Formale Modellierung von Zielen und zielorientiertem Verhalten in Agentensystemen mit Ziel/Transitions-Netzen

von

Dennis Chong

Dissertation

zur Erlangung des Grades eines
Doktors der Ingenieurswissenschaften
- Dr.-Ing. -

Vorgelegt im Fachbereich 3 (Mathematik & Informatik)
der Universität Bremen
im Februar 2011

Datum des Promotionskolloquiums: 20. Mai 2011

Gutachter:

Prof. Dr. Hans-Jörg Kreowski (Universität Bremen)

Prof. Dr. Sofie Czernik (Hochschule Bremerhaven)

Abstract

Für die Modellierung von Agenten existieren verschiedene Ansätze mit unterschiedlichen Vor- und Nachteilen. Insbesondere die Modellierung von Zielen und die Darstellung von zielorientiertem Verhalten stellt eine Herausforderung dar, weil Ziele als mentale Zustände eines Agenten zur Laufzeit Änderungen unterliegen können und deshalb als dynamische Elemente auf der Modellebene abzubilden sind. Für die Modellierung von Agentensystemen mit dynamischen Zielen wird in dieser Arbeit mit Ziel/Transitions-Netzen ein Petri-Netz-orientierter Ansatz vorgeschlagen. Der Formalismus basiert auf Prädikat/Transitions-Netzen, einer verbreiteten höheren Petri-Netz-Art. Für die Darstellung von Systemabläufen in Ziel/Transitions-Netzen werden Protokollnetze verwendet. Mit Protokollnetzen ist es möglich, die Ziele eines Agenten und die Zustände des Systems in einem Zusammenhang zu erfassen und ihre Wechselwirkung zu untersuchen. Auf diese Weise lässt sich der Begriff der Zielorientierung von Systemabläufen formal erfassen.

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Motivation 1

1.2 Ziele und Aufbau der Arbeit 3

2 Modellierung von Agentensystemen **5**

2.1 Der Agentenbegriff 5

2.2 Agentencharakteristika 9

2.2.1 Autonomie 9

2.2.2 Reaktivität 10

2.2.3 Proaktivität 11

2.2.4 Sozialität 12

2.3 Agentenmodelle 14

2.3.1 UML/AUML 14

2.3.2 Automaten 20

2.3.3 Zelluläre Automaten 21

2.3.4 Algebraische Spezifikation 24

2.3.5 Modal- und Temporallogik 26

2.3.6 Graphtransformationen 28

2.3.7 Petri-Netze 31

3 Modellierung von Zielen **41**

3.1 Grundidee 42

3.2 Algebraische Grundlagen 46

3.3 Ziel/Transitions-Netze 51

3.4 Grundbegriffe 56

3.4.1 Erreichbarkeit 57

3.4.2 Nebenläufigkeit 58

3.4.3 Konflikt 59

3.4.4 Beschränktheit 61
3.4.5 Lebendigkeit 62
3.5 Mächtigkeit 63
3.6 Modifikationen 66
3.6.1 Indizierte Prädikate 66
3.6.2 Kapazitäten 67
3.6.3 Vollständige Bindung 70
3.7 Anwendungsbeispiel 71

4 Modellierung von zielorientiertem Verhalten 75
4.1 Grundidee 75
4.2 Protokollnetze 78
4.3 Grundeigenschaften 84
4.4 Linien 87
4.5 Schnitte 91
4.6 Zielorientierte Abläufe 95
4.7 Anwendungsbeispiel 99

5 Schlussbetrachtung 105
5.1 Zusammenfassung 105
5.2 Ausblick 107

Literaturverzeichnis 109

1

Einleitung

1.1 Motivation

Die Disziplin der Softwaretechnik hat das Ziel, mit geeigneten Methoden und Werkzeugen die Komplexität von Software zu verstehen und zu beherrschen [Pre04, Bal96]. Eine zentrale Aufgabe besteht dabei in der Entwicklung von geeigneten Sprachmitteln zur Modellierung von Software-Systemen. Modelle bilden die Struktur und das Verhalten von Software-Systemen nach. Sie unterstützen den Software-Entwicklungsprozess, indem sie von unwesentlichen Details abstrahieren und die relevanten Aspekte in den Vordergrund stellen. Außerdem besitzen sie häufig eine graphische Repräsentation, um den Vorteil der visuellen Wahrnehmung auszunutzen. So lassen sich komplizierte Zusammenhänge übersichtlicher, anschaulicher und damit verständlicher darstellen. Bereits in den 70er Jahren wurden mit *strukturierten Methoden* [YC79] die Vorteile graphischer Modelle für die Entwicklung von Software erkannt. Als entsprechende Ansätze wurden u.a. Entity-Relationship-Diagramme und Flussdiagramme vorgeschlagen. Die Nachteile bestanden vor allem in der mangelnden Darstellung der Beziehung zwischen Struktur und Verhalten eines Systems. Als Lösung wurde in den 80er Jahren das Konzept der abstrakten Datentypen propagiert, das sich schließlich als Objektorientierung in der Softwaretechnik durchsetzen konnte. Die objektorientierten Sprachmittel wie z.B. die Unified Modeling Language (UML) sind jedoch nicht gleichermaßen für die Modellierung von Agentensystemen geeignet, weil für die Darstellung von Agenten andere Aspekte in den Vordergrund gestellt werden müssen.

Die Agentenorientierung ist eine Reaktion auf den zunehmenden Trend von autonomen Software-Systemen mit verteilten Daten und Lösungen. Sie unterstützt die Kernpunkte der Komplexitätsbewältigung: Modularisierung, Abstraktion und Strukturierung [Boo93, GJM91]. Agenten repräsentieren in sich geschlossene Software-Entitäten, die über einen längeren Zeitraum in einer Umgebung persistent sind und die Fähigkeit zu autonomen Aktionen, Reaktionen und Interaktionen haben. Das Verhalten eines Agenten ist dabei an Ziele gekoppelt, die die Beschaffenheit der Umgebung beschreiben, die er mit seinen Handlungen zu erreichen versucht. Ziele stellen mentale Zustände eines Agenten dar und können zur Laufzeit Änderungen unterliegen (dynamische Ziele). Eine besondere Herausforderung bei der Modellierung von Agentensystemen besteht in der geeigneten Repräsentation von dynamischen Zielen.

Ansätze zur Modellierung von Agentensystemen werden seit Jahren unter dem Begriff *Agent Oriented Software Engineering* erforscht. Eine Reflektion über die unterschiedlichen Forschungsrichtungen bieten [CW01, WWC02, GOW03, GMO03, OGM04, MZ06, PZ07, LP08, LGS09]. Ein Beispiel stellt die *Agent UML* [BMO00] dar, die als Weiterentwicklung der Unified Modeling Language (UML) die besonderen Aspekte von Agentensystemen bei der Modellierung berücksichtigen soll. Modellierungssprachen wie die UML oder die Agent UML besitzen allerdings keine formale Semantik. Sie können nicht dazu verwendet werden, formale Aussagen über das Modellverhalten zu machen, weil die Systemabläufe nicht exakt durch die Modelle festgelegt werden. Dies ist bei die Modellierung von Agentensystemen inakzeptabel, wenn die Agenten u.A. sicherheitskritische Systeme darstellen, wie z.B. in der Verkehrsüberwachung [Ram98] oder in der Weltraumforschung [PGK+97]. Durch den Einsatz von formalen Modellierungssprachen ist es möglich, das Verhalten eines Systems vollständig zu beschreiben und Mehrdeutigkeiten und Inkonsistenzen zu vermeiden [Win90]. Auf diese Weise können alle relevanten Systemabläufe einer formalen Überprüfung unterzogen werden.

> "Every program performs some task correctly. What is of interest to computer scientists is whether a program performs its intended task." [LB79]

Obwohl formale Methoden die Software-Qualität entscheidend unterstützen können, haben sie sich in der Softwaretechnik nicht vollständig durchsetzen können. In vielen Branchen ist nicht die Qualität von Software, sondern die Dauer bis zu ihrer Auslieferung entscheidend [Sum07]. Mit zunehmender Verbreitung der Agentenorientie-

rung wird formalen Methoden jedoch eine immer größere Bedeutung zugesprochen, während gleichzeitig die Verfügbarkeit formaler Methoden große Bedeutung für die Akzeptanz der Agentenorientierung hat [WJ04]. Aus diesem Grund stellt die Suche nach einem formalen Ansatz zur Modellierung von Agentensystemen den Ausgangspunkt für diese Arbeit dar. Der gesuchte Formalismus sollte über eine graphische Repräsentation verfügen und gleichzeitig die besonderen Aspekte von Agenten, insbesondere die Repräsentation von dynamischen Zielen, berücksichtigen. Idealerweise besteht die Aufgabe dabei weniger darin, einen völlig neuen Formalismus zu entwerfen, sondern geeignete Ansätze zu identifizieren und bei Bedarf zu erweitern [DP04]. Die Auswahl eines geeigneten Formalismus für ein Problem kann jedoch selbst ein nichttriviales Problem sein [SRG99].

1.2 Ziele und Aufbau der Arbeit

Im Mittelpunkt dieser Arbeit steht die Frage, wie Ziele und zielorientiertes Verhalten von Agenten formal erfasst werden können. Zu diesem Zweck werden die in der Literatur verbreiteten Ansätze zur Modellierung von Agentensystemen untersucht, um die Stärken und die Schwächen der einzelnen Ansätze im Allgemeinen und im Bezug auf die Agentenorientierung herauszuarbeiten. Aufbauend auf diesen Ergebnissen wird gezeigt, dass insbesondere Petri-Netze einen geeigneten Ausgangspunkt für die Modellierung von Agentensystemen darstellen. Anschließend wird der Formalismus der Petri-Netze erweitert, um die Modellierung von dynamischen Zielen eines Agenten zu ermöglichen. Daraufhin wird eine entsprechende Ablaufdarstellung entwickelt, so dass Systemabläufe angemessen dargestellt und in Bezug auf ihre Zielorientierung untersucht werden können. Mit diesen Erkenntnissen soll ein Beitrag zur agentenorientierten Software-Entwicklung geleistet werden. Zu diesem Zweck wird der Rest dieser Arbeit wie folgt gegliedert:

- **Modellierung von Agentensystemen**
  In Kapitel 2 werden die grundlegenden Ideen und Begriffe des Agentenkonzepts dargestellt. Im Mittelpunkt steht dabei die Frage, was Agentensysteme von anderen Software-Systemen unterscheidet und welche Eigenschaften im Bezug auf die Agentenorientierung unbedingt von einem geeigneten Ansatz zur Modellierung zu berücksichtigen sind. Anschließend werden die in der Literatur verbreiteten Ansätze auf der Grundlage dieser Ergebnisse untersucht. Der Formalismus der Petri-Netze bildet dabei einen besonderen Schwerpunkt.

- **Modellierung von Zielen**
  In Kapitel 3 wird mit der Einführung von Ziel/Transitions-Netzen der Formalismus der Petri-Netze erweitert, um dynamische Ziele eines Agenten modellieren zu können. Ausgangspunkt stellen Prädikat/Transitions-Netze dar, in denen bereits ein Zusammenhang zwischen den Zuständen eines Systems und logischen Formeln mit *variablen Prädikaten* erklärt ist. Dieser Zusammenhang wird in Ziel/Transitions-Netzen ausgenutzt, indem Formeln mit variablen Prädikaten als Modellbestandteile eingeführt werden. Diese Erweiterung ermöglicht es, Zustände des Systems auf der Modellebene zu beschreiben und damit dynamische Ziele eines Agenten zu repräsentieren. Anschließend werden die wichtigsten Grundbegriffe der Petri-Netz-Theorie auf Ziel/Transitions-Netze übertragen. Außerdem werden einige Aspekte zur Mächtigkeit diskutiert und Modifikationen vorgestellt. Das Kapitel wird mit einem Anwendungsbeispiel abgeschlossen, das den Einsatz von Ziel/Transitions-Netzen bei der Modellierung eines einfachen Agenten mit dynamischen Zielen demonstrieren soll.

- **Modellierung von zielorientiertem Verhalten**
  In Kapitel 4 wird eine Ablaufsdarstellung für Ziel/Transitions-Netze entwickelt, um Systemabläufe angemessen darstellen und untersuchen zu können. Ausgangspunkt stellen Protokollnetze dar, die eine kompakte und anschauliche Darstellung der Vorgänge in einem Petri-Netz sowie ihren Vor- und Nachbedingungen ermöglichen. Auf der Grundlage der entwickelten Ablaufdarstellung wird ein Zusammenhang zwischen den Zielen eines Agenten und den Zuständen des Systems hergestellt, um den Begriff der Zielorientierung formal zu erfassen. An einem Anwendungsbeispiel wird gezeigt, wie zielorientiertes Verhalten in ausgewählten Systemabläufen eines einfachen Agenten überprüft werden kann.

In Kapitel 5 werden die Ergebnisse dieser Arbeit zusammengefasst und weitere mögliche Arbeiten zu diesem Thema identifiziert.

2

Modellierung von Agentensystemen

Agentenorientierte Ansätze wurden in den vergangenen Jahren in zunehmenden Maße attraktiv für die Softwaretechnik [KG97]. Einen Überblick über verschiedene Anwendungen des Agentenkonzepts liefern z.B. [Klü04], [Par99], [JSW98] und [JW98]. Zudem haben Standardisierungsbemühungen der FIPA [Age11a] für eine immer größere Verbreitung in Forschung und Industrie gesorgt. Gleichzeitig wächst der Bedarf nach geeigneten Ansätzen zur Modellierung von Agentensystemen. Um einen solchen Ansatz zu identifizieren, ist es notwendig, die grundlegenden Ideen und Begriffe der Agentenorientierung herauszuarbeiten. Es stellt sich die Frage, wie sich der Begriff *Agent* präzise erfassen lässt und welche Aspekte im besonderen Maße bei der Modellierung von Agentensystemen zu berücksichtigen sind. Anschließend können die in der Literatur verbreiten Ansätze auf der Grundlage dieser Ergebnisse untersucht werden.

2.1 Der Agentenbegriff

Der Agentenbegriff etablierte sich Mitte der 90er in der (Verteilten) Künstlichen Intelligenz. Ursprünglich wurde er für Software-Systeme verwendet, die selbständig nach Lösungen in komplexen Problemstellungen suchen sollten. Inzwischen ist der Begriff in Form eines neuen Paradigmas in der Disziplin der Softwaretechnik aufgegangen, um den modernen Anforderungen autonomer, verteilter und offener Software-Systeme angemessen zu begegnen.

> "We need to recognize that intelligent agents are ninety-nine percent computer science and one percent AI."
> [Etz97, S. 2].

Der Agentenbegriff war und ist Gegenstand vieler Präzisierungsbemühungen. Eine einheitliche und allgemein anerkannte Definition ist in der Literatur allerdings nicht zu finden. Neben zahlreichen unterschiedlichen Formulierungen existiert ebenso eine Vielzahl von Versuchen, den Agentenbegriff mit alternativen Konzepten darzustellen [HS99a, MPHG02] oder die bisherigen Definitionen zu klassifizieren und in eine umfassendere Definition zusammenzuführen [Pet96, Cas97, FG97, Pet97]. Eine Auswahl der am häufigsten zitierten Ansätze ist wie folgt gegeben:

> "An autonomous agent is a system situated within and a part of an environment that senses that evironment and acts on it, over time, in pursiut of its own agenda and so as to effect what it senses in the future." [FG97, S. 24]

> "An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives." [Woo02, S. 15]

> "Autonomous agents are computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals or tasks for which they are designed." [Mae95, S. 108]

Die meisten in der Literatur verbreiteten Formulierungen liegen trotz unterschiedlicher Sichtweisen der selben Idee zu Grunde: Agenten sind in sich geschlossene Software-Entitäten, die den Bedingungen einer Umgebung über einen längeren Zeitraum ausgesetzt sind. Über Sensoren, gleich welcher Beschaffenheit, nehmen sie ihre Umgebung wahr. Eine Effektorik ermöglicht es ihnen, mit geeigneten Aktionen, Reaktionen und Interaktionen ihre Umgebung zu beeinflussen, um so nach Vorgabe bestimmter Ziele selbstständig operieren zu können. Schematisch wird ein Agent dabei häufig ähnlich wie in Abbildung 2.1 dargestellt.

Wooldridge und Jennings präzisieren den Agentenbegriff, indem sie eine Reihe von Merkmalen formulieren, die einen Agenten charakterisieren [WJ95, Woo98, WC00]. Dieser Ansatz hat eine breite Akzeptanz in der Literatur gefunden und kommt einer einheitlichen und allgemein anerkannten Charakterisierung des Agentenbegriffs

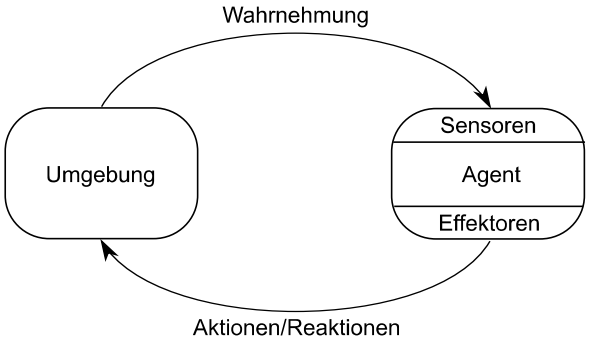


Abbildung 2.1: Schematische Darstellung eines Agenten.

in der Softwaretechnik am nächsten. Er identifiziert vier zentrale Merkmale eines Agenten:

- *Autonomie:*
  Agenten können ohne direkten Eingriff des Menschen operieren und kontrollieren selbstständig ihr Verhalten und ihren inneren Zustand.
- *Reaktivität:*
  Agenten können ihre Umgebung wahrnehmen und reagieren in angemessener Zeit und in angemessener Weise auf Änderungen, um ihre Ziele zu erfüllen.
- *Proaktivität:*
  Agenten können die Initiative ergreifen und spontane Handlungen initiieren, um ihre Ziele zu erfüllen.
- *Sozialität:*
  Agenten können mit anderen Agenten (und Menschen) interagieren, um ihre Ziele zu erfüllen.

