{cases}$$

ein Gleichungssystem mit Alternierungstiefe 1, dessen Abhängigkeitsgraph aus einer maximalen streng zusammenhängenden Komponente besteht und in dem $\sigma \in \{\mu, \nu\}$. Dann können die Gleichungen von E beliebig miteinander vertauscht werden, ohne dessen Lösung zu beeinflussen.

Beweis: Laut Theorem B.7 kann das gegebene Gleichungssystem mit Hilfe der Formeln B.5 berechnet werden. Diese verlangen, dass mit der Gleichung für u_1^{i+1} begonnen wird und die Gleichungen in aufsteigender Reihenfolge berechnet werden, weil die Gleichung für u_1^{i+1} die einzige ist, die nur von den Werten abhängt, die in Schritt i berechnet wurden. Die Gleichung für u_k^{i+1} benötigt alle davor berechneten u_g^{i+1}, $g < k$. In ähnlicher Weise lassen sich für jede Permutation $\pi(k)$ der Gleichungen Formeln angeben, so dass die Gleichungen in der Reihenfolge $u_{\pi(1)}^{i+1}, u_{\pi(2)}^{i+1}, \ldots, u_{\pi(n)}^{i+1}$ berechnet werden müssen.

Es lässt sich dann zeigen, dass die so gewonnenen Formeln den gleichen Fixpunkt berechnen wie B.1. Die Vorgehensweise ist die gleiche wie in dem Beweis von Theorem B.7, mit dem kleinen Unterschied, dass nun nicht mehr für alle Gleichungen gilt, dass $v_k^i \sqsubseteq u_k^{i,\ldots,i}$, sondern $v_k^i \sqsubseteq u_k^{i+1,\ldots,i+1}$, was aber nach wie vor zu dem gewünschten Ergebnis $v_k^\infty = u_k^{\infty,\ldots,\infty}$führt.

Die Formeln, die durch die Permutation π entstehen, sind aber genau die, die entsteht, wenn im Gleichungssystem die Gleichungen nach π vertauscht werden. Folglich wird das Ergebnis durch eine Vertauschung nicht beeinflusst. □

Literaturverzeichnis

[1] A. Arnold and P. Crubille. A linear algorithm to solve fixed-point equations on transition systems. *Information Processing Letters*, 29(2):57–66, 1988.

[2] A. Arnold, A. Vincent, and I. Walukiewicz. Games and synthesis of controllers with partial observation. *Theoretical Computer Science*, 303(1):7–34, 2003.

[3] S. Balemi, G. J. Hoffmann, P. Gyugyi, H. Wong-Toi, and G. F. Franklin. Supervisory control of a rapid thermal multiprocessor. *IEEE Transactions on Automatic Control*, 38(7):1040–1059, Jul. 1993.

[4] C. Berthet, O. Coudert, and J.C. Madre. New ideas on symbolic manipulations of finite state machines. In *International Conference on Computer Design (ICCD)*, pages 224–227. IEEE Computer Society, 1990.

[5] Jörg Bewersdorff. *Glück, Logik und Bluff*. Vieweg, Wiesbaden, 3rd. edition, 2003. ISBN 3-528-26997-9.

[6] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Tools and Algorithms for the Analysis and Construction of Systems (TACAS) – LNCS 1579*. Springer Verlag, 1999.

[7] G.V. Bochmann. Hardware specification with temporal logic: An example. *IEEE Transactions on Computers*, C-31(3):223–231, March 1982.

[8] Charles L. Bouton. Nim, a game with a complete mathematical theory. *Annals of Mathematics*, 3(2):35–39, 1902.

[9] B.A. Brandin. The real–time supervisory control of an experimental manufacturing cell. *IEEE Transactions on Robotics and Automation*, 12(1):1–14, 1996.

[10] A. Browne, E.M. Clarke, S. Jha, D.E. Long, and W.R. Marrero. An improved algorithm for the evaluation of fixpoint expressions. *Theoretical Computer Science*, 178(1–2):237–255, 1997.

[11] R.E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[12] J.R. Burch, E.M. Clarke, K.L. McMillan, and D.L. Dill. Sequential circuit verification using symbolic model checking. In *Design Automation Conference (DAC)*, pages 46–51, Los Alamitos, CA, June 1990. ACM.

[13] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Symposium on Logic in Computer Science (LICS)*, pages 1–33, Washington, D.C., June 1990. IEEE Computer Society.

[14] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, June 1992.

[15] R.M. Burstall. Program proving considered as hand simulation plus induction. In *Congress on Information Processing*, pages 308–312, 1974.

[16] C.G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, Boston, U.S.A., 1999. ISBN 0-7923-8609-4.