Die Charakterisierung des Agentenbegriffs nach Wooldridge und Jennings bildet für die meisten Forscher eine ausreichende Arbeitsgrundlage. Sie betrifft primär funktionale Eigenschaften und macht keine Vorgaben über den inneren Aufbau eines Agenten, d.h. die Konzeption und die Realisierung eines Agenten unterliegt keinen strukturellen Einschränkungen. Da der Ansatz die Grundideen der meisten Präzisierungsbemühungen in der Literatur reflektiert, stellt er zudem einen geeigneten Ausgangspunkt dar, um die verschiedenen Ansätze zur Modellierung von Agentensystemen zu vergleichen.

Neben einer Charakterisierung des Agentenbegriffs wird in der Literatur auch häufig eine architektonische Differenzierung vorgenommen. Agentenarchitekturen legen fest, aus welchen strukturellen Komponenten ein Agent zusammengesetzt ist und wie das Zusammenspiel dieser Komponenten funktioniert. Typischerweise werden dabei deliberative und reaktive Architekturen unterschieden [Fer01, WJ95, Woo02]:

- **Deliberative Architekturen:**
  Unter deliberativen Architekturen lassen sich alle Konzepte zusammenfassen, die eine explizite symbolische Repräsentation des mentalen Zustands eines Agenten vorsehen. Häufig werden in diesem Zusammenhang logische Sprachen verwendet. Die Gesamtheit der symbolischen Repräsentation wird als Wissensbasis bezeichnet. Sie wird dazu verwendet, um durch Schlussfolgerungen neues Wissen zu erzeugen und über die Aktionen und Reaktionen eines Agenten zu entscheiden. Für die Implementierung eines deliberativen Agenten muss zunächst eine geeignete symbolische Darstellung von Wissen und Verhalten gefunden werden. Dieses Problem gilt allerdings als noch nicht vollständig gelöst.

  > "Despite the immense volume of work that the problems have generated, many people would argue that neither problem is anywhere near solved." [Woo02, S. 48]

- **Reaktive Architekturen:**
  Reaktive Architekturen liegen nach Brooks der Idee zu Grunde, dass intelligentes Verhalten keine explizite symbolische Repräsentation und Manipulation benötigt und aus einfachen Verhaltensregeln emergieren kann [Bro91a, Bro91b]. Stattdessen verfügen reaktive Agenten über einfache Stimulus-Response-Regeln, die die Wahrnehmung des Agenten direkt auf eine oder mehrere Aktionen abbilden. Jede Regel repräsentiert einen speziellen Aspekt, der ein Bestandteil seines Verhaltens sein soll. Treffen neue Daten der Sensorik ein, wird anhand einer vorgegebenen Hierarchie (Subsumptionshierarchie) aus den anwendbaren Regeln eine Regel ausgewählt und ausgeführt. Arbeiten wie [Bro91a], [Bro91b], [Ste90b] und [CDFL92] demonstrieren, wie bereits aus einer kleinen Menge von einfachen Stimulus-Response-Regeln intelligentes Verhalten emergieren kann.

Der Vorteil von reaktiven Architekturen besteht in ihrer einfachen Implementierung. Da reaktive Agenten zudem sehr schnell auf Änderungen in der Umgebung reagieren können, sind sie gegenüber deliberativen Agenten häufig auch robuster

und effizienter [HWP01]. Die Nachteile resultieren unmittelbar aus der fehlenden Wissensrepräsentation. Da die Auswahl einer Aktion nur von der aktuellen Wahrnehmung abhängt, kann ein reaktiver Agent im Gegensatz zu einem deliberativen Agenten keine fortlaufenden Änderungen in der Umgebung, die in einem Zusammenhang stehen, identifizieren und frühere Entscheidungen nicht in sein Verhalten miteinbeziehen. Konzepte, die sowohl deliberative als auch reaktive Ansätze vereinen, werden als hybride Architekturen bezeichnet [Woo02].

2.2 Agentencharakteristika

2.2.1 Autonomie

Aus Sicht der Softwaretechnik ist Autonomie das einflussreichste Merkmal eines Agenten. Raumsonden wie Deep Space 1 können ohne Eingriff des Menschen den Flug durch das Sonnensystem steuern. Der Erkundungsroboter Sojourner (NASA Pathfinder-Missionen) auf dem Mars erhält nur einmal täglich Instruktionen von der Erde und operiert die restliche Zeit selbstständig, um seine vorgegebenen Ziele zu erfüllen. Die Autonomie eines Robocup-Agenten ist durch das Regelwerk des Spiels vorgegeben, das eine Kontrolle durch den Menschen im Spielverlauf weitestgehend untersagt. Ein Agent als Stellvertreter des Menschen bei Online-Auktionen macht unter Einhaltung bestimmter Restriktionen selbstständig Gebote, um den Zuschlag für die gewünschten Waren zu erhalten.

Autonomie bedeutet, dass Agenten selbstständig auf der Basis ihrer Wahrnehmung entscheiden, welche Aktionen sie auslösen, um ihre Ziele zu erfüllen. Es bedarf keinerlei Abstimmung mit Dritten, ob, wann und wie ein Agent seine Sensorik auf seine Effektorik abbildet. Ein Agent kann unabhängig von anderen Agenten aktiv werden und über die Art und den Zeitpunkt seiner Aktionen und Reaktionen frei entscheiden. Er operiert somit nebenläufig. Autonomie bedeutet auch, dass ein Agent seinen inneren Zustand und sein Verhalten selbständig kontrolliert. Dies setzt eine geeignete Form von Kapselung voraus, die die Interna eines Agenten gegenüber der Sensorik und Effektorik anderer Agenten und der Umgebung schützt. Dies ist auch der grundlegende Unterschied zum objektorientierten Ansatz.

> "There is a fundamental mismatch between the concepts used by object-oriented developers [...] and the agent-oriented view." [WJK00, S. 285]

Objekte haben keine Kontrolle über ihr Verhalten. Sie können zwar ihren inneren Zustand vor dem Zugriff von außen schützen, jedoch nicht ihr Verhalten. Ein Objekt kann nicht beeinflussen, wann und warum seine Methoden ausgeführt werden. Sobald eine Methode angestoßen wird, wird auch mit Sicherheit das in ihr gekapselte Verhalten ausgelöst. Objekte verhalten sich untereinander gehorsam und lassen sich im Gegensatz zu Agenten fremdkontrollieren [Tve01]. Agenten können dagegen selbst entscheiden, wann und warum sie welche Handlungen auslösen. Nur die Kommunikationsschnittstellen eines Agenten sind u.U. öffentlich zugänglich. Ein Agent entscheidet selbst, ob es in seinem Interesse ist, einer Aufforderung nachzukommen.

> "Objects do it for free, agents do it for money."
> [Woo99, S. 35]

> "Objects do it for free, agents because they want to do." [Woo02, S. 26]

Ein geeigneter Formalismus zur Modellierung von Autonomie in Agentensystemen sollte somit in der Lage sein, Nebenläufigkeit abzubilden und entsprechende Kapselungskonzepte zu unterstützen.

2.2.2 Reaktivität

Reaktivität bezeichnet die Fähigkeit eines Agenten, seine Umgebung wahrzunehmen, Änderungen zu erkennen und sein Verhalten entsprechend anzupassen. Mit Reaktivität ist also die Wechselwirkung zwischen Sensorik und Effektorik gemeint. Die Raumsonde Deep Space 1 reagiert auf einen fehlerhaften Kurs durch Aktivieren der entsprechenden Antriebsvorrichtungen zur Kurskorrektur. Der Mars-Erkundunsroboter Sojourner weicht aus, wenn er ein unpassierbares Hindernis unmittelbar voraus wahrnimmt. Ein Robocup-Agent nimmt ein Passspiel wahr und bewegt sich zur berechneten Ballposition. Ein Online-Auktions-Agent stellt fest, dass sein letztes Gebot überboten wurde und gibt vielleicht ein neues Gebot ab.

Im Gegensatz zu klassischen Software-Systemen kann ein Agent beliebig lange in einer Umgebung persistent sein. Er terminiert nicht nach dem klassischen Muster Eingabe-Berechnung-Ausgabe. Ein Agent handelt reaktiv, wenn er auf spontane Änderungen in der Umgebung vorbereitet ist und angemessen reagieren kann. Angemessen bedeutet, dass die Reaktion rechtzeitig erfolgt und mit den Zielen des Agenten übereinstimmt. Solche Änderungen können die Folge von Aktionen und Reaktionen anderer Agenten sein oder unberechenbaren Vorgängen der Umgebung zu Grun-

de liegen. Entweder ist Nichtdeterminismus tatsächlich ein Bestandteil des Systems oder die Umgebung erweist sich für den Agenten als nicht vollständig beobachtbar, was aus seiner Sicht ebenfalls einem Nichtdeterminismus entspricht. Nichtdeterministische Vorgänge sind somit die entscheidende Ursache für die Notwendigkeit von reaktivem Verhalten. Ein geeigneter Formalismus zur Modellierung von Reaktivität in Agentensystemen sollte daher in der Lage sein, Nichtdeterminismus abzubilden.

2.2.3 Proaktivität

Proaktivität bezeichnet die Fähigkeit eines Agenten, nicht nur auf Änderungen in der Umgebung zu reagieren, sondern auch spontane Handlungen initiieren zu können. Agenten sollen ihre Ziele kontinuierlich verfolgen, und nicht erst dann, wenn sie von außen dazu angestoßen werden. Deep Space 1 kann von sich aus eine komplette Systemdiagnose durchführen, um auf Fehler frühzeitig zu reagieren. Sojourner bewegt sich ohne direkten Befehl durch das Mars-Gelände, um sein Reiseziel zu erreichen. Ein Robocup-Agent bewegt sich ohne Aufforderungen in eine freie Position für einen guten Schuss ins Tor. Ein Online-Auktions-Agent gibt von sich aus ein Eröffnungsgebot für die Ware ab, die er zu ersteigern versucht.

Proaktives Verhalten ist an Ziele gekoppelt. Ziele stellen mentale Zustände eines Agenten dar. Sie beschreiben die Beschaffenheit der Umgebung, die der Agent mit seinen Handlungen zu erreichen versucht. Sie beeinflussen, welche Pläne, Strategien und Verpflichtungen er sich selbst oder Anderen gegenüber verfolgt [WJ04]. Ihre explizite Repräsentation auf der Modell-Ebene ist notwendig, um das Verhalten eines Agenten angemessen untersuchen zu können [WPHT02]. Ziele können darüber hinaus in statische und in dynamische Ziele unterschieden werden [KR91]. Dynamische Ziele können im Gegensatz zu statischen Zielen zur Laufzeit Änderungen unterliegen: Agenten sind in der Lage, auf Grundlage ihrer Erfahrungen selbstständig Ziele zu definieren, zu verwerfen und zu modifizieren. Zu einer Änderung kann es kommen, wenn ein Ziel erreicht wurde oder wenn es aus anderen Gründen nicht mehr sinnvoll erscheint, es in der ursprünglichen Form weiterzuverfolgen. Die Berücksichtigung von dynamischen Zielen ist ein wichtiger Aspekt bei der Modellierung von Agentensystemen. Ein geeigneter Formalismus zur Modellierung von Proaktivität in Agentensystemen unterstützt ihre explizite Repräsentation auf der Modell-Ebene.

2.2.4 Sozialität

> "There is no such thing as a single agent system."
> [Woo02, S. 105]

Sozialität umfasst alle Fähigkeiten eines Agenten, die der Koordination mit anderen Agenten dienen. Dies schließt auch die Interaktion mit dem Menschen ein. Auf der Raumsonde Deep Space 1 sind verschiedene Agenten für Planung, Steuerung und Diagnose zuständig, die sich gegenseitig austauschen müssen, um den reibungslosen Betrieb sicherzustellen. Sojourner kommuniziert einmal täglich mit der Erde, um neue Ziele zu empfangen, während ein Robocup-Agent sich u.U. mit einem seiner Teamkameraden über ein Passspiel abstimmen kann. Für einen Online-Auktions-Agenten ist Kommunikation eine notwendige Voraussetzung, um überhaupt mit anderen Agenten verhandeln zu können.

Koordinationsprobleme treten insbesondere dann auf, wenn sich eine Gemeinschaft von Agenten räumlich und zeitlich begrenzte Ressourcen teilen oder ihre Aktionen aufgrund von beschränkten Informationen und Fähigkeiten aufeinander abstimmen muss [Fer01, HS99b]. Koordination ist erforderlich, um unnötige Konflikte zu vermeiden, gemeinsame Ziele und Aufgaben zu ermitteln und Wissen und Expertise untereinander auszutauschen. Agenten können sich dabei kooperativ verhalten und anderen Agenten ihre Informationen und ihre Fähigkeiten zur Verfügung stellen. Agenten können aber auch indifferent eingestellt sein und ausschließlich in ihrem eigenen Interesse handeln. Auch Konkurrenzsituationen können eine Koordination zwischen den Agenten erforderlich machen.

Koordination zwischen Agenten setzt die Fähigkeit zur Kommunikation voraus. Agenten, die ihr Handeln untereinander koordinieren wollen, um individuelle oder gemeinsame Ziele zu erreichen, müssen auch in der Lage sein, Informationen auszutauschen. Die Kommunikation zwischen Agenten kann dabei implizit oder explizit erfolgen. Implizite Kommunikationsmechanismen finden nach dem Vorbild der Blackboard-Metapher [New62] über die Wirkung von Aktionen in der gemeinsamen Umgebung statt [WJ04]. Die Agenten verändern dabei die Beschaffenheit der Umgebung, was von anderen Agenten wahrgenommen und interpretiert werden kann. In der Regel wird aber von einem expliziten Kommunikationsmechanismus ausgegangen, in dem die Kommunikationspartner gerichtete Nachrichten austauschen. Explizite Kommunikationsmechanismen setzen öffentliche Schnittstellen für den Nachrichtenaustausch voraus, die den Kommunikationspartnern bekannt sein

müssen. Die Kommunikation selbst kann dabei auf sehr hohem Abstraktionsniveau stattfinden wie z.B. Verhandlungen zur Abstimmung von Aktionen im Rahmen des Verteilten Planens und Problemlösens [Dur99, Dur01, Mül93] oder das Auflösen von Konflikten.

Um Nachrichten zwischen Agenten austauschen zu können, werden einheitliche Kommunikationsprotokolle benötigt, die festlegen, wie eine Nachricht zu formulieren ist. Die in der Literatur am häufigsten verwendeten Kommunikationsprotokolle für Agenten sind die *Knowledge Query Manipulation Language* (KQML) [The11] und die von der FIPA standardisierte *Agent Communication Language* (ACL) [Age11c]. Die Grundidee beider Protokolle besteht darin, dass alle Informationen, die notwendig sind, um den Inhalt einer Nachricht zu verstehen, bereits in der Nachricht selbst formuliert sind. Der Inhalt der Nachricht kann in einer beliebigen Sprache wie z.B. Prolog, SQL oder auch KQML bzw. ACL ausgedrückt werden. Der Sender hat die Möglichkeit, die verwendete Sprache in der Nachricht selbst zu hinterlegen. Beide Protokolle unterstützen außerdem die Angabe von so genannten Performativen, die zusätzliche Informationen über die kommunikative Geltung einer Nachricht enthalten. KQML und ACL orientieren sich somit an der Sprechakt-Theorie von Austin [Aus62] und Searle [Sea90], die das Wesen sprachlicher Handlungen zu klassifizieren versucht. Kommunikation wird dabei als Folge von Sprechakten verstanden, in der jedem Sprechakt ein illokutionärer Akt zugeordnet werden kann (z.b. request oder inform).

Kommunikation zwischen zwei Agenten besteht in der Regel nicht aus isolierten Nachrichten, sondern aus einem wechselseitigen Austausch von zusammenhängenden Nachrichten (Konversation, Dialog). Wenn ein Agent eine Nachricht sendet, dann hat er eine bestimmte Erwartungshaltung, ob und wie der Empfänger auf diese Nachricht antwortet. Diese Erwartung ist für gewöhnlich nicht in der Nachricht selbst formuliert. Denkbar einfachstes Beispiel für eine nachrichtenübergreifende Konversation ist eine Frage/Antwort-Folge. Komplexere Konversationen mit alternativen Abfolgen können für einen Agenten jedoch einen hohen Aufwand bedeuten, da er ständig die verschiedenen Alternativen verwalten muss. Aus diesem Grund erscheint es sinnvoll, wenn sich miteinander kommunizierende Agenten im Kontext einer Konversation auf eine mögliche Abfolge von Nachrichten einigen [GHB00]. Solche Abfolgen können durch Interaktionsprotokolle beschrieben werden. Während Kommunikationsprotokolle wie KQML oder ACL die Mechanismen zum Austausch einer Nachricht festlegen, schreiben Interaktionsprotokolle vor, wie eine Konversation zu führen ist. Interaktionsprotokolle begrenzen die Anzahl der möglichen Nachrichten

zwischen den Kommunikationspartnern und vereinfachen somit den Nachrichtenaustausch. Sie sollen zudem sicherstellen, dass der Nachrichtenaustausch zwischen Agenten einem einheitlichen Muster folgt und von beiden Seiten korrekt interpretiert werden kann. Konkret bestehen diese Einschränkungen darin, die möglichen Folgen von Performativen, die im Kontext einer Konversation ausgetauscht werden können, festzulegen. Neben einfachen Frage/Antwort-Protokollen sind auch komplexere Interaktionsprotokolle wie z.B. das Kontrakt-Netz-Protokoll [Smi88] in Multiagentensystemen gebräuchlich. Ein geeigneter Formalismus zur Modellierung von Sozialität in Agentensystemen ist somit in der Lage, Kommunikations- und Interaktionsprotokolle darzustellen.

2.3 Agentenmodelle

In diesem Abschnitt werden die in der Literatur verbreiteten Ansätze zur Modellierung von Agentensystemen auf Grundlage der folgenden Kriterien untersucht:

- **Formalität:** Der Ansatz sollte eine formale Syntax und Semantik besitzen, um formale Aussagen über das Modellverhalten zu ermöglichen.
- **Anschaulichkeit:** Der Ansatz sollte eine anschauliche graphische Repräsentation besitzen.
- **Agentenorientierung:** Der Ansatz sollte die Modellierung der charakteristischen Agentenmerkmale Autonomie, Reaktivität, Proaktivität und Sozialität unterstützen.