[17] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In D. Kozen, editor, *Workshop on Logics of Programs*, volume 131 of *LNCS*, pages 52–71, Yorktown Heights, New York, May 1981. Springer.

[18] 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 (TOPLAS)*, 8(2):244–263, April 1986.

[19] E.M. Clarke, O. Grumberg, and D.E. Long. Verification tools for finite state concurrent systems. In J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *A Decade of Concurrency – Reflections and Perspectives*, volume 803 of *LNCS*, pages 124–175, Noordwijkerhout, Netherlands, June 1993. REX School/Symposium, Springer.

[20] E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. MIT, London, England, 1999.

[21] R. Cleaveland, M. Klein, and B. Steffen. Faster model checking for the modal μ-calculus. In G.v. Bochmann and D. Probst, editors, *Conference on Computer Aided Verification (CAV)*, volume 663 of *LNCS*, pages 410–422, Montreal, June/July 1992. Springer.

[22] R. Cleaveland and B. Steffen. A linear-time model checking algorithm for the alternation-free μ-calculus. In K.G. Larsen and A. Skou, editors, *Conference on Computer Aided Verification (CAV)*, volume 575 of *LNCS*, pages 48–58, Aalborg, Denmark, July 1991. Springer.

[23] M. Dam. Temporal logic, automata, and classical theories - an introduction. Notes for the Sixth Summer School in Logic, Language, and Information, 1994.

[24] L. de Alfaro, T.A. Henzinger, and F.Y.C. Mang. The control of synchronous systems, part ii. In *Conference on Concurrency Theory (CONCUR)*, volume 2154 of *LNCS*, pages 566–582, Aalborg, Denmark, 2001. Springer.

[25] L. de Alfaro, Th. Henzinger, and R. Majumdar. From verification to control: Dynamic programs for omega-regular objectives. In *Proceedings of the 16th Annual Symposium on Logic in Computer Science*, pages 279–290. IEEE Computer Society Press, 2001.

[26] A. Dicky. An algebraic and algorithmic method of analyzing transition systems. *Theoretical Computer Science*, 46:285–303, 1986.

[27] E.A. Emerson and E.M. Clarke. Characterizing correctness properties of parallel programs as fixpoints. In *Colloquium on Automata, Languages and Programming (ICALP)*, volume 85 of *LNCS*, pages 169–181, Berlin, 1980. Springer.

[28] E.A. Emerson and J.Y. Halpern. ”sometimes” and ”not never” revisited: On branching versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, January 1986.

[29] E.A. Emerson and C.S. Jutla. The complexity of tree automata and logics of programs. In *Symposium on Foundations of Computer Science (FOCS)*, pages 328–337, White Plains, New York, 1988.

[30] E.A. Emerson and C.S. Jutla. The complexity of tree automata and logics of programs. *SIAM Journal on Computing*, 29(1):132–158, 1999.

[31] E.A. Emerson, C.S. Jutla, and A.P. Sistla. On model checking for fragments of μ-calculus. In C. Courcoubetis, editor, *Conference on Computer Aided Verification (CAV)*, volume 697 of *LNCS*, pages 385–396, Stanford, California, USA, June/July 1993. Springer.

[32] E.A. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional μ-calculus. In *Symposium on Logic in Computer Science (LICS)*, pages 267–278, Washington, D.C., 1986. IEEE Computer Society.

[33] E.A. Emerson and C.-L. Lei. Temporal reasoning under generalized fairness constraints. In B. Monien and G. Vidal-Naquet, editors, *Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 210 of *LNCS*, pages 21–36, Orsay, France, January 1986. Springer.

[34] E.A. Emerson and C.-L. Lei. Modalities for model checking: Branching time strikes back. *Science of Computer Programming*, 8:275–306, 1987.

[35] R.W. Floyd. Assigning meaning to programs. In *Symposia in Applied Mathematics: Mathematical Aspects of Computer Science*, volume 19, pages 19–31, 1967.

[36] S. Gottwald, H. Küstner, M. Hellwich, and H. Kästner, editors. *Handbuch der Mathematik*. Buch und Zeit Verlagsgesellschaft mbH, Köln, 1986. ISBN 3-7042-6019-3.

[37] E. Grädel, W. Thomas, and T. Wilke. *Automata, Logics, and Infinite Games*, volume 2500 of *LNCS*. Springer, 2002.

[38] K. Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme. *Monatshefte für Mathematik und Physik*, 38:173–198, 1931.

[39] C.A.R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12:576–580, 1969.

[40] J.E. Hopcroft, R. Motwani, and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, U.S.A., 2 edition, 2001. ISBN 0-201-44124-1.

[41] G.E. Hughes and M.J. Cresswell. *An Introduction to Modal Logic*. Methuen, London, 1968.