Die verschiedenen Ansätze, die in diesem Abschnitt behandelt werden, können sehr unterschiedlich hinsichtlich ihrer Notation und ihrer Ausdrucksfähigkeit sein. Sie haben ihre Ursprünge entweder in der Softwaretechnik oder in der Theoretischen Informatik. Für eine Vertiefung sei auf die entsprechenden bibliographischen Hinweise in den einzelnen Abschnitten verwiesen.

2.3.1 UML/AUML

Die Unified Modeling Language ist eine seit 1997 von der *Object Management Group* (OMG) standardisierte (Menge von) Modellierungssprache(n), die sich in der Softwaretechnik durchgesetzt hat. Die UML führt eine Reihe von Begriffen für die ob-

jektorientierte Modellierung von Software-Systemen ein und legt die möglichen Beziehungen zwischen diesen Begriffen fest. Wesentliche Ausschnitte der resultierenden Modelle können dabei in Form von verschiedenen Diagrammen anschaulich dargestellt werden. Die UML hat den hohen Anspruch, ein System aus verschiedenen Sichten einheitlich darzustellen, indem sie unterschiedliche Diagrammtypen für die Modellierung zur Verfügung stellt [RJB04]. In der aktuellen Version 2.3 von März 2010 werden insgesamt 14 Diagrammtypen unterstützt, um die Struktur (statische Sicht) und das Verhalten (dynamische Sicht) eines Systems zu spezifizieren [Gro11]. Zu den Strukturdiagrammen zählen:

- *Class Diagram:*
  Beschreibt die Klassen des Software-Systems und stellt ihre Assoziationen, Aggregationen und Kompositionen dar.
- *Component Diagram:*
  Beschreibt die Komponenten, aus denen das Software-System zusammengesetzt ist, und über welche Schnittstellen die Komponenten miteinander verbunden sind.
- *Composite Structure Diagram:*
  Beschreibt die interne Struktur von Klassen und ihre Interaktionsbeziehungen zu anderen Systembestandteilen.
- *Deployment Diagram:*
  Beschreibt die Bestandteile der physischen Entwicklungs- und Laufzeitumgebung des Software-Systems.
- *Object Diagram:*
  Beschreibt Objekte, Assoziationen und Werte in konkreten Zuständen des Software-Systems.
- *Package Diagram:*
  Beschreibt die logischen Gruppen, aus denen das Software-System zusammengesetzt ist, und stellt ihre Abhängigkeiten dar.
- *Profil Diagram:*
  Beschreibt benutzerdefinierte Stereotype.

Die Verhaltensdiagramme umfassen:

- *Use Case Diagram:*
  Beschreibt die Anwendungsfälle des Software-Systems aus Sicht der Akteure und stellt die Abhängigkeiten zwischen den Anwendungsfällen dar.
- *Activity Diagram:*
  Beschreibt das Verhalten des Software-Systems durch Abfolgen von einzelnen Aktivitäten.
- *Statechart Diagram:*
  Beschreibt die Zustände und Zustandsübergänge des Software-Systems.
- *Communication Diagram:*
  Beschreibt die Interaktionen, die zwischen den Objekten des Software-Systems möglich sind.
- *Interaction Overview Diagram:*
  Beschreibt den Zusammenhang zwischen den Interaktionen im Software-System.
- *Sequence Diagram:*
  Beschreibt den Informationsaustausch zwischen den Objekten des Software-Systems.
- *Timing Diagram:*
  Beschreibt das Verhalten des Software-Systems in Abhängigkeiten von der Zeit.

Der größte Vorteil der UML besteht in ihrer graphischen Notation, was maßgeblich zu ihrer Verbreitung in der Softwaretechnik beigetragen hat. Grundsätzlich problematisch ist die große Vielfalt an unterschiedlichen Diagrammtypen und Sprachelementen, die nicht immer klar voneinander abgegrenzt werden können und teilweise redundant sind. Da die UML zudem auf dem objektorientierten Paradigma aufbaut, werden die abstrakteren Konzepte der Agentenorientierung wie Autonomie, Reaktivität, Proaktivität und Sozialität nicht ausreichend unterstützt [BBMO02]. Nebenläufigkeit und Nichtdeterminismus sind in einigen Verhaltensdiagrammen repräsentierbar, allerdings ist nicht immer erkennbar, wie die statische und die dynamische Sicht eines Systems zusammenhängen. Auch ist unklar, wie mentale Zustände eines Agenten dargestellt werden können.

Aufgrund der mangelnden Unterstützung agentenorientierter Konzepte wurden viele Ansätze in der Literatur vorgeschlagen, die die UML erweitern oder die vorhandenen Sprachmittel neu kombinieren. Eine Übersicht findet sich in [BM03] und [Tve01].

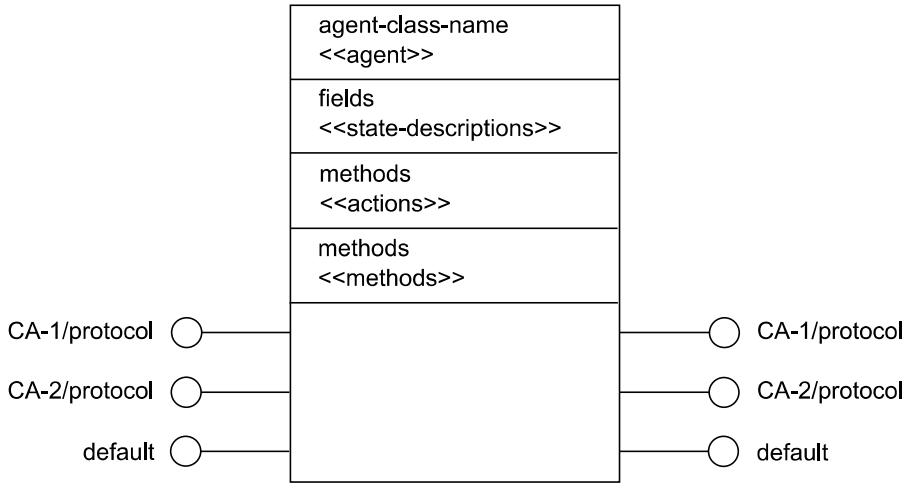


Abbildung 2.2: Schematische Darstellung eines Agent Class Diagrams in Agent UML nach [BMO00].

Die populärste Erweiterung der UML in diesem Zusammenhang ist die Agent UML [UML11]. Agent UML entstand als Zusammenarbeit zwischen der FIPA und der OMG und wird in den Spezifikationen des FIPA-Standards als Modellierungssprache für Agentensysteme propagiert. Die Erweiterungen beziehen sich auf die UML Version 1.4 und einige Ideen haben grundlegende Schwächen der UML identifiziert, so dass sie seit Versionen 2.0 in den UML-Standard übernommen wurden [BM03]. Die neuen Sprachmittel sollen dabei sowohl die statische Sicht als auch die dynamische Sicht auf einen Agenten geeignet unterstützen. Dazu werden *Agent Class Diagrams* und *Protocol Diagrams* eingeführt:

- *Agent Class Diagrams:*
  Agent Class Diagrams erweitern Class Diagrams, indem sie die klassische, objektorientierte Aufteilung in Attribute und Methoden durch eine agentenorientierte Aufteilung in Zustände, Aktionen und Kommunikationsschnittstellen ersetzen. Durch geeignete Stereotype können auch mentale Zustände eines Agenten modelliert werden. Die Aktionen eines Agenten werden in Form von Methoden dargestellt, wobei zusätzliche Vor- und Nachbedingungen hinterlegt werden können. Proaktive Aktionen setzen einen bestimmten inneren Zustand des Agenten voraus, d.h. sie werden vom Agenten selbst ausgelöst. Reaktive Aktionen sind Reaktionen auf äußere Umstände und werden von Ereignissen der Umgebung wie z.B. durch den Empfang von Nachrichten aus-

gelöst. Darüber hinaus sind spezielle Kommunikationsschnittstellen im Diagramm vorgesehen, um die Interaktionen zwischen den Agenten abzubilden. Eine Schnittstelle legt dabei sowohl die Performative (Communicative Act, CA) als auch das jeweilige Interaktionsprotokoll einer entsprechenden Nachricht fest. Stimmt die Performative und das Protokoll einer ein- oder ausgehenden Nachricht mit einer Schnittstelle überein, so wird das entsprechende Verhalten ausgelöst, das mit der Schnittstelle assoziiert ist. Durch weitere Verhaltensdiagramme des UML-Standards kann das auszulösende Verhalten genauer beschrieben werden. Abbildung 2.2 zeigt eine schematische Darstellung eines Agent Class Diagrams.

- *Protocol Diagrams:*
  Um neben den Agenteninterna auch die Interaktionen zwischen Agenten darstellen zu können, wurde in der Agent UML mit Protocol Diagrams ein neuer Diagrammtyp eingeführt. Protocol Diagrams kombinieren Sequence Diagrams mit einer erweiterten Syntax, um Interaktionsprotokolle für den Nachrichtenaustausch zwischen Agenten zu modellieren. Auf diese Weise legen sie die Performativen fest, die im Rahmen einer Konversation ausgetauscht werden können. Die erweiterte Syntax ermöglicht es, die Interaktionspfeile und die Lebenslinien eines Agenten zu verzweigen, um parallele und alternative Abläufe zu modellieren. Die Interaktionspfeile beschreiben, welche Performativen an welchem Punkt der Interaktion ausgetauscht werden. Eine Verzweigung kann bedeuten, dass an einer Stelle mehrere Nachrichten parallel verschickt werden, oder dass die entsprechende Performative vom Zustand des Agenten oder dem Inhalt der Nachricht abhängt. Mit der zusätzlichen Angabe von Kardinalitäten kann die Anzahl der Sender bzw. Empfänger einer Nachricht notiert werden, falls mehr als zwei Agenten an der Interaktion beteiligt sind. Die Lebenslinien legen die Dauer fest, in der ein Agent die Rolle des entsprechenden Interaktionsteilnehmers ausübt. In Abhängigkeit des Nachrichtenverlaufs können sich diese Lebenslinien aufspalten, wenn ein Agent an einer Stelle parallele oder alternative Nachrichtenfolgen verwaltet. Abbildung 2.3 zeigt das FIPA-English-Auction-Protocol in Form eines Protocol Diagram.

Erwartungsgemäß unterstützt die Agent UML die agentenorientierten Konzepte besser als der UML-Standard. Das größte Problem an der UML im Allgemeinen und an der Agent UML im Speziellen besteht aber in der fehlenden formalen Semantik der Sprachen. UML und Agent UML können nicht dazu verwendet werden, formale Aussagen über das Modellverhalten zu machen, weil die Systemabläufe nicht voll-

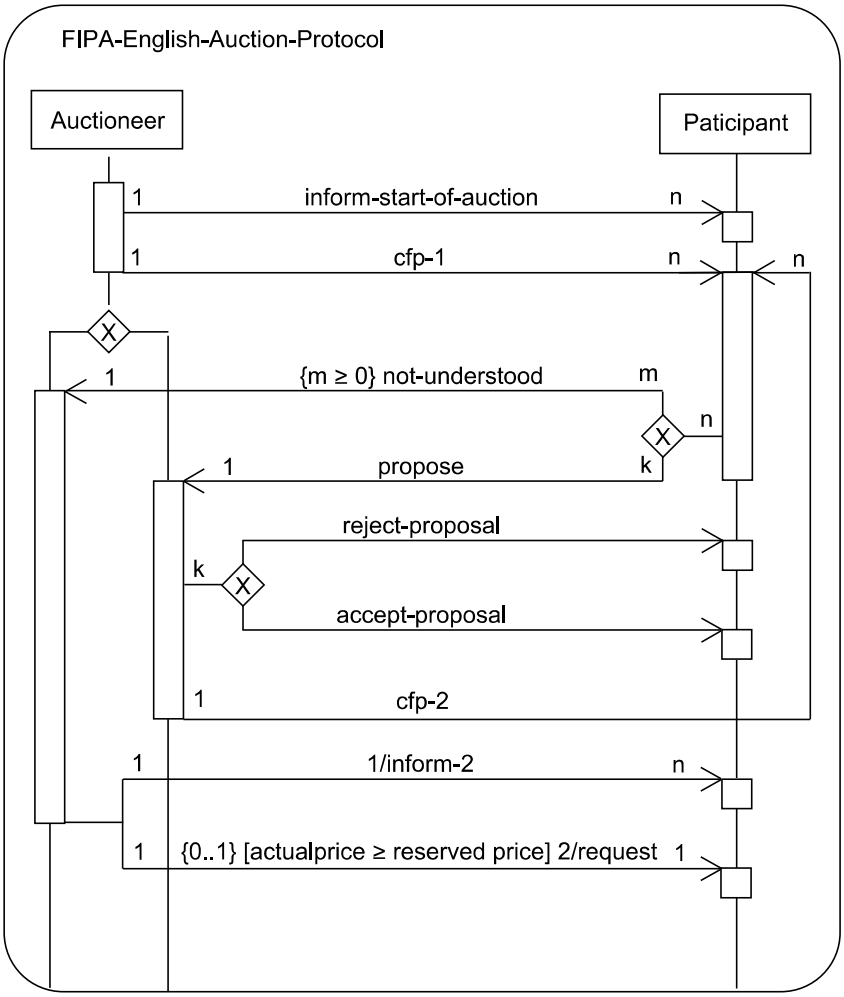


Abbildung 2.3: Darstellung des FIPA-English-Auction-Protocol in Agent UML nach [BMO00].

ständig und eindeutig durch die Modelle festgelegt werden. Einige Ansätze begegnen diesem Problem, indem sie ausgewählten UML- oder Agent UML-Diagrammen mit Modallogik [KRH03], Graphtransformationen [KGKZ09, ZHG05] oder Petri-Netzen [BP01, CM04, CMR03] nachträglich eine formale Semantik zuordnen (siehe Abschnitt 2.3.5, 2.3.6 und 2.3.7).

2.3.2 Automaten

Ein einfaches Mittel zur Beschreibung von Abläufen stellen endliche Automaten dar. Endliche Automaten sind Gegenstand der Theoretischen Informatik, finden aber auch häufig in der Softwaretechnik Verwendung, z.B. als Statechart Diagrams im Rahmen des UML-Standards. Da sie auch unabhängig von der UML als gängige Modellierungssprache zur Beschreibung von Software-Systemen in der Informatik eingesetzt werden, sollen sie an dieser Stelle separat betrachtet werden.

Die Idee von endlichen Automaten besteht darin, das Verhalten eines Systems durch alle Zustände und Zustandsübergänge zu beschreiben, die im System möglich sind. Der Zusammenhang zwischen den Zuständen und den Zustandsübergängen wird durch eine Zustandsübergangsfunktion bestimmt, die festlegt, in welche Folgezustände das Eintreten eines Ereignisses führt. Zu jedem Zeitpunkt befindet sich das System und die Umgebung immer in genau einem Zustand. Tritt ein Ereignis ein, wechselt der aktuelle Zustand je nach Art des Ereignisses in den Folgezustand, der durch die Zustandsübergangsfunktion festgelegt wird. Zeit wird dabei als diskrete Größe angenommen. Die Zustandsübergangsfunktion kann partiell sein, d.h. nicht für alle Zustands-Ereignis-Kombinationen muss ein Folgezustand definiert sein. Ein deterministischer Automat bestimmt den Folgezustand eines Ereignisses eindeutig. Sollen auch mehrdeutige Übergänge dargestellt werden, so können nichtdeterministische Automaten eingesetzt werden. Nichtdeterministische Übergänge sind dadurch gekennzeichnet, dass von einem Zustand und einem Ereignis aus mehrere Folgezustände möglich sind. Anstelle einer Zustandsübergangsfunktion bestimmt eine Zustandsübergangsrelation den Zusammenhang zwischen den Zuständen und den Ereignissen des Systems. Darüber hinaus sind mehrere Startzustände möglich. Typischerweise werden in nichtdeterministischen Automaten auch spontane Zustandsübergänge zugelassen, die keinem Ereignis zugeordnet werden (siehe z.B. [VW06]).

Endliche Automaten stellen ein einfaches Beschreibungsmittel dar und liegen gleichzeitig einem mathematischen Formalismus zu Grunde. Darüberhinaus ist es möglich, die Zustände und Zustandsübergänge eines endlichen Automaten als Knoten und Kanten eines gerichteten, beschrifteten Graphen darzustellen. Üblicherweise werden Start- und Endzustände dabei besonders gekennzeichnet. Damit besitzt ein endlicher Automat auch eine graphische Repräsentation. Die Anzahl der möglichen Zustände und Zustandsübergänge stellt allerdings das größte Problem bei der Verwendung von endlichen Automaten dar. Jede Veränderung des Systems bedeutet notwendigerweise

einen Zustandsübergang, der explizit angegeben werden muss. Dadurch wächst die Größe des Automaten explosionsartig an, insbesondere wenn mehrere Agenten und deren Zustände zu berücksichtigen sind. Außerdem stehen in endlichen Automaten keine Informationen über die Struktur des Systems zur Verfügung, so dass sich Zusammenhänge zwischen den Zustandsübergängen nicht begründen lassen. Konzepte wie Autonomie, Proaktivität, Reaktivität und Sozialität eines Agenten, die auf einem sehr viel höheren Abstraktionsniveau ansetzen, sind damit gegenstandslos. Darüber hinaus eignen sich endliche Automaten nur zur Darstellung sequentieller Prozesse. Nebenläufigkeit kann nicht (übersichtlich) dargestellt werden.

Um dem Problem der Explosion des Zustandsraumes zu begegnen, kann ein Automat um ein Gedächtnis erweitert werden. Dadurch entstehen mächtigere Konzepte wie z.B. Turing- oder Registermaschinen, die auch als Grundlage für den Berechbarkeitsbegriff in der Informatik dienen. Turing- bzw. Registermaschinen besitzen gegenüber einfachen Automaten einen zusätzlichen Speicher, in denen Symbole eines vorgegebenen Alphabets bzw. natürliche Zahlen hinterlegt werden können. Beim Programmstart wird von einer gegebenen Speicherkonfiguration ausgegangenen. Zur Laufzeit wechselt die Maschine mit jedem Ausführungsschritt in Abhängigkeit einer Konfigurationsvorschrift ihren Zustand und manipuliert dabei den Inhalt des Speichers. Dadurch lassen sich unterschiedliche Zustände des Systems in verschiedenen Speicherkonfigurationen abbilden. Viele Zustände müssen somit nicht mehr explizit angegeben werden, was dem Problem der Explosion des Zustandsraumes entgegenwirkt. Für komplexe Systeme lässt sich die Konfigurationsvorschrift einer Turing- oder Registermaschine allerdings nicht mehr in übersichtlicher Form angeben. Darüber hinaus erweist sich die Modellierung mit Turing- oder Registermaschinen aufgrund des mangelnden Abstraktionsniveaus ebenfalls als ungeeignet, um agentenorientierte Konzepte wie Autonomie, Proaktivität, Reaktivität und Sozialität eines Agenten darzustellen.

2.3.3 Zelluläre Automaten

Zelluläre Automaten wurden von John von Neumann als abstraktes Modell zur Selbstproduktion entwickelt. Während endliche Automaten auf der Vorstellung basieren, dass Aktionen zu globalen Veränderungen im System und in der Umgebung führen, besteht die Idee bei Zellulären Automaten darin, Aktionen als lokale Veränderungen zu begreifen. Dazu wird von einer Menge von Zellen ausgegangen, die in

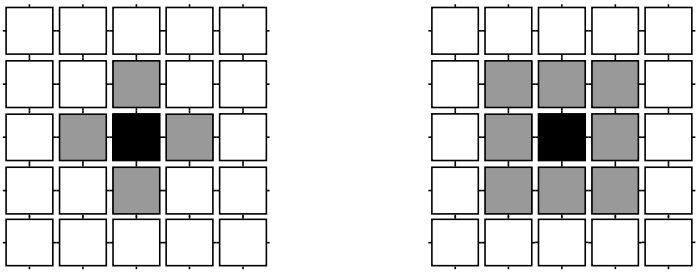


Abbildung 2.4: Graphische Darstellung eines homogenen zweidimensionalen zellulären Raums mit *Neumann*-Nachbarschaft (links) und *Moore*-Nachbarschaft (rechts).

einem Gitter angeordnet sind. Das Gitter und die Zellenmenge bilden den zellulären Raum, wobei die Anordnung und die Geometrie des Gitters eindeutig eine Nachbarschaftsbeziehung zwischen den Zellen festlegen. Prinzipiell kann das Gitter eine beliebige Dimension aufweisen, aber für gewöhnlich werden ausschließlich ein- oder zweidimensionale Strukturen betrachtet. Abbildung 2.4 zeigt ein Beispiel für einen zweidimensionalen zellulären Raum mit zwei möglichen Nachbarschaftsbeziehungen.

Zelluläre Automaten stellen (synchrone) Automatennetze dar. Jede Zelle repräsentiert einen Automaten und befindet sich zu einem Zeitpunkt t in genau einem Zustand. Der Zustand der Zelle zum Zeitpunkt $t + 1$ hängt aber nicht nur von ihrem aktuellen Zustand ab, sondern auch vom Zustand der Zellen in ihrer Nachbarschaft. Eine Zustandsübergangsfunktion legt für jede Zelle, ihren Zustand und den Zuständen ihrer Nachbarn einen Folgezustand fest. Dabei ist es möglich, dass die Zustandsübergänge für alle Zellen sehr individuell geregelt sind. Doch ein besonderer Aspekt bei der Betrachtung von Zellulären Automaten ist gerade, wie auf der Grundlage einfacher, einheitlicher Regeln auf unterster Ebene ein komplexes Gesamtverhalten auf höherer Ebene emergiert. Aus diesem Grund entsprechen die Zustandsübergänge aller Zellen in der Regel einem einheitlichem Muster (mit möglichen Ausnahmen am Rand des Gitters), d.h. alle Automaten haben die selben Schnittstellen und das selbe Verhalten (siehe [GS95] oder [VW95]).

Ein klassisches Beispiel für einen zellulären Automaten ist John Conway's *Game of Life* [Gar70]. In diesem Spiel sind die Zellen in einem endlichen aber grenzenlosen zweidimensionalen zellulären Raum angeordnet. Es gilt die Nachbarschaftsbeziehung nach Moore und jede Zelle kann nur einen von zwei Zuständen annehmen: lebendig oder tot. Eine lebendige Zelle stirbt (an Einsamkeit oder Überbevölkerung) genau

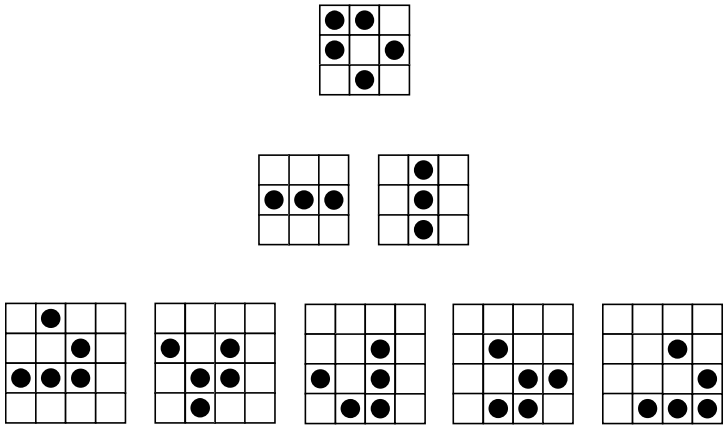


Abbildung 2.5: Einige Muster im *Game of Life*. Oben: Ein stabiles *Boat*, das in jedem Zyklus erhalten bleibt. Mitte: Ein periodischer *Blinker*, der alle zwei Zyklen oszilliert. Unten: Ein so genannter *Glider*, der sich alle vier Zyklen reproduziert und dabei eine Zelle in diagonaler Richtung bewegt.

dann, wenn weniger als zwei oder mehr als drei ihrer Nachbarn leben. Eine tote Zelle wird genau dann neu geboren, wenn drei ihrer Nachbarn leben. Wird das Spiel mit beliebigen Startzuständen der einzelnen Zellen gestartet, so zeigt sich trotz der einfachen, lokalen Regeln ein überraschend komplexes Gesamtverhalten. In Abhängigkeit der Startzustände kann eine Population vollständig aussterben oder in einen stabilen Zustand konvergieren. Bestimmte Muster eignen sich sogar dazu, unbegrenztes Wachstum zu ermöglichen. Häufig bilden sich lokale Strukturen, die solange periodisch oszillieren, bis sie sich mit einem anderen Muster überschneiden. Abbildung 2.5 zeigt einige Beispiele. Das Gesamtverhalten des Spiels ist wesentlich komplexer als es die einfachen Regeln vermuten lassen, d.h. emergent.

Zelluläre Automaten eignen sich zur Beschreibung von dynamischen Systemen, deren Verhalten auf räumlichen Beziehungen und lokalen Verhaltensregeln basiert. Es bestehen Ähnlichkeiten zwischen den emergenten Phänomenen in Zellulären Automaten und in Multiagentensystemen, insbesondere in reaktiven Architekturen. Dennoch eignen sich Zelluläre Automaten nur begrenzt zur Beschreibung von Agentensystemen. Die Nachbarschaft und die Zustandsübergänge einer Zelle können zwar beliebig komplex sein, die entsprechenden Verbindungen bleiben jedoch über die gesamte Lebensdauer des Systems erhalten und können nicht mehr geändert werden. Eine Zelle kann keinen Einfluss darauf nehmen, mit wem sie gerade kommuniziert oder wann sie ihren Zustand wechselt. Die Zustandsübergänge finden für alle Zellen

immer gleichzeitig statt. Dies widerspricht dem Autonomiecharakter von Agenten, nach dem es ihnen möglich ist, frei zu entscheiden, wann und mit wem sie Informationen austauschen und welche Beziehungen sie mit anderen Agenten eingehen. Agenten können die Initiative ergreifen und selbst entscheiden, ob und wie sie auf Reize von Außen reagieren. Sie können ihr Verhalten zur Laufzeit anpassen und unabhängig von ihrer Umgebung ihren Zustand wechseln. Darüber hinaus operieren Agenten nebenläufig und synchronisieren ihre Aktionen untereinander nicht gezwungenermaßen.

2.3.4 Algebraische Spezifikation

In der (klassischen) Künstlichen Intelligenz werden logische Sprachen dazu verwendet, um Wissensbasen von Agenten (insbesondere in deliberativen Architekturen) zu formulieren und Schlussfolgerungen durchzuführen (siehe z.B. [RN04]). Sie können aber auch dazu verwendet werden, um die Anforderungen und den Entwurf von Software-Systemen in einem gemeinsamen sprachlichen Rahmen formal auszudrücken. Logikbasierte Verfahren waren die ersten Ansätze in der Softwaretechnik, um die Korrektheit von Programmen nachzuweisen und gehen auf die Arbeiten von Hoare [Hoa69] zurück. Das entsprechende Konzept ist als Algebraische Spezifikation in der (Theoretischen) Informatik verbreitet [EM85, EM90].

Die Idee von algebraischen Spezifikationen besteht darin, alle wichtigen Aspekte eines Softwaresystems durch abstrakte Datentypen mengentheoretisch zu beschreiben. Ein abstrakter Datentyp besteht dabei, wie in der Objektorientierung auch, aus einer oder mehreren Mengen von Daten und einer Menge von Operationen, die auf diesen Daten erklärt sind. Zu diesem Zweck ist für jeden abstrakten Datentyp eine Signatur erklärt, die auf syntaktischer Ebene Bezeichner für Sorten und Operationen festlegt. Die semantische Ebene wird durch eine Struktur abgebildet, die jedem Sortensymbol der Signatur eine konkrete Menge und jedem Operationssymbol eine konkrete Funktion zuordnet.[1] Die Menge der möglichen Strukturen für eine Signatur ist normalerweise unendlich groß. Im Rahmen der Anforderungsspezifikation soll diese Menge jedoch auf die geeigneten Kandidaten eingeschränkt werden. Dazu werden die Anforderungen an das System durch eine Reihe von Beziehungen zwischen den Daten und Operationen beschrieben. In der Regel wird dabei eine Menge von

[1] Für eine formale Betrachtung von Signaturen und Strukturen sei auf das nächste Kapitel verwiesen.

Gleichungen aufgestellt, die festlegen, welche Zusammenhänge zwischen den Daten und Operationen untereinander gelten. Häufig ist die Anforderungsspezifikation aber noch sehr offen und legt noch nicht konkret fest, wie die Operationen zu realisieren sind, sondern nur, was sie leisten sollen. Der Entwurf wird in einer Entwurfsspezifikation formuliert, in der die Zusammenhänge zwischen den Daten und Operationen vollständig und eindeutig durch entsprechende Gleichungen festgelegt werden. Dabei hängt es vom Einzelfall ab, wo genau Anforderungs- und Entwurfsspezifikation voneinander abzugrenzen sind. Sind die Anforderungen zu offen, so können auf Grund zu vieler Freiheiten inakzeptable Entwürfe entstehen. Sind die Anforderungen dagegen zu restriktiv, kann dies den Entwurf unnötig erschweren [Win90]. Um die Korrektheit eines Entwurfs nachzuweisen, ist zu zeigen, dass die durch die Entwurfsspezifikation festgelegte Struktur ebenfalls ein Modell für die Anforderungsspezifikation ist. Die Gleichungen in der Entwurfsspezifikation bilden dabei ein Axiomensystem, das das Verhalten des Systems beschreibt. Um den Entwurf gegen die Anforderungen zu verifizieren, muss gezeigt werden, dass jedes Theorem der Anforderungsspezifikation aus dem Axiomensystem abgeleitet werden kann, d.h. die Verifikation reduziert sich zu einem (syntaktischen) Beweisproblem.

Algebraische Spezifikationen stellen eine strukturierte und kompakte Methode für die formale Spezifikation von Software-Systemen dar. Sie werden von Spezifikationssprachen wie CASL [BM04], OBJ [GM00], Z [Spi92] oder VDM [Jon90] unterstützt. Korrektheitsnachweise können mit den Mitteln der mathematischen Logik durchgeführt werden und lassen sich mit vollautomatischen oder interaktiven Theorem-Beweisern unterstützen. Einige Werkzeuge wie z.B. HETS [MML07] unterstützen sogar den Austausch zwischen verschiedenen Spezifikationssprachen und können unterschiedliche Theorem-Beweiser integrieren. Darüber hinaus ist die axiomatische Verifikation eine vollständige Methode, deren Leistungsfähigkeit nur durch die Fähigkeiten des Benutzers bzw. des Theorem-Beweisers begrenzt ist. Obwohl Algebraische Spezifikationen viele Vorteile bieten, sind sie für die Modellierung von Agentensystemen nur bedingt geeignet. Die effektorischen, sensorischen und kommunikativen Schnittstellen eines Agenten lassen sich wie beim objektorientierten Ansatz der UML mit den vorhandenen Sprachmitteln nicht immer adäquat darstellen. Somit bleibt fraglich, was im Bezug auf die Agentenorientierung durch Modelle auf dieser Abstraktionsebene gewonnen wird [Fis97, Wag97]. Eine graphische Repräsentation von Algebraischen Spezifikationen ist ebenfalls nicht vorgesehen. Die Verfügbarkeit einer graphischen Repräsentation kann für eine Modellierungssprache

jedoch ein entscheidendes Kriterium in der Praxis sein, wie der Erfolg der UML zeigt.

2.3.5 Modal- und Temporallogik

Mit der klassischen Logik können Aussagen und Schlussfolgerungen über *eine Welt* gemacht werden. Bei der Modellierung von Agentensystemen müssen jedoch auch häufig Zusammenhänge zwischen verschiedenen *möglichen Welten* berücksichtigt werden. Aus diesem Grund kommen häufig erweiterte Formalismen wie Modal- oder Temporallogik bei der Modellierung von Agentensystemen zum Einsatz. In diesem Zusammenhang ist insbesondere die BDI-Logik nach Rao und Georgeff [RG91] zu nennen. Der Ansatz ist inspiriert von der Theorie der menschlichen Entscheidungsfindung nach Bratman [Bra87] und soll den normativen Zusammenhang zwischen informationsbezogenen Zuständen (Beliefs), affektiven Zuständen (Desires) und konativen Zuständen (Intentions) formalisieren. Weitere wichtige Ansätze, die direkt oder indirekt auf Modal- und Temporallogik aufbauen, sind die agentenorientierte Programmiersprache (AOP) nach Shoham [Sho93] und die intentionale Logik nach Cohen und Levesque [CL90]. Eine Übersicht nichtklassischer Logiken, die für die Modellierung von Agentensystemen geeignet sind, findet sich in [vdH01, SRG99].

Die Modallogik erweitert die klassische Logik um die modalen Operatoren $\Box$ (notwendigerweise) und $\Diamond$ (möglicherweise) (siehe z.B. [Eme90, MK06]). Für eine mögliche Semantik wird von einer Menge von möglichen Welten ausgegangen. Eine modallogische Struktur (Kripke-Struktur), legt die Menge der möglichen Welten und eine binäre Erreichbarkeitsrelation zwischen den Welten fest. Zusätzlich definiert sie eine Abbildung, die jeder atomaren Aussage in einer Welt einen eindeutigen Wahrheitswert zuordnet. Eine Formel gilt in einer Welt *notwendigerweise* genau dann, wenn sie in allen erreichbaren Welten gilt. Entsprechend gilt eine Formel in einer Welt *möglicherweise* genau dann, wenn sie in mindestens einer erreichbaren Welt gilt. Die Temporallogik basiert auf der Modallogik und soll Zusammenhänge ausdrücken, die eine zeitliche Dimension aufweisen. Zeit wird dabei als diskrete Größe angenommen. Die Menge der möglichen Zeitpunkte ist durch die Menge der möglichen Welten gegeben und der Übergang zwischen zwei Zeitpunkten wird durch die Erreichbarkeitsrelation, die über den Welten definiert ist, fixiert. Die entsprechenden Systemabläufe sind dadurch vollständig und eindeutig bestimmt. Die modalen Operatoren $\Box$ und $\Diamond$ lassen sich im einfachsten Fall temporallogisch mit *immer*

bzw. *irgendwann* übersetzen. Darüber hinaus ist auf der Temporallogik der so genannte *until*-Operator erklärt, mit dessen Hilfe weitere temporallogische Operatoren definiert werden können. Die Temporallogik eignet sich somit insbesondere zur Beschreibung von Systemen, die über einen längeren Zeitraum aktiv sind.

Mit der Modal- bzw. Temporallogik können Agentensysteme ebenfalls axiomatisch spezifiziert und verifiziert werden [Woo92]. Für modal- bzw. temporallogische Spezifikationen wird im Gegensatz zu Algebraischen Spezifikationen mit Model Checking aber ein anderer Ansatz zur Verfikation propagiert. Model Checking ist ein voll automatisches Beweisverfahren, das nicht auf syntaktischer, sondern auf semantischer Ebene arbeitet. Die Idee besteht darin, ausgehend von einer modal- bzw. temporallogischen Entwurfsspezifikation (semantisch) zu überprüfen, ob jedes Theorem der Anforderungsspezifikation in den entsprechenden Systemabläufen erfüllt ist. Wenn eine nachzuweisende Eigenschaft verletzt wird, kann der Fehler sowie der fehlerauslösende Systemablauf präsentiert werden. Die Vorteile dieses Ansatzes gegenüber axiomatischen Verfahren liegen in der Komplexität [HV91]. Der Erfolg von Model Checking ist aber vor allem auf den Fortschritt von entsprechenden Werkzeugen wie Spin [Hol03] zurückzuführen. Ansätze, die Model Checking für Agentensysteme vorschlagen, finden daher in der Literatur eine gewisse Verbreitung. Konkrete Beispiele sind [WHFP06, BVF+03, BB05, BVF+03, BGS98].