[42] G. Ifrah. *The Universal History of Computing – From the Abacus to the Quantum Computer*. John Wiley & Sons, Inc., New York, USA, 2001. ISBN 0-471-39671-0.

[43] S. Jiang and R. Kumar. Supervisory control of discrete event systems with CTL* temporal logic specifications. In *Proceedings of the of the 4Oth IEEE Conference on Decision and Control*, Orlando, FL, December 2001.

[44] M. Jurdziński. Deciding the winner in parity games is in UP $\cap$ co-UP. *Information Processing Letters*, 68(3):119–124, 1998.

[45] D. Kirsten. Alternating Tree Automata and Parity Games. In GrTW02 [37], pages 153–167.

[46] B. Kolman, R.C. Busby, and S. Ross. *Discrete Mathematical Structures*. Prentice Hall International, New Jersey, NY, 3rd. edition, 1996. ISBN 0-13-518549-1.

[47] D. Kozen. Results on the propositional μ-calculus. In *Colloquium on Automata, Languages and Programming (ICALP)*, pages 348–359, 1982.

[48] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, December 1983.

[49] S.A. Kripke. Semantical considerations on modal logic. In *Colloquium on Modal and Many Valued Logics*, volume 16 of *Acta Philosophica Fennica*, pages 83–94, August 1963.

[50] F. Kröger. Lar: A logic of algorithmic reasoning. *Acta Informatica*, 8:243–266, 1977.

[51] R. Kumar and V. Garg. *Modeling and Control of Logical Discrete Event Systems*. Kluwer Academic Publishers, 1995. ISBN 0-7923-9538-7.

[52] L. Lamport. ”sometime” is sometimes ”not never”-on the temporal logic of programs. In *Symposium on Principles of Programming Languages (POPL)*, pages 174–185, New York, 1980. ACM.

[53] M. Lawford, J.S. Ostroff, and W.M. Wonham. Model reduction of modules for state-event temporal logics. In R. Gotzhein and J. Bredereke, editors, *Formal Description Techniques IX: Theory, Application and Tools, Proceedings of FORTE/PSTV'96*, pages 263–278. Chapman & Hall, 1996.

[54] C.Y. Lee. Representation of switching circuits by binary decision diagrams. *Bell Systems Technical Journal*, 38:985–999, 1959.

[55] Y. Li and W.M. Wonham. Concurrent vector discrete-event systems. *IEEE Trans. on Automatic Control*, 40(4):628–638, 1995.

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

[57] F. Lin and W.M. Wonham. On observability of discrete-event systems. *Information Sciences*, 44(3):173–198, 1988.

[58] D.E. Long, A. Browne, E.M. Clarke, S. Jha, and W.R. Marrero. An improved algorithm for the evaluation of fixpoint expressions. In D.L. Dill, editor, *Conference on Computer Aided Verification (CAV)*, volume 818 of *LNCS*, pages 338–350, Stanford, California, USA, June 1994. Springer.

[59] Y. Malachi and S.S. Owicki. Temporal specifications of self-timed systems. In H.T. Kung, B. Spoull, and G. Steele, editors, *VLSI Systems and Computations*, pages 203–212, Rockville, MD, 1981. IEEE Computer Society.

[60] P. Malik. *From Supervisory Control to Nonblocking Controllers for Discrete Event Systems*. PhD thesis, Universität Kaiserslautern, 2003.

[61] Z. Manna and A. Pnueli. The anchored version of the temporal framework. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, volume 354 of *LNCS*, pages 428–437, Noordwigherhout, Netherland, May/June 1988. Springer.

[62] Z. Manna and A. Pnueli. A hierarchy of temporal properties. In *Symposium on Principles of Distributed Computing*, pages 377–408, 1990.

[63] H. Matis. *Die Wundermaschine – Die unendliche Geschichte der Datenverarbeitung: Von der Rechenuhr zum Internet*. Wirtschaftsverlag Carl Ueberretuer, Wien, Austria, 2002. ISBN 3-8266-0953-0.

[64] G.J. Myers. *Methodisches Testen von Programmen*. Oldenbourg Wissenschaftsverlag GmbH, München, 2001. ISBN 3-486-25634-3.

[65] D. Niwiński. On fixed point clones. In L. Kott, editor, *Colloquium on Automata, Languages and Programming (ICALP)*, volume 226 of *LNCS*, pages 464–473. Springer, 1986.

[66] A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*, pages 46–57, Providence, Rhode Island, Nov. 1977. IEEE.

[67] V. Pratt. A decidable μ-calculus. In *Symposium on Foundations of Computer Science (FOCS)*, pages 421–427, New York, 1981. IEEE Computer Society.

[68] J.P. Quielle and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Symposium in Programming*, 1981.