Modal- bzw. temporallogische Spezifikationen besitzen gegenüber der klassischen Logik einige Vorteile bei der Beschreibung von Agentensystemen. Insbesondere mentale Zustände wie Wissen und Glauben können auf diese Weise sehr einfach dargestellt werden [Hin62]. Es ist allerdings häufig unklar, wie das Verhalten eines Agenten in eine modal- bzw. temporallogische Form überführt werden kann [WC00]. Eine graphische Repräsentation von modal- und temporallogischen Modellen ist ebenfalls nicht verfügbar. Darüber hinaus sind die entsprechenden Modelle sehr abstrakt und eignen sich in der Regel nicht dazu, um sie in eine konkrete Programmiersprache zu überführen [CHSB96].

> "Theories of agents are often abstract and obtuse and not related to concrete computational models. For example, formalisms such as temporal logics and multi-modal logics seem some distance from agents that have actually been implemented." [dFL+97, S. 315]

2.3.6 Graphtransformationen

Graphen sind in der Informatik zur Darstellung von Entitäten und ihren Beziehungen allgegenwärtig. Beispiele sind endliche Automaten, UML-Diagramme, Entity-Relationship-Modelle, Flussdiagramme oder Petri-Netze. Graphen bestehen grundsätzlich aus einer Menge von Knoten, sowie einer Menge von Kanten, die die Knoten miteinander verbinden. Weitere Bestandteile können Abbildungen sein, die Knoten und Kanten eines Graphen mit Elementen oder Wörtern eines beliebigen Alphabets beschriften. Den Farben und Formen bei der Darstellung eines Graphen sind im Allgemeinen kaum Grenzen gesetzt. In der Regel werden Knoten als Punkte, Ellipsen oder Rechtecke dargestellt, während Kanten als Pfeile zwischen den Knoten gezeichnet werden.

Graphen eignen sich hervorragend dazu, die Bestandteile eines Systems und deren Beziehungen anschaulich darzustellen. Für die Beschreibung des Verhaltens erscheint es daher sinnvoll, eine Dynamik auf dieser Darstellung zu erklären. Ein verbreiteter Ansatz in diesem Zusammenhang ist der semantische Bereich der Graphtransformationen [Roz97, EEKR99, EKMR99]. Die Grundidee besteht darin, die Zustände des Systems durch Graphen und die Zustandsübergänge durch Veränderungen an den Graphen zu repräsentieren. Ausgangspunkt ist eine Menge von Regeln, die lokale Änderungen eines Graphen beschreiben und festlegen, welche Elemente beim Anwenden einer Regel zu entfernen und welche Elemente hinzuzufügen sind. Graphtransformationssysteme kombinieren damit die Vorteile von Graphen und regelbasierten Ansätzen [AEH$^+$99].

Eine Graphtransformationsregel r besteht aus einem Graphen L (linke Regelseite) und einem Graphen R (rechte Regelseite). Beim so genannten *double pushout* wird der Zusammenhang von L und R durch einen weiteren Graphen K (Klebegraph) und zwei Graphmorphismen $m_l : K \to L$ und $m_r : K \to R$ hergestellt. Im einfachsten Fall ist K dabei ein Teilgraph von L und R. Für die Transformation eines Graphen G in einen Graphen H durch Anwenden einer Regel r wird zunächst (durch einen geeigneten Graphmorphismus) eine Instanz von L in G identifiziert. Daraufhin wird der Zwischengraph Z erzeugt, indem aus G alle Knoten und Kanten gelöscht werden, die in L, aber nicht in K enthalten sind. Anschließend wird der Ergebnisgraph H erzeugt, indem alle Knoten und Kanten in Z hinzugefügt werden, die in R, aber nicht in K enthalten sind. Dabei ist die Anwendung einer Regel nicht möglich, wenn durch das Entfernen der entsprechenden Elemente *hängende* Kanten entstehen (contact

condition) oder ein zu entfernendes und ein zu erhaltendes Element auf den selben Knoten abgebildet werden (identification condition). Beim *single pushout* wird der Zusammenhang von L und R stattdessen durch einen partiellen Graphmorphismus $m : L \to R$ hergestellt. Die durch m in Beziehung gesetzten Elemente bleiben erhalten, die nicht in Beziehung gesetzten Elemente aus L werden gelöscht und die aus R werden hinzugefügt. Das Löschen von Elementen hat Vorrang vor deren Erhaltung und hängende Kanten werden automatisch mitgelöscht. Als zusätzliche Voraussetzung für die Anwendbarkeit einer Regel ist es in beiden Ansätzen darüber hinaus möglich, so genannte Negativbedingungen zu formulieren, d.h. Bestandteile eines Graphen festzulegen, die für die Anwendung einer Regel nicht im Graphen vorhanden sein dürfen. Eine genauere Darstellung findet sich in [HHT96].

Eine Möglichkeit zur Modellierung von Agentensystemen mit Graphtransformationen besteht darin, ausgewählten UML-Diagrammtypen nachträglich eine formale Semantik zuzuordnen. So werden in [DHK02, DHK01, DH00, DH04] die Interaktionen der Agenten von Graphtransformationsregeln repräsentiert, deren Regelseiten sich auf Object Diagrams der UML beziehen. Jede Regel beschreibt den Effekt, den die Interaktion auf das Gesamtsystem hat und gibt den Zustand vor und nach der Interaktion in Form von entsprechenden Object Diagrams wieder. Es ist möglich, ein solches Graphtransformationssystem in eine temporallogische Form zu überführen und durch Model Checking zu verifizieren [DH04]. Von besonderer Bedeutung ist dabei die Frage, ob jede Folge von Regelanwendungen in endlich vielen Schritten in einen Zielzustand führt. Ein ähnlicher Ansatz zur Formalisierung der UML wird in [KGKZ09, ZHG05] vorgeschlagen. Dem Ansatz gelingt es, mehrere unterschiedliche UML-Diagrammtypen im Rahmen der vorgeschlagenen Semantik zu integrieren und auf diese Weise einen Großteil der UML zu berücksichtigten.

Die Modellierung von Agentensystemen ist auch mit *Autonomen Transformationseinheiten* [HKHK$^+$07, HKK09, HKK06, KK07, KK10] möglich. Bei diesem Ansatz wird die Umgebung des Systems von einem Graphen repräsentiert, während die Agenten in Form von Graph-Transformationseinheiten vorliegen. Jede Einheit *aut* = (*goal*, *rules*, *control*) verfügt über ein Ziel *goal*, eine Menge von Aktionen *rules* und eine Steuereinheit *control*. Das Ziel *goal* kann ein beliebiger Ausdruck sein, dessen Interpretation eindeutig eine Klasse von Graphen beschreibt. Jeder durch *goal* festgelegte Graph stellt einen gewünschten Endzustand der Umgebung dar. Das Aktionsrepertoire *rules* der Einheit besteht aus einer Menge von Graphtransformationsregeln, deren geeignete Anwendung den Umgebungsgraphen in einen Zielgraphen aus *goal* überführen soll. Können in einer Situation mehrere Regeln auf unterschied-

liche Art zur Anwendung kommen, findet die Entscheidung nichtdeterministisch statt. Die Steuereinheit *control* legt zusätzliche Restriktionen fest, in welchen Situationen welche Regeln anwendbar sind und schränkt den möglichen Nichtdeterminismus ein. Entsprechende Restriktionen können z.B. reguläre Ausdrücke sein, die festlegen, welche Folgen von Regelanwendungen erlaubt sind [Kre02]. Autonome Transformationseinheiten können auch kombiniert werden, indem eine Einheit eine beliebige Menge anderer Einheiten importiert. Das Aktionsrepertoire des Importeurs wird in diesem Fall durch die Regeln der importierten Einheit ergänzt, die ihrerseits wiederum andere Einheiten importieren kann. Durch die Lokalisierung von Regeln in Autonomen Transformationseinheiten und ihren Kombinationen ist es möglich, komplexe Systeme in kompakter und strukturierter Form zu beschreiben. Ein System Autonomer Transformationseinheiten $S = (A, I, T)$ besteht dabei aus einer Menge A von Autonomen Transformationseinheiten, einem Klassenausdruck I, der die initialen Umgebungen festlegt und einem Klassenausdruck T zur Festlegung der terminalen Umgebung, d.h. den gemeinsamen Zielen aller Agenten des Systems. Jede Einheit des Systems ist gleichberechtigt und operiert autonom, da keine zentrale Steuerung existiert.

Komplexe Systeme werden häufig in der Informatik durch Graphen veranschaulicht. Es erscheint daher sinnvoll, auch die Struktur von Agentensystemen mit Graphen und das Verhalten von Agenten mit Graphtransformationen zu beschreiben [KK00]. Graphtransformationen sind dabei ein sehr allgemeiner Ansatz, um dynamische Vorgänge zu modellieren. Die Stärken dieses Ansatzes resultieren aus der visuellen Repräsentation und dem darunter liegenden algebraischen Formalismus. Bei der Modellierung von Agentensystemen ist es möglich, die sensorischen und effektorischen Schnittstellen eines Agenten durch entsprechende Regeln zu repräsentieren. Die linken Regelseiten stellen dabei die Sensorik der Einheit dar, während die rechten Regelseiten die Effektorik beinhalten. Nebenläufigkeit und Nichtdeterminismus können auf diese Weise problemlos berücksichtigt werden. Kommunikation lässt sich nach dem Vorbild der Blackboard Metapher über die Veränderung des Umgebungsgraphen darstellen [KK00]. Auch die explizite Repräsentation von Zielen ist in Autonomen Transformationseinheiten möglich. Unklar bleibt, wie sich dynamische Ziele eines Agenten darstellen lassen.

2.3.7 Petri-Netze

Petri-Netze sind ein verbreiteter Formalismus zur Spezifikation nebenläufiger Systeme. Wie Zelluläre Automaten stellen auch Petri-Netze ein Netzwerk von Automaten dar. Die Zustandsübergänge der einzelnen Automaten finden jedoch asynchron statt, weshalb man auch von einem asynchronen Automatennetz spricht. Der dynamische Charakter von Petri-Netzen eignet sich ideal für die Modellierung von Systemen, in denen das Gesamtverhalten aus den dynamischen Wechselwirkungen der einzelnen Bestandteile resultiert [Ber04]. Die Ausführbarkeit von Petri-Netzen erlaubt zudem die Konstruktion von Simulatoren, so dass sich ausgewählte Ausschnitte des Modellverhaltens in einer Simulation überprüfen lassen. Darüber hinaus stehen für Petri-Netze zahlreiche Werkzeuge wie z.B. CPN/Tools [BLMJ$^+$01] oder Renew [KWD$^+$04] zur Verfügung, die die Modellierung, Simulation und Analyse entsprechender Modelle unterstützen (eine Übersicht findet sich in [Pet11]). Grundlage aller Petri-Netz-Arten ist ein bipartiter gerichteter Graph, der aus *Stellen* und *Transitionen* besteht. Die Stellen repräsentieren die statischen (passiven) Elemente des Systems, während die Transitionen die dynamischen (aktiven) Elemente darstellen. Die möglichen Wechselwirkungen zwischen den Aktiva und Passiva des Systems werden durch eine Flussrelation abgebildet, die die Verbindungen zwischen den Stellen und den Transitionen eindeutig festlegt. Eine *Markierung* auf den Stellen beschreibt den jeweils aktuellen Zustand des Systems.

Als klassische Petri-Netz-Variante gelten die so genannten S/T-Systeme ohne Kapazitäten [DR98, Rei82, Sta90]. Ein S/T-System $N = (P, T, F, W, M_0)$ besteht aus einer endlichen Menge von Stellen P, einer endlichen Menge von Transitionen T mit $P \cap T = \emptyset$, einer Menge von Kanten $F \subseteq (P \times T) \cup (T \times P)$, einer Abbildung $W : F \to \mathbb{N}_+$, die jeder Kante ein positives Gewicht zuordnet, und einer (Anfangs-) Markierung $M_0 : P \to \mathbb{N}$, die jeder Stelle eine Anzahl von Marken zuordnet. Der Vorbereich einer Transition ist durch $Ft = \{x \mid (x,t) \in F\}$ und der Nachbereich durch $tF = \{x \mid (t,x) \in F\}$ gegeben. Darüber hinaus ist $W(x,y) = 0$ für alle $(x,y) \notin F$. S/T-Systeme erweitern die ursprünglichen B/E-Systeme, in denen die Stellen entweder eine oder keine Marke enthalten können und jede Kante einfach gewichtet ist.

In der graphischen Darstellung werden Stellen üblicherweise als Kreise und Transitionen als Rechtecke gezeichnet. Die Anzahl der Marken auf einer Stelle wird durch schwarze Punkte angedeutet. Die semantische Grundlage bildet die Schaltregel. Die

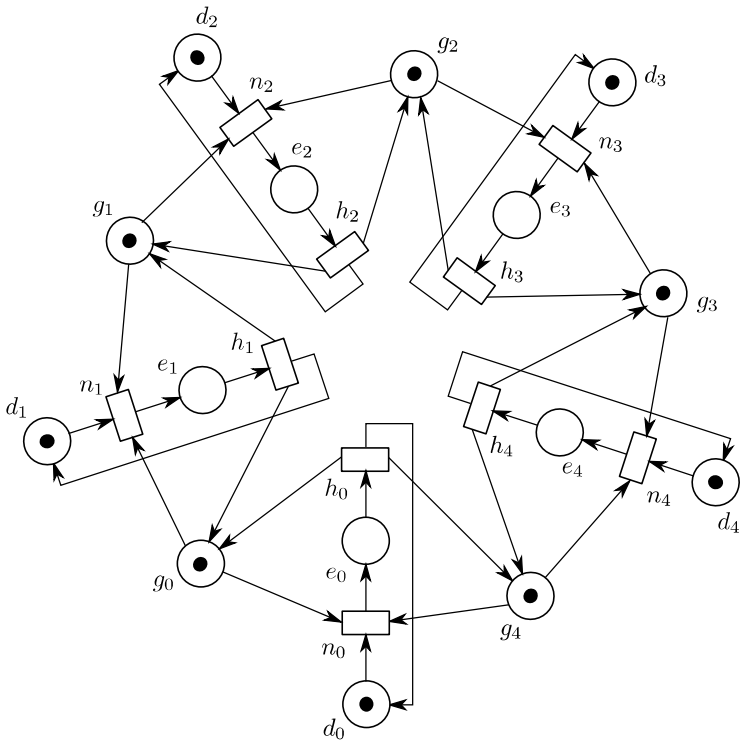


Abbildung 2.6: Dijkstra's Dining Philosophers als S/T-System in Anlehnung an [Bau96]: Fünf Philosophen sitzen an einem Tisch mit fünf Gabeln und können abwechselnd denken und essen. Marken auf d_i repräsentieren dabei denkende und Marken auf e_i essende Philosophen. Mit dem Schalten von n_i beginnt der Philosoph i mit dem Essen, wobei er die beiden Gabeln zu seiner Linken und Rechten benötigt. Nach dem Essen legt er diese mit dem Schalten von h_i wieder zurück.

Schaltregel legt fest, wann und wie Zustandsübergänge durch das Schalten von Transitionen stattfinden können. Eine Transition $t \in T$ heißt genau dann *aktiviert* unter einer Markierung M und kann schalten, wenn $\forall p \in Ft : M(p) \geq W(p,t)$. Wenn eine Transition unter einer Markierung aktiviert ist und schaltet, werden gemäß den jeweiligen Kantengewichten auf den Stellen des Vorbereiches Marken entfernt und auf den Stellen des Nachbereiches Marken erzeugt. Es entsteht eine *Folgemarkierung* M' mit $M'(p) = M(p) - W(p,t) + W(t,p)$ für alle $p \in P$. Abbildung 2.6 zeigt ein Beispiel für ein S/T-System zur Modellierung von Dijkstra's Dining Philosophers.

Das größte Problem bei der Verwendung von S/T-Systemen stellt die Zahl der notwendigen Stellen und Transitionen dar, sobald das zu modellierende System eine gewisse Komplexität überschreitet. Die Modellierung mit S/T-Systemen führt schnell

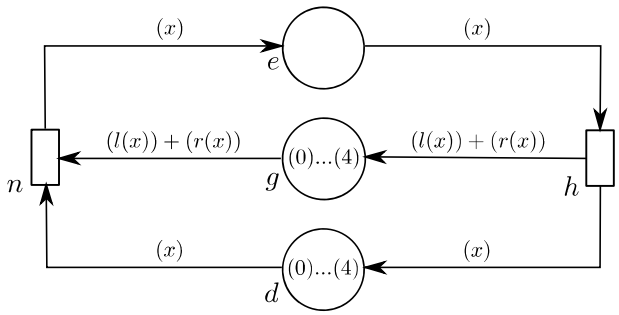


Abbildung 2.7: Dijkstra's Dining Philosophers als Prädikat/Transitions-Netz in Anlehnung an [Sta90]: Die Stellen können mit einstelligen Tupeln von Zahlen zwischen 0 und 4 markiert werden, die die jeweiligen Philosophen repräsentieren. Die Kantengewichte sind Multimengen über einstelligen Tupeln, die aus Termen zusammengesetzt sind. x ist dabei eine Variable und l bzw. r sind Funktionen, die durch $l(x) = x$ und $r(x) = (x + 4) \bmod 5$ gegeben sind.

zu unüberschaubaren Netzstrukturen und der Vorteil der Anschaulichkeit des Modells wird hinfällig. Aus dem Bedarf nach kompakteren Darstellungsmöglichkeiten sind viele Erweiterungen entstanden, die unter dem Begriff *höhere Petri-Netze* zusammengefasst werden. In der Regel wird das Prinzip der anonymen Marken zugunsten von individuellen Marken aufgegeben, in denen zusätzliche Informationen hinterlegt werden können. Zu den verbreitetsten höheren Systemtypen zählen Prädikat/-Transitions-Netze [Gen87, GL81] und Gefärbte Petri-Netze [Jen87, Jen92, Jen94, Jen97]. Abbildung 2.7 zeigt Dijkstra's Dining Philosophers als Prädikat/Transitions-Netz. Es wird deutlich, dass höhere Petri-Netze eine wesentlich kompaktere Darstellung ermöglichen.