[69] J.P. Quielle and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Symposium in Programming*, volume 137 of *LNCS*, pages 337–351, New York, 1982. Springer.

[70] P. J. Ramadge and W. M. Wonham. Modular feedback logic for discrete event processes. *SIAM Journal of Control and Optimization*, 25(5):1202–1218, 1987.

[71] P. J. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, 1989.

[72] P.J. Ramadge and W.M. Wonham. Supervisory control of a class of discrete event processes. *SIAM Journal of Control and Optimization*, 25(1):206–230, 1987.

[73] K. Rudie and W. M. Wonham. Think globally, act locally: Decentralized supervisory control. *IEEE Transactions on Automatic Control*, 37(11):1692–1708, 1992.

[74] S. Safra. On the complexity of ω-automata. In *Symposium on Foundations of Computer Science (FOCS)*, pages 319–327, 1988.

[75] K. Schneider. *Verification of Reactive Systems – Formal Methods and Algorithms*. Texts in Theoretical Computer Science (EATCS Series). Springer, 2003. ISBN 3-540-00296-0.

[76] H. Seidl. Fast and simple nested fixpoints. *Information Processing Letters*, 59(6):303–308, 1996.

[77] M. Seidl, T. Kleinert, J. Wagner, and B. Plannerer. Systematischer Steuerungsentwurf zur Kontrolle lastintensiver Call-Center. *at - Automatisierungstechnik*, 50:19–27, June 2002.

[78] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific J. Math*, 5:285–309, 1955.

[79] G.E. Thaller. *Software-Test – Verifikation und Validation*. Verlag Heinz Heise, 2002. ISBN 3-88229-198-2.

[80] J. G. Thistle and W. M. Wonham. Supervision of infinite behavior of discrete–event systems. *SIAM Journal of Control and Optimization*, 32(4):1098–1113, 1994.

[81] J.G. Thistle and W.M. Wonham. Supervision of infinite behavior of discrete event systems. *SIAM Journal of Control and Optimization*, 32(4):1098–1113, 1994.

[82] M.Y. Vardi. Branching vs. linear time: Final showdown. In *Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 2031 of *LNCS*, pages 1–22, Genova, Italy, 2001. Springer.

[83] T. Wilke. Alternating tree automata, parity games, and modal μ-calculus. *Bulletin of the Société Mathématique Belgique*, 8(2), 2001.

[84] K.C. Wong. *Discrete-Event Control Architecture: An Algebraic Approach*. Phd thesis, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Canada, 1994. http://www.control.utoronto.ca/theses/theses.html.

[85] W. M. Wonham. Supervisory control of discrete-event systems. Technical report, Dept. of Electrical and Computer Engineering, University of Toronto, Jul. 2005. ECE 1636F/1637S 2005-06, http://www.control.utoronto.ca/DES.

[86] W.M. Wonham and P.J. Ramadge. On the supremal controllable sublanguage of a given language. *SIAM Journal of Control and Optimization*, 25(3):637–659, May 1987.

[87] W.M. Wonham and P.J. Ramadge. Modular supervisory control of discrete event systems. *Mathematics of control of discrete event systems*, 1(1):13–30, 1988.

[88] Z.H. Zhang. *Smart TCT: An Efficient Algorithm for Supervisory Control Design*. M.a.sc. thesis, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Canada, April 2001.

[89] H. Zhong and W.M. Wonham. On the consistency of hierarchical supervision in discrete-event systems. *IEEE Transactions on Automatic Control*, 35(10):1125–1134, 1990.

[90] R. Ziller. System modelling using marker states in the RW-framework. In *GI/ITG/GMM-Workshop: Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen*, Meißen, Germany, 19-21 February 2001.

[91] R. Ziller. Finding bad states during symbolic supervisor synthesis. In *GI/ITG/GMM Workshop: Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen*, pages 209–218, Tübingen, Germany, 25-27 February 2002.

[92] R. Ziller and K. Schneider. A generalized approach to supervisor synthesis. In *Formal Methods and Models for Codesign (MEMOCODE)*, pages 217–226, Mont Saint-Michel, France, 2003. IEEE Computer Society.

[93] R. Ziller and K. Schneider. A μ-calculus approach to supervisor synthesis. In R. Drechsler, editor, *GI/ITG/GMM Workshop: Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen*, pages 132–143, Bremen, Germany, 24-26 February 2003. Shaker.

[94] R. Ziller and K. Schneider. Reducing Complexity of Supervisor Synthesis. In *IFAC Conference on Control Systems Design (CSD)*, pages 183–192, Bratislava, Slovak Republic, 2003. Elsevier.

[95] R. Ziller and K. Schneider. Combining Supervisor Synthesis and Model Checking. *ACM Transactions on Embedded Computing Systems*, 4(2):331–362, May 2005.

Index