Die Analyse von Petri-Netz-Modellen kann über die Struktur oder über das Verhalten eines Netzes stattfinden. Strukturelle Analyseverfahren bestimmen Eigenschaften eines Modells, indem sie ausschließlich den Aufbau des Netzes untersuchen, ohne den Zustandsraum des Modells explizit zu berechnen. Ein klassisches strukturelles Analyseverfahren für Petri-Netze stellt die Invarianten-Analyse dar, die mit den Mitteln der linearen Algebra erfolgt. Für die Invarianten-Analyse wird ein Petri-Netz in eine Matrix aus Stellen und Transitionen übersetzt, in der alle Kantengewichte festgehalten werden (Inzidenzmatrix bzw. Akzidenzmatrix). Mit Hilfe dieser Darstellung lassen sich Gleichungssysteme aufstellen, deren Lösungsvektoren zu so genannten S- und T-Invarianten führen und Aussagen über bestimmte Eigenschaften des

Modells erlauben. Die Invarianten-Analyse hat wie alle strukturellen Analyseverfahren den Vorteil, nicht von der Explosion des Zustandsraumes betroffen zu sein und ist auch in höheren Petri-Netzen möglich.[1] Die Ermittlung bzw. Interpretation von algebraischen Invarianten wird allerdings umso komplizierter, je mehr High-Level-Konzepte der verwendete Systemtyp enthält. Verhaltensorientierte Analyseverfahren sehen stattdessen die möglichst vollständige Berechnung des Zustandsraumes vor. Viele Eigenschaften können dann entschieden werden, indem einfach die gesamte Zustandsmenge auf die gegebene Fragestellung untersucht wird. Werkzeuge wie CPN/Tools können den Zustandsraum eines Netzes sogar automatisch erzeugen. Mit zunehmender Rechnerleistung lassen sich mit solchen Werkzeugen immer komplexere Modelle analysieren. Die vollständige Berechnung aller Zustände und Zustandsübergänge eines Petri-Netzes wird als Erreichbarkeitsanalyse bezeichnet und in Form eines Erreichbarkeitsgraphen dargestellt. Der Erreichbarkeitsgraph repräsentiert den Übergang von einem Netzmodell zu einem Automatenmodell. Die automatisierte Verifikation des Erreichbarkeitsgraphen entspricht dem Prinzip des Model Checkings (siehe Abschnitt 2.3.5).

Bei der Modellierung von Agentensystemen mit Petri-Netzen werden die Agenten und die Umgebung typischerweise von verschiedenen Teilnetzen repräsentiert und anschließend zum Systemmodell zusammengesetzt. Transitionen stellen dabei die Aktionen der Agenten dar, während der Zustand der Agenten und der Zustand der Umgebung mit Stellen modelliert wird. Bei der Verknüpfung der Teilnetze ist darauf zu achten, dass die Verbindung auf der Seite der Agenten nur über Aktionen (Transitionen) erfolgt. Auf diese Weise kann die Kapselung der Agenteninterna gegenüber dem Zugriff von Außen gewährleistet werden. Als Beispiel lassen sich in Abbildung 2.6 die Teilnetze mit den Knoten d_i, n_i, e_i und h_i sowie deren Kanten für jedes $i \in \{0, ..., 4\}$ als Agenten interpretieren. Die Transitionen n_i und h_i bilden ihre sensorischen und effektorischen Schnittstellen. Die Umgebung setzt sich aus den Stellen $g_0, ..., g_4$ zusammen.

Für die Darstellung von Kommunikationsprotokollen eignen sich insbesondere höhere Petri-Netze, in denen strukturierte Marken definiert werden können. Dazu wird ein entsprechender Markentyp als Kreuzprodukt über den jeweiligen Typen der Kommunikationsparameter erklärt. Der Austausch von Nachrichten kann dann über eine zentrale Stelle dieses Typs stattfinden, die als Nachrichtenpool des Systems fun-

[1] Für Prädikat/Transitions-Netze siehe z.B. [Gen87, GL83, GL81, Mev82], für Gefärbte Petri-Netze siehe z.B. [NV86, Jen87, Jen81].

giert. Jeder Agent ist über seine kommunikativen Schnittstellen mit dieser Stelle verbunden. Sendet ein Agent eine Nachricht, so wird ein entsprechendes Tupel in den Nachichtenpool abgelegt, wobei er selbst als Sender definiert ist. Entsprechend entfernt ein Agent genau die Nachrichten aus dem Nachrichtenpool, die ihn als Empfänger definieren. Abbildung 2.8 (oben) zeigt ein Beispiel für die Kommunikation zwischen zwei Agenten über einen zentralen Nachrichtenpool. Eine andere Möglichkeit besteht darin, ein Teilnetz zu integrieren, das als Verteiler das Routing der Nachrichten übernimmt. In diesem Fall senden die Agenten ihre Nachrichten ausschließlich an den Verteiler, dessen interner Mechanismus die Nachrichten an die jeweiligen Empfänger weiterleitet. Diese Variante erinnert stärker an den *Message Transport Service* der FIPA-Spezifikation [Age11b]. Abbildung 2.8 (unten) zeigt ein Beispiel für einen entsprechenden Verteiler. Es ist ebenfalls möglich, die Kommunikationsschnittstellen der jeweiligen Agenten direkt miteinander zu verbinden. Allerdings entstehen in diesem Fall sehr schnell sehr viele Verbindungen und die Struktur des Netzes wird unübersichtlich.

Bei der Darstellung von Interaktionsprotokollen haben Petri-Netze den Vorteil, dass sie problemlos parallele und alternative Abläufe innerhalb der Interaktion abbilden können. Gleichzeitig können sie den Fortschritt der Konversation anhand der jeweils aktuellen Markierung visualisieren. Im einfachsten Fall werden dabei ausschließlich Netze verwendet, die genau eine Quelle (Stelle ohne Vorbereich) und genau eine Senke (Stelle ohne Nachbereich) als Beginn und Ende der Interaktion kennzeichnen. Jeder weitere Knoten im Netz repräsentiert einen Zustand bzw. einen Zustandsübergang der Interaktion und liegt daher auf einem Pfad zwischen Quelle und Senke. Solche Netze entsprechen den Restriktionen, die auch für so genannte Workflow-Petri-Netze gelten [Age79]. Die resultierenden Protokolle können in ein Systemmodell integriert werden, indem die entsprechenden Transitionen des Protokolls mit den Nachrichteneingängen bzw. Nachrichtenausgängen der Agenten gekoppelt werden. Wenn innerhalb des Protokolls individuelle Marken für verschiedene Interaktionspartner verwendet werden, dann lassen sich mit einem einzigen Protokoll auch multiple Konversationen verwalten. Abbildung 2.9 zeigt eine Darstellung des FIPA-Request-Protokolls in Form eines Petri-Netzes.

Neben Graphtransformationen stellen Petri-Netze einen viel versprechenden Ausgangspunkt dar, um Agenten formal und anschaulich zu beschreiben. Es gibt viele verschiedene Arbeiten, die zeigen, wie Petri-Netze (insbesondere höhere Petri-Netze) konkret zur Modellierung von Agentensystemen eingesetzt werden können. Eine Auswahl ist wie folgt gegeben:

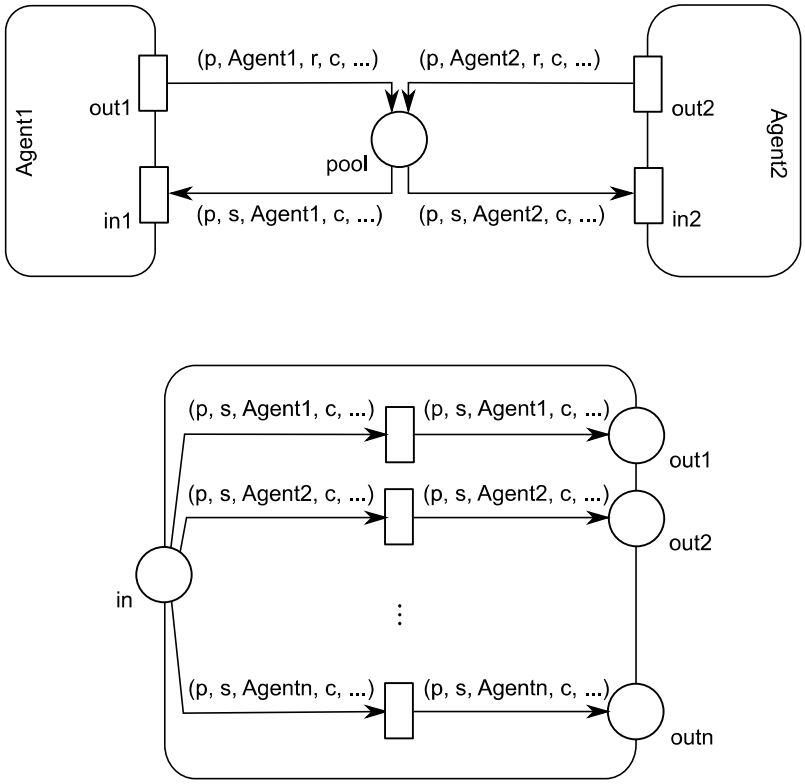


Abbildung 2.8: Oben: Kommunikation zwischen zwei Agenten über einen zentralen Nachrichtenpool. Die Variablen p, s, r und c repräsentieren Performative, Sender, Empfänger und Inhalt der Nachricht. Unten: Darstellung eines zentralen Verteilers.

- **Klassische Petri-Netze:**
  In [Fer01] werden S/T-Systeme mit einer erweiterten Syntax vorgestellt, um Agentensysteme zu modellieren. Agenten und Umgebung werden dabei von einzelnen Petri-Netz-Komponenten repräsentiert, deren sensorische und effektorische Verknüpfung über gesonderte Schnittstellen (Input- und Output-Pole) stattfindet. Auf semantischer Ebene werden diese Schnittstellen durch einfache Stellen und Transitionen ersetzt. Abbildung 2.10 zeigt eine entsprechende Petri-Netz-Komponente, die einen Philosophen modelliert.

- **Höhere Petri-Netze:**
  In [XVIY03, XVIY02] und [XD00] werden Prädikat/Transitions-Netze für die Modellierung von Agentensystemen vorgeschlagen. In [Hol95], [MP99] und [FB98] werden Gefärbte Petri-Netze verwendet. Die Arbeiten stellen entspre-

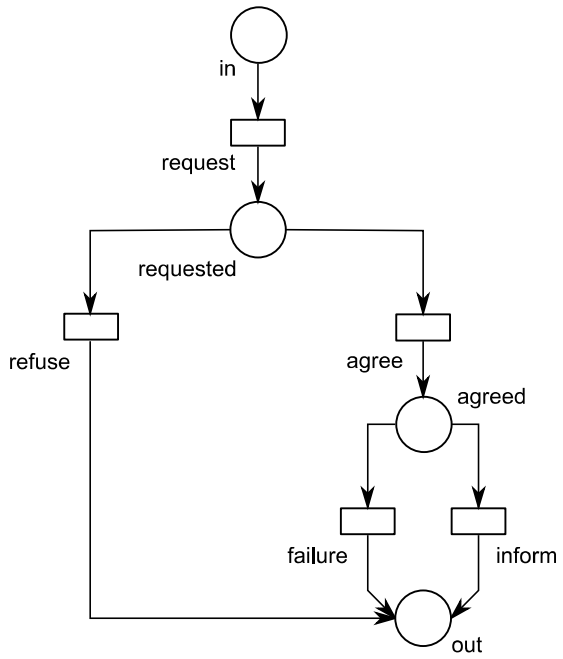


Abbildung 2.9: Darstellung des FIPA-Request-Protokolls in Form eines Petri-Netzes: Die Interaktion beginnt mit dem Ablegen einer Marke auf die Quelle *in* und endet mit dem Erreichen der Senke *out*.

chende Anwendungsbeispiele vor, führen mit Ausnahme von [Hol95] aber keine neuen Sprachelemente ein. Sie unterscheiden sich insbesondere in ihrem Detaillierungsgrad. [CCF+00] und [NPC01] benutzen Gefärbte Petri-Netze zur Modellierung von Interaktionsprotokollen.

- **Objektorientierte Petri-Netze:**
  In [XS03, XS01], [CHSB96] und [MK96, MK99] werden verschiedene objektorientierte Erweiterungen von Petri-Netzen für die Modellierung von Agentensystemen vorgeschlagen und entsprechende Anwendungsbeispiele vorgestellt. Der Schwerpunkt dieser Ansätze liegt vor allem in der Modellierung und Komposition von lose gekoppelten, objektorientierten Komponenten mit einheitlichen Schnittstellen.

- **Referenz-Netze:**
  In [KMR01, LM04, KMR03] werden Agenten mit Referenz-Netzen modelliert. Die Arbeiten gehen auf [MW97] zurück. Referenz-Netze [Kum02] basieren auf Gefärbten Petri-Netzen und vereinen verschiedene High-Level-Konzepte wie

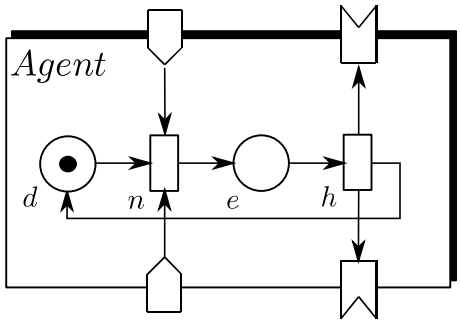


Abbildung 2.10: Modellierung eines Philosophen als Petri-Netz-Komponente nach [Fer01].

das Netze-in-Netzen-Konzept [Val98] und die Synchronisation von Transitionen [CH94]. Auf diese Weise kann die Modellierung auf unterschiedlichen Detaillierungsgraden stattfinden wie Abbildung 2.11 zeigt. Auch die Modellierung von Interaktionsprotokollen der Agent UML und ihre Integration ist möglich [CM04, CMR03].

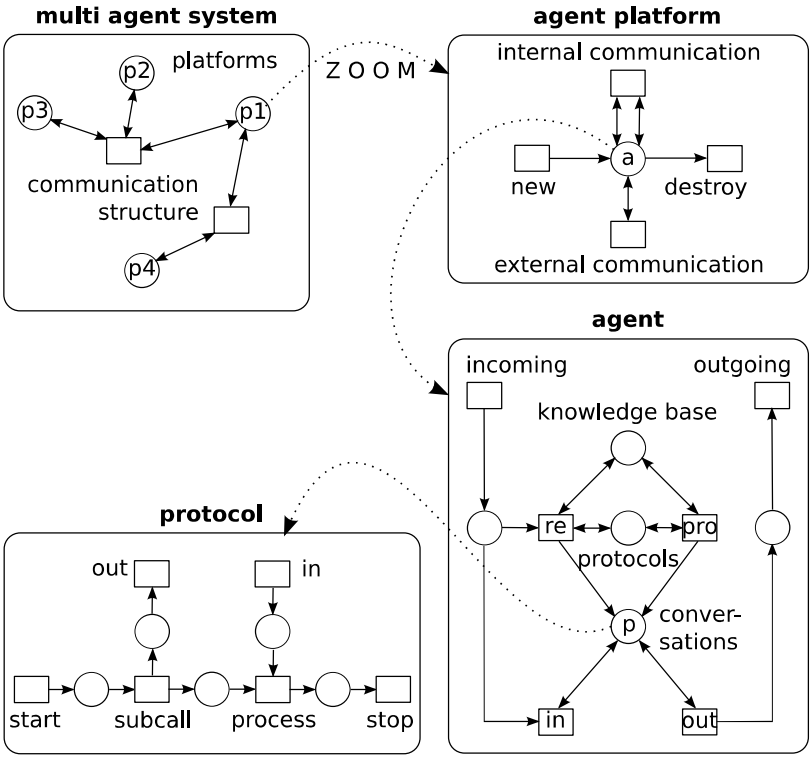


Abbildung 2.11: Architektur für die Modellierung von Agentensystemen mit Referenz-Netzen nach [LM04].

3

Modellierung von Zielen

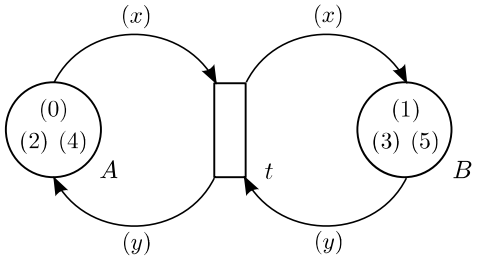
In der Literatur existieren zahlreiche Ansätze, die die Modellierung von Agentensystemen mit Petri-Netzen propagieren. Die Popularität von Petri-Netzen in der agentenorientierten Software-Entwicklung basiert auf mehreren Vorteilen. Zu diesen Vorteilen zählen die formale Syntax und Semantik, die anschauliche graphische Repräsentation, die strukturierte und kompakte Darstellung, die Berücksichtigung von Nebenläufigkeit und Nichtdeterminismus und die Verbreitung von Petri-Netzen in der Literatur. Es bleibt allerdings unklar, wie dynamische Ziele von Agenten in Petri-Netzen repräsentiert werden können. Ziele beschreiben die gewünschte Systemzustände eines Agenten und sind gleichzeitig als Bestandteil seines mentalen Zustands zu verstehen. Sie müssen als dynamische Elemente auf der Modell-Ebene abgebildet werden, weil sie einerseits das Modellverhalten beeinflussen und anderseits durch das Modellverhalten beeinflusst werden. Da Zustände in Petri-Netzen von Markierungen repräsentiert werden, erscheint es für die Modellierung von dynamischen Zielen in Agentensystemen sinnvoll, Beschreibungen von Markierungen als Bestandteile von Markierungen zuzulassen. In den folgenden Abschnitten wird mit der Einführung von Ziel/Transitions-Netzen (Z/T-Netze) eine verbreitete höhere Petri-Netz-Art um diese Fähigkeit erweitert. Ausgangspunkt stellen Prädikat/Transitions-Netze dar.

3.1 Grundidee

Prädikat/Transitions-Netze sind höhere Petri-Netze, in denen die Netz-Elemente mit algebraischen Ausdrücken beschriftet werden können (für eine formale Einführung siehe [Gen87]). Auf diese Weise lassen sich viele Zusammenhänge sehr strukturiert und kompakt beschreiben, so dass die Zahl der notwendigen Stellen und Transitionen begrenzt werden kann. Grundlage ist eine Signatur sowie eine entsprechende Struktur, die verschiedene Trägermengen, Operationen und Prädikate für das Netz erklären. Aus den Trägermengen werden Tupel gebildet, die als Marken und als Kantengewichte im Netz dienen. Abbildung 2.7 zeigt ein einfaches Beispiel für ein Prädikat/Transitions-Netz. Darüber hinaus ist es möglich, das Schalten von Transitionen durch *Wächter* einzuschränken. Ein Wächter ist eine Formel, die aus den Operations- und Prädikatensymbolen der Signatur gebildet wird und zusätzliche Restriktionen für die Aktivierung einer Transition beschreibt. Um eine Transition zu aktivieren, muss ihr Wächter für die gegebene Variablenbelegung erfüllt sein.

Zwischen Prädikat/Transitions-Netzen und logischen Formeln existiert ein wichtiger Zusammenhang. Wenn die Stellen eines Netzes als *variable Prädikate* interpretiert werden, dann lassen sich mit Prädikat/Transitions-Netzen, die tote Transitionen (Fakten) enthalten, logische Formeln repräsentieren [GL81, GTM76]. Der Übergang von klassischen Petri-Netzen zu Prädikat/Transitions-Netzen entspricht dabei dem Übergang von der Aussagenlogik zur Prädikatenlogik erster Ordnung. Die Extension eines variablen Prädikates ist eindeutig durch die Markierung der entsprechenden Stelle festgelegt: Das Prädikat ist genau dann erfüllt, wenn die Stelle ein Tupel mit den Argumenten des Prädikates enthält. Umgekehrt lassen sich mit logischen Formeln, die variable Prädikate enthalten, auch Markierungen von Netzen repräsentieren. Die Grundidee besteht nun darin, den Zusammenhang von Stellen und Prädikaten bzw. Markierungen und Formeln in die Modellebene abzubilden, um Zustände des Modells in den Zuständen des Modells beschreiben zu können.

Abbildung 3.1 zeigt ein Beispiel für ein einfaches Ziel/Transitions-Netz. Wenn der in eckigen Klammern dargestellte Wächter vernachlässigt wird, handelt es sich bei diesem Netz um ein klassisches Prädikat/Transitions-Netz mit zwei Stellen A und B sowie einer Transition t. In der Anfangsmarkierung seien die Stellen A und B mit beliebigen Mengen von (einstelligen Tupeln von) natürlichen Zahlen markiert. Mit dem Schalten der Transition t wird eine Marke auf A mit einer Marke auf B vertauscht. Aufgrund des inhärenten Nichtdeterminismus der Schaltregel ist dabei



$[\neg\forall x : \forall y : A(x) \wedge B(y) \rightarrow x \leq y]$

Abbildung 3.1: Ein Ziel/Transitions-Netz mit einem Wächter für die Transition t.

nicht festgelegt, um welche Zahlen es sich handelt, weil verschiedene Belegungen für die Variablen x und y möglich sind. Mit dieser Konstruktion lässt sich ein sehr einfaches Sortierverfahren formulieren, das solange den Tausch zweier beliebiger Zahlen zwischen A und B vorsieht, bis alle Zahlen auf A kleiner oder gleich den Zahlen auf B sind. Um dieses Verhalten zu realisieren, muss jedoch ein Kontrollmechanismus integriert werden, der eine Aktivierung von t nach der Sortierung verhindert. Von einer beliebigen Markierung ausgehend, macht die Formulierung eines solchen Kontrollmechanismus die Einführung weiterer Stellen und Transitionen erforderlich, mit denen zusätzliche Informationen über die Beschaffenheit der Markierung ermittelt werden können. Der Versuch, einen geeigneten Wächter für t zu formulieren, scheitert in Prädikat/Transitions-Netzen an den vorhandenen Sprachmitteln. Es kann keine geeignete Formel formuliert werden, die den gewünschten Zielzustand des Netzes beschreibt. Es ist zwar möglich, mit einem Wächter der Form $x > y$ den Zielzustand zu erzwingen, allerdings können in diesem Fall nicht mehr beliebige Marken zwischen A und B während der Ausführung getauscht werden. Der Wächter schränkt also das Verhalten des Systems zu sehr ein. Soll die Möglichkeit bestehen, kleinere Zahlen auf A mit größeren Zahlen auf B zu tauschen, solange die Sortierung noch nicht abgeschlossen ist, muss ein Kontrollmechanismus eingeführt werden, der ausschließlich die Beschaffenheit der Markierung berücksichtigt und nicht die Bindung der Variablen vorschreibt. Wenn für die Bildung von Formeln auch variable Prädikate zugelassen werden, dann lässt sich mit der dargestellten Formel in Abbildung 3.1 das Gewünschte erreichen: t kann solange beliebige Zahlen zwischen A und B tauschen, bis alle Zahlen auf A kleiner oder gleich den Zahlen auf B sind. Sobald die Sortierung abgeschlossen ist, bleibt t inaktiv, weil der Wächter von t nicht erfüllt ist. Die Aktivierung von t hängt also von der Zusammensetzung der Marken auf A und B ab, wobei es sich bei $A(x)$ und $B(y)$ um variable Prädikate handelt, die anhand

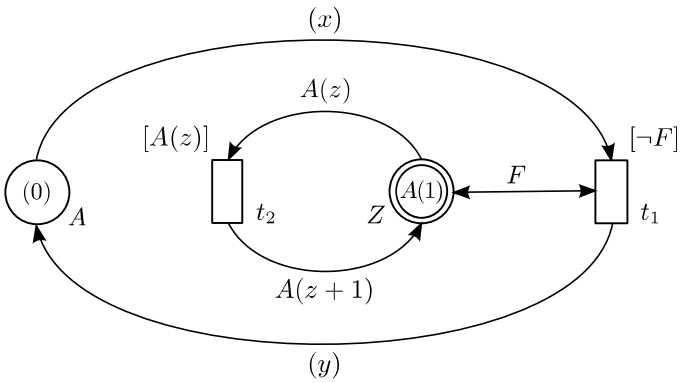


Abbildung 3.2: Ein Ziel/Transitions-Netz mit einer Ziel-Stelle Z für dynamische Ziele.

der aktuellen Markierung ausgewertet werden. Der Wächter von t repräsentiert in diesem Beispiel die Negation eines gewünschten Zielzustands. Ein Fall, in dem eine Transition nur dann aktiviert werden kann, wenn ein Ziel erfüllt ist, ist genauso denkbar.

Formeln, die variable Prädikate enthalten, sind auch als Marken im Netz denkbar. Voraussetzung ist ein gesonderter Typ von Stellen, der nicht mit Tupeln, sondern mit Formeln markiert wird. Abbildung 3.2 zeigt ein Beispiel. In der Anfangsmarkierung sei die Stelle A mit dem Tupel (0) markiert. Mit dem Schalten der Transition t_1 wird die entsprechende Marke entfernt und eine neue Marke auf A erzeugt. Das dargestellte Netz setzt keine Bindung der Variablen im Nachbereich einer Transition durch den Vorbereich voraus, so dass beim Schalten von t_1 die Variable y mit einer beliebigen natürlichen Zahl belegt werden kann. Die durch einen doppelten Rand hervorgehobene Stelle Z kann mit Formeln, d.h. mit Zielen, markiert werden. Für diese Stelle sei die Anfangsmarkierung durch die Formel $A(1)$ gegeben, d.h. das Ziel besteht zu Beginn darin, das Tupel (1) auf der Stelle A zu erzeugen. A stellt dabei ein variables Prädikat dar, das mit der Stelle A assoziiert wird. Die Transition t_1 ist durch eine Schleife mit der Stelle Z verbunden. Über eine Formelvariable F, die mit beliebigen Formeln belegt werden kann, erhält t_1 Zugriff auf das aktuelle Ziel und verwendet es gleichzeitig im Wächter als Negation. Auf diese Weise kann t_1 nur dann aktiviert werden, wenn das aktuelle Ziel nicht erfüllt ist, d.h. wenn eine entsprechende Marke nicht auf A vorhanden ist. Tritt durch fortlaufendes Schalten schließlich der Fall ein, dass auf A tatsächlich die auf Z geforderte Marke vorhanden ist, bleibt t_1 inaktiv. Stattdessen ist in diesem Fall die Transition t_2 aktiviert, weil

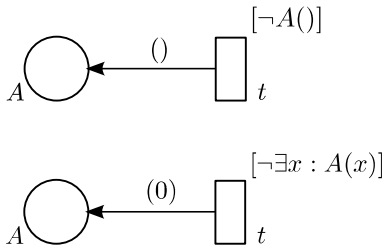


Abbildung 3.3: Oben: Ein Ziel/Transitions-Netz mit einem Nulltest für das 0-stellige Prädikat A. Unten: Nulltest eines 1-stelligen Prädikates.

nur dann das variable Prädikat $A(z)$ erfüllt ist und die Formel im Wächter von t_2 gilt. Die Aktivierungen von t_2 hängt also ebenfalls von der Markierung auf A ab, obwohl keine direkte Verbindung zwischen t_2 und A existiert. Sobald t_2 schaltet, wird das aktuelle Ziel von Z entfernt und ein neues Ziel (der Nachfolger von z) auf der Ziel-Stelle erzeugt. Daraufhin kann t_1 wieder fortlaufend schalten, bis die gewünschte Marke auf A vorliegt. In diesem Beispiel können die Transitionen t_1 und t_2 als Aktionen eines Agenten interpretiert werden, während die Stelle Z seinen mentalen Zustand die Stelle A seine Umgebung repräsentiert.

Die Verwendung von variablen Prädikaten in Ziel/Transitions-Netzen wirkt sich auch auf die Ausdrucksmöglichkeiten aus. Ein häufiges Problem bei der Modellierung mit Petri-Netzen ist der Wunsch, die Aktivierung einer Transition mit einem so genannten Nulltest zu koppeln. Dabei soll eine Transition nur dann schalten können, wenn eine oder mehrere Stellen *sauber* sind, d.h. keine Marken enthalten. In beschränkten Netzen kann ein solcher Nulltest durch die Einführung von *Komplementstellen* erfolgen (siehe z.B. [PW03, Sta90]), aber für unbeschränkte Netze ist dies im Allgemeinen nicht möglich. In Ziel/Transitions-Netzen kann ein solcher Nulltest sehr einfach formuliert werden. Abbildung 3.3 zeigt zwei Beispiele.

Die Beispiele in diesem Abschnitt zeigen, wie sich Systemzustände in Ziel/Transitions-Netzen mit Formeln beschreiben lassen und wie entsprechende Formeln auf der Modellebene mit dem Modellverhalten gekoppelt werden können. Auf diese Weise können dynamische Ziele eines Agenten modelliert werden. Dazu wird jedes Ziel von einer Formel repräsentiert, die als Marke auf einer Ziel-Stelle hinterlegt wird. Durch den üblichen Konsum und die Produktion von Marken beim Schalten des Netzes, können Ziele erstellt, gelöscht und geändert werden. Durch ihre Verwendung in

Wächtern von Transitionen können sie gleichzeitig auf das Modellverhalten Einfluss nehmen. In den folgenden Abschnitten werden diese Ideen formal eingeführt.

3.2 Algebraische Grundlagen

Dieser Abschnitt enthält eine Zusammenfassung der algebraischen Grundlagen, die für eine formale Definition von Ziel/Transitions-Netzen notwendig sind.

$\mathbb{N} = \{0, 1, 2, ...\}$ bezeichne die Menge der natürlichen Zahlen, $\mathbb{N}_+ = \mathbb{N}\backslash\{0\}$ die Menge der natürlichen Zahlen ohne 0 und $\mathbb{B} = \{\top, \bot\}$ die Menge der Wahrheitswerte. Seien A und B beliebige Mengen. Für eine bijektive Funktion $f : A \to B$ bezeichne $f^{-1} : B \to A$ die Umkehrfunktion von f mit $f^{-1}(f(a)) = a$ und $f(f^{-1}(a)) = a$ für alle $a \in A$. Für eine Relation $R \subseteq A \times B$ und $a \in A$ und $b \in B$ ist $aR = \{b \mid (a, b) \in R\}$ und $Rb = \{a \mid (a, b) \in R\}$. Statt $(a, b) \in R$ wird auch aRb geschrieben. Mit R^+ sei die transitive und mit R^* die reflexiv-transitive Hülle von R bezeichnet. Ein Wort $a_1...a_n$ über A bzw. ein Tupel $(a_1, ..., a_n)$ von A ist eine endliche geordnete Folge von Elementen aus A, so dass $a_i \in A$ für alle $i \in \{1..n\}$. λ bezeichne das leere Wort, A^* bezeichne die Menge aller Wörter über A und $A^+ = A \setminus \{\lambda\}$ die Menge aller Wörter über A ohne das leere Wort λ. Für eine geordnete Menge A bezeichne $max(A)$ bzw. $min(A)$ das größte bzw. kleinste Element aus A. Für eine beliebige Menge I bezeichne $(A_i)_{i\in I}$ eine Familie von Mengen, so dass A_i für alle $i \in I$ eine Menge ist und $A_i \cap A_j = \emptyset$ für alle $i, j \in I$ mit $i = j$. Für eine Familie $A = (A_i)_{i\in I}$ bezeichne A ebenso die Vereinigung $\bigcup_{i\in I} A_i$.

Signaturen und Strukturen

In Ziel/Transitions-Netzen werden die Netz-Elemente mit Ausdrücken der Prädikatenlogik erster Orndung (mit Identität) beschriftet. Grundlage bildet eine Signatur und eine entsprechende Struktur, auf denen die syntaktischen und semantischen Begriffe erklärt sind.

Eine Signatur $\Sigma = (S, \Omega, \Pi)$ besteht aus einer endlichen Menge S von Sorten, einer Familie $\Omega = (\Omega_w)_{w\in S^+}$ von Operationssymbolen und einer Familie $\Pi = (\Pi_w)_{w\in S^*}$ von Prädikatensymbolen. Eine Σ-Struktur A_Σ für eine Signatur Σ besteht aus einer Menge A_s für jede Sorte $s \in S$, deren Elemente Individuen genannt werden, einer Funktion $f_\omega : A_{s_1} \times ... \times A_{s_n} \to A_{s_{n+1}}$ für jedes Operationssymbol $\omega \in \Omega_{s_1...s_n s_{n+1}}$

und einer Relation $r_\pi : A_{s_1} \times ... \times A_{s_n}$ für jedes Prädikatensymbol $\pi \in \Pi_{s_1...s_n}$. Ein Operationssymbol $\omega \in \Omega_s$ mit $s \in S$ heißt Konstantensymbol der Sorte s, die entsprechende Funktion f_ω heißt Konstante der Sorte s.

Die Signatur ist syntaktischer Natur. Sie legt Bezeichner für Sorten, Operationen und Prädikate fest und bestimmt deren Stelligkeit. Auf ihr lassen sich syntaktische Begriffe wie Terme und Formeln erklären, die als Ausdrücke in Ziel/Transitions-Netzen verwendet werden. Die Struktur ist semantischer Natur und ordnet den Bezeichnern in der Signatur konkrete Mengen, Funktionen und Relationen zu. Auf diese Weise können die entsprechenden Ausdrücke eindeutig ausgewertet werden. Aus Gründen der Flexibilität werden bei der Modellierung von Ziel/Transitions-Netzen mehrsortige Signaturen und Strukturen berücksichtigt.

Im Folgenden sei o.B.d.A. vorausgesetzt, dass für jedes Individuum $a \in A_s$ einer Sorte $s \in S$ genau ein Konstantensymbol $\omega_a \in \Omega_s$ mit $f_{\omega_a} = a$ existiert, d.h. jedes Individuum kann syntaktisch durch genau ein Konstantensymbol der Signatur und semantisch durch genau eine Konstante der Struktur repräsentiert werden. Darüber hinaus wird für die Beispiele von Ziel/Transitions-Netzen in dieser Arbeit die Signatur und Struktur der natürlichen Zahlen mit den gängigen Operationen und Prädikaten vorausgesetzt (siehe z.B. [Kre91]).

Terme und Formeln

Die syntaktischen Regeln zur Bildung von Ausdrücken in Ziel/Transitions-Netzen sind durch Terme und Formeln gegeben. Zu diesem Zweck wird für eine Signatur $\Sigma = (S, \Omega, \Pi)$ eine Familie $X = (X_s)_{s \in S}$ von Variablen mit einer Variablenmenge X_s für jede Sorte $s \in S$ vorausgesetzt.

(1.) Die Menge der Terme einer Sorte $s \in S$ ist wie folgt gegeben:

- (1.1.) Jede Variable $x \in X_s$ ist ein Term der Sorte s.
- (1.2.) $\omega(t_1, ..., t_n)$ ist ein Term der Sorte s genau dann, wenn $\omega \in \Omega_{s_1...s_n s}$ ein Operationssymbol ist und t_i für alle $i \in \{1..n\}$ ein Term der Sorte s_i ist.

(2.) Die Menge der Formeln ist wie folgt gegeben:

- (2.1.) $(t_1 = t_2)$ ist eine Formel, genau dann wenn t_1 und t_2 Terme sind.

(2.2.) $\pi(t_1, ..., t_n)$ ist eine Formel genau dann, wenn $\pi \in \Pi_{s_1 ... s_n}$ ein Prädikatensymbol ist und t_i für alle $i \in \{1..n\}$ ein Term der Sorte s_i ist.

(2.3.) $(\neg F_1)$, $(F_1 \wedge F_2)$, $(F_1 \vee F_2)$, $(F_1 \rightarrow F_2)$, $(F_1 \leftrightarrow F_2)$, $(\exists x : F_1)$ und $(\forall x : F_1)$ sind Formeln genau dann, wenn F_1 und F_2 Formeln sind.

Jedes Vorkommen einer Variablen x in einem Term t heißt frei. Ein Vorkommen von x in einer Formel F heißt genau dann frei, wenn x in keiner Teilformel von F der Form $\forall x : G$ oder $\exists x : G$ vorkommt. Ein Ausdruck (Term oder Formel) heißt geschlossen, wenn keine freien Variablen darin vorkommen, sonst offen.

Im Folgenden wird auf Klammern in Termen und Formeln verzicht, wo keine Mehrdeutigkeiten entstehen. Darüber hinaus wird neben der Präfix-Notation auch die Infix-Notation vrewendet, wenn sie die Lesbarkeit untersützt, z.B. in mathematischen Ausdrücken.

Auswertung und Vereinfachung

Die Interpretation eines Ausdrucks hängt von der gewählten Struktur ab und ist im Folgenden nur für geschlossene Ausdrücke erklärt. Die freien Variablen in offenen Termen und Formeln müssen somit zunächst geeignet substituiert werden, bevor die Ausdrücke ausgewertet werden können.

Sei $\Sigma = (S, \Omega, \Pi)$ eine Signatur, $\{x_1, ...x_n\} \subseteq X$ eine Menge von Variablen und $\{c_1, ...c_n\} \subseteq \Omega$ eine Menge von Konstantensymbolen. Eine Abbildung $\alpha = (x_1 \rightarrow c_1, ..., x_n \rightarrow c_n)$, die jede Variable x_i auf ein Konstantensymbol c_i der selben Sorte abbildet, heißt Belegung. Für einen beliebigen Ausdruck e bezeichne $e : (x_1 \rightarrow c_1, ..., x_n \rightarrow c_n)$ den Ausdruck, der durch die Ersetzung jeder freien Variable x_i in e durch c_i entsteht.

Die Auswertung ε von Ausdrücken ist eine Abbildung, die Terme auf Individuen und Formeln auf Wahrheitswerte abbildet.

(1.) Für einen geschlossenen Term $t = \omega(t_1, ..., t_n)$ mit $\omega \in \Omega$ ist $\varepsilon(t) = f_\omega(\varepsilon(t_1), ..., \varepsilon(t_n))$.

(2.) Für eine geschlossene Formel F ist $\varepsilon(F)$ wie folgt festgelegt:

(2.1.) Für $F = (t_1 = t_2)$ ist $\varepsilon(F) = \top$ genau dann, wenn $\varepsilon(t_1) = \varepsilon(t_2)$.

(2.2.) Für $F = \pi(t_1, ..., t_n)$ mit $\pi \in \Pi$ ist $\varepsilon(F) = \top$ genau dann, wenn $(\varepsilon(t_1), ..., \varepsilon(t_n)) \in r_\pi$.

(2.3.) Für $F = \neg F_1$ ist $\varepsilon(F) = \top$ genau dann, wenn $\varepsilon(F_1) = \bot$.

(2.4.) Für $F = F_1 \wedge F_2$ ist $\varepsilon(F) = \top$ genau dann, wenn $\varepsilon(F_1) = \varepsilon(F_2) = \top$.

(2.5.) Für $F = F_1 \vee F_2$ ist $\varepsilon(F) = \top$ genau dann, wenn $\varepsilon(\neg((\neg F_1) \wedge (\neg F_2))) = \top$.

(2.6.) Für $F = F_1 \rightarrow F_2$ ist $\varepsilon(F) = \top$ genau dann, wenn $\varepsilon(\neg F_1 \vee F_2) = \top$.

(2.7.) Für $F = F_1 \leftrightarrow F_2$ ist $\varepsilon(F) = \top$ genau dann, wenn $\varepsilon(F_1 \rightarrow F_2) = \varepsilon(F_2 \rightarrow F_1) = \top$.

(2.8.) Für $F = \exists x : F_1$ mit $x \in X_s$ und $s \in S$ ist $\varepsilon(F) = \top$ genau dann, wenn ein Konstantensymbol a der Sorte s existiert, so dass $\varepsilon(F_1 : (x \rightarrow a)) = \top$.

(2.9.) Für $F = \forall x : F_1$ mit $x \in X_s$ und $s \in S$ ist $\varepsilon(F) = \top$ genau dann, wenn $\varepsilon(\exists x : \neg F_1) = \top$.

Neben der Auswertung von Ausdrücken wird auch der Begriff der Vereinfachung in Ziel/Transitions-Netze benötigt. Die Vereinfachung von Ausdrücken bezeichnet die syntaktische Ersetzung von Termen durch semantisch äquivalente Konstantensymbole.

Sei t ein geschlossener Term und F eine geschlossene Formel. Die Vereinfachung von t ist gegeben durch $\sigma(t) = c$ für ein Konstantensymbol c mit $\varepsilon(t) = \varepsilon(c)$. Mit $\sigma(F)$ sei die Vereinfachung der Formel F bezeichnet, in der alle Terme vereinfacht wurden.

Da vorausgesetzt wird, dass zu jedem Individuum genau ein Konstantensymbol existiert, ist die Vereinfachung von Ausdrücken für alle Terme und Formeln eindeutig.

Multimengen

Ein weiteres wichtiges Konzept in Ziel/Transitions-Netzen sind Multimengen. Sie werden als Kantengewichte und als Markierungen verwendet. Multimengen sind eine Verallgemeinerung von Mengen, in denen das mehrfache Vorkommen eines Elementes möglich ist, d.h. jedem Element wird eine Häufigkeit in Form einer natürlichen Zahl zugeordnet. Über die Häufigkeit der Elemente lassen sich Addition, Subtraktion und Multiplikation, sowie einige Prädikate auf Multimengen erklären.

Sei A eine beliebige nichtleere Menge. Eine Multimenge über A ist eine Abbildung $m : A \to \mathbb{N}$, die jedem Element aus A eine Häufigkeit zuordnet. $\mathbb{N}^A$ bezeichne die Menge aller Multimengen über A. Für ein $a \in A$ gilt $a \in m$ genau dann, wenn $m(a) > 0$. m heißt endlich genau dann, wenn $m(a) > 0$ für endlich viele $a \in A$ gilt. Darüber hinaus bezeichnet 0 die leere Multimenge mit $0(a) = 0$ für alle $a \in A$. Für zwei Multimengen $m_1, m_2 \in \mathbb{N}^A$ gilt $m_2 \leq m_1$ genau dann, wenn $m_2(a) \leq m_1(a)$ für alle $a \in A$. Die Addition ist gegeben durch $(m_1 + m_2)(a) = m_1(a) + m_2(a)$ für alle $a \in A$. Für $m_2 \leq m_1$ ergibt sich die Subtraktion zu $(m_1 - m_2)(a) = m_1(a) - m_2(a)$ für alle $a \in A$. Die Multiplikation einer natürlichen Zahl $n \in \mathbb{N}$ mit einer Multimenge $m \in \mathbb{N}^A$ ist gegeben durch $(n \cdot m)(a) = n \cdot m(a)$ für alle $a \in A$. Eine Folge über A heißt schließlich Folge von m genau dann, wenn sie jedes $a \in A$ genau $m(a)$-mal enthält.

Für die Schreibweise von Multimengen wird die Mengen- oder die Produktsummennotation verwendet. So ist für die Multimenge $m = (a_1 \to 2, a_2 \to 1, a_3 \to 0, a_4 \to 2)$ über der Menge $A = \{a_1, a_2, a_3, a_4\}$ sowohl $\{a_1, a_1, a_2, a_4, a_4\}$ als auch $2a_1 + a_2 + 2a_4$ zulässig. Eine Funktion f, die auf den Elementen einer Menge A definiert ist, wird bei Bedarf auf Tupel, Mengen und Multimengen über A erweitert, indem f auf jedes Element des Tupels, der Menge oder der Multimenge angewandt wird.

Erweiterte Signatur

Um den Zusammenhang zwischen Markierungen und logischen Formeln auf der Modellebene von Ziel/Transitions-Netzen abbilden zu können, muss die Signatur Σ um eine Familie von *variablen* Prädikatensymbolen erweitert werden. Im Gegensatz zu den in Σ definierten Prädikatensymbolen, deren Extension durch eine Relation in einer Σ-Struktur festgelegt wird, werden variable Prädikaten anhand von Markie-

rungen ausgewertet. Die Extension eines variablen Prädikates kann also in verschiedenen Zuständen eines Ziel/Transitions-Netzes variieren (siehe Definition 3.2).

Sei $\Sigma = (S, \Omega, \Pi)$ eine Signatur und $X = (X_s)_{s \in S}$ eine Familie von Variablen für Σ. Die erweiterte Signatur $\widehat{\Sigma} = (\Sigma, \widehat{\Pi})$ von Σ besteht zusätzlich aus einer Familie $\widehat{\Pi} = (\widehat{\Pi}_a)_{a \in S^*}$ von variablen Prädikatensymbolen mit $\widehat{\Pi} \cap \Pi = \emptyset$. Darüber hinaus wird eine Menge von Formelvariablen $\widehat{X}$ mit $\widehat{X} \cap X = \emptyset$ vorausgesetzt. Eine erweiterte Formel ist eine Formel, in der variable Prädikatensymbole aus $\widehat{\Pi}$ und Formelvariablen aus $\widehat{X}$ als Teilformeln enthalten sein können. Da eine Formelvariable nicht durch Quantoren gebunden werden kann, heißt jedes Vorkommen in einer erweiterten Formel frei. Für ein variables Prädikat $\widehat{\pi} \in \widehat{\Pi}$ und eine erweiterte Formel $\widehat{F}$ gilt $\widehat{\pi} \in \widehat{F}$ genau dann, wenn $\widehat{\pi}$ in $\widehat{F}$ enthalten ist.

3.3 Ziel/Transitions-Netze

Definition 3.1 (Ziel/Transitions-Netz)
Sei $\widehat{\Sigma} = (\Sigma, \widehat{\Pi})$ mit $\Sigma = (S, \Omega, \Pi)$ eine erweiterte Signatur und A_Σ eine Σ-Struktur. Ein Ziel/Transitions-Netz *$ZT = (N, A_N, M_0)$ besteht aus folgenden Bestandteilen:*

(1.) $N = (P_N, \widehat{P}_N, T_N, F_N)$ *ist ein* Netz *mit:*

- *(1.1.)* P_N *ist eine endliche Menge von* Prädikat-Stellen.
- *(1.2.)* $\widehat{P}_N$ *ist eine endliche Menge von* Ziel-Stellen, *so dass* $P_N \cap \widehat{P}_N = \emptyset$.
- *(1.3.)* T_N *ist eine endliche Menge von* Transitionen, *so dass* $T_N \cap (P_N \cup \widehat{P}_N) = \emptyset$.
- *(1.4.)* $F_N \subseteq ((P_N \cup \widehat{P}_N) \times T_N) \cup (T_N \times (P_N \cup \widehat{P}_N))$ *ist eine Menge von* Kanten.

Der Vorbereich einer Stelle bzw. einer Transition $x \in P_N \cup T_N$ ist durch $F_N x = \{y \mid (y, x) \in F_N\}$, der Nachbereich durch $xF_N = \{y \mid (x, y) \in F_N\}$ gegeben.

(2.) $A_N = (A_P, A_T, A_F)$ *ist eine* Beschriftung *mit:*

- *(2.1.)* A_P *ist eine bijektive Abbildung, die jeder Prädikat-Stelle $p \in P_N$ ein variables Prädikatensymbol $\widehat{\pi} \in \widehat{\Pi}$ zuordnet.*
- *(2.2.)* A_T *ist eine Abbildung, die jeder Transition $t \in T_N$ eine erweiterte Formel, genannt* Wächter, *zuordnet.*

(2.3.) *A_F ist eine Abbildung, die jeder Kante $(x, y) \in F$ mit $x = p$ oder $y = p$ für ein $p \in P_N$ mit $A_P(p) \in \widehat{\Pi}_{s_1,...,s_n}$ eine endliche, nichtleere Multimenge über Tupel von Termen der Sorte $s_1, ..., s_n$, und jeder Kante $(x, y) \in F$ mit $x = \widehat{p}$ oder $y = \widehat{p}$ für ein $\widehat{p} \in \widehat{P}_N$ eine endliche, nichtleere Multimenge über erweiterten Formeln zuordnet.*

(3.) *M_0 ist eine* Markierung *von ZT, genannt* Anfangsmarkierung. *Eine* Markierung *von ZT ist eine Abbildung, die jeder Prädikat-Stelle $p \in P_N$ mit $A_P(p) \in \widehat{\Pi}_{s_1...s_n}$ eine endliche Multimenge über Tupel von Individuen über $A_{s_1} \times ... \times A_{s_n}$, und jeder Ziel-Stelle $\widehat{p} \in \widehat{P}_N$ eine endliche Multimenge über geschlossenen, erweiterten Formeln zuordnet.*

Die Struktur eines Ziel/Transitions-Netzes wird durch ein Netz aus Stellen, Transitionen und Kanten festgelegt. Dabei wird die Menge der Stellen in eine Menge von Ziel-Stellen und in eine Menge von Prädikat-Stellen unterteilt. Während Prädikat-Stellen wie in Prädikat/Transitions-Netzen Tupel von Individuen als Marken enthalten, werden Ziel-Stellen mit geschlossenen, erweiterten Formeln markiert. Jede Prädikat-Stelle wird außerdem durch eine Bijektion mit einem variablen Prädikat assoziiert und jedes Tupel, das die Stelle markiert, muss die Stelligkeit des entsprechenden Prädikates respektieren. Kanten, die mit Prädikat-Stellen verbunden sind, werden mit nichtleeren Multimengen über Tupel von Termen gewichtet, die ebenfalls die Stelligkeiten der entsprechenden Prädikate berücksichtigen. Kanten, die mit Ziel-Stellen verbunden sind, werden mit nichtleeren Multimengen von erweiterten Formeln gewichtet. Erweiterte Formeln können auch Transitionen zugeordnet werden und als Wächter zusätzliche Restriktionen für die Aktivierung einer Transitioneninstanz beschreiben (siehe Definition 3.2).

Die graphische Darstellung von Ziel/Transitions-Netzen lehnt sich an die übliche Notation für Petri-Netze an: Stellen werden als Kreise, Transitionen als Rechtecke und Kanten als gerichtete Pfeile gezeichnet. Die Prädikat-Stellen werden mit den variablen Prädikatensymbolen beschriftet, während die Ziel-Stellen zur Unterscheidung durch einen doppelten Rand hervorgehoben werden. Der Wächter einer Transition wird in eckigen Klammern notiert. Wenn er lediglich eine Tautologie repräsentiert (der Wächter ist immer erfüllt, z.B. mit $\omega = \omega$ für ein beliebiges Konstantensymbol $\omega \in \Omega$), dann wird auf eine Darstellung verzichtet. Beispiele für Ziel/Transitions-Netze sind alle Abbildungen aus Abschnitt 3.1.

Für die Schaltregel von Ziel/Transitions-Netzen müssen zunächst die Regeln zur Auswertung von variablen Prädikaten festlegt werden. Wie bereits angedeutet, hängt die Extension eines variablen Prädikates vom Zustand des Ziel/Transitions-Netzes ab und nicht von einer in der Struktur zugeordneten Relation. Ein variables Prädikat ist genau dann erfüllt, wenn die Argumente des Prädikates in Form eines Tupels die assoziierte Stelle markieren. Umgekehrt bedeutet dies, dass die Negation des Prädikates genau dann erfüllt ist, wenn eine solche Marke nicht vorhanden ist. Wie in Prädikat/Transitions-Netzen reicht es aber nicht aus, die Begriffe Aktivierung und Folgemarkierung auf einer Transition zu erklären. Für die Auswertung von Kantengewichten und Wächtern wird eine Belegung der Variablen vorausgesetzt, um Terme und Formeln im Umfeld einer Transition zu schließen. Aus diesem Grund wird der für Prädikat/Transitions-Netze verwendete Begriff der *Transitioneninstanz* auf Ziel/Transitions-Netze übertragen. Eine Transitioneninstanz wird unter einer Markierung genau dann für aktiviert erklärt, wenn der Wächter der Transition erfüllt ist (für die gegebene Markierung und Belegung) und genug Marken im Vorbereich vorhanden sind, um den entsprechenden Kantengewichten zu genügen. Das Schalten einer aktivierten Transitioneninstanz konsumiert die identifizierten Tupel und Formeln von den Stellen des Vorbereichs und produziert neue Tupel und Formeln auf den Stellen des Nachbereichs. Es ist zu beachten, dass die Gewichte von Kanten an Ziel-Stellen im Gegensatz zu den Gewichten von Kanten an Prädikat-Stellen nicht vollständig durch die gegebene Struktur ausgewertet, sondern nur vereinfacht werden.

Definition 3.2 (Transitioneninstanz, Folgemarkierung)
Sei $ZT = (N, A_N, M_0)$ ein Ziel/Transitions-Netz, $t \in T_N$ eine Transition und M eine Markierung von ZT.

(1.) Für die Auswertung einer erweiterten Formel F bei M wird $\varepsilon(F)$ um folgenden Fall erweitert: Für $F = \widehat{\pi}(t_1, ..., t_n)$ mit $\widehat{\pi} \in \widehat{\Pi}$ ist $\varepsilon(F) = \top$ genau dann, wenn $(\varepsilon(t_1), ..., \varepsilon(t_n)) \in M(A_P^{-1}(\widehat{\pi}))$.

(2.) Sei $\chi(t)$ die Menge aller freien Variablen, die im Wächter $A_T(t)$ oder in einer Kante (x, y) mit $x = t$ oder $y = t$ vorkommen. Darüber hinaus sei α eine Belegung, die jeder Variablen $x \in X \cap \chi(t)$ einer Sorte $s \in S$ ein Konstantensymbol der Sorte s und jeder Formelvariablen $\widehat{x} \in \widehat{X} \cap \chi(t)$ eine geschlossene Formel zuordnet. Dann heißt $t : \alpha$ Transitioneninstanz.

(3.) Sei $t : \alpha$ eine Transitioneninstanz. $t : \alpha$ heißt genau dann aktiviert *unter M und kann schalten, wenn $\varepsilon(A_T(t) : \alpha) = \top$ bei M und:*

(3.1.) $\varepsilon(A_F(p,t):\alpha) \leq M(p)$ *für alle* $p \in P_N$,

(3.2.) $\sigma(A_F(\widehat{p},t):\alpha) \leq M(\widehat{p})$ *für alle* $\widehat{p} \in \widehat{P}_N$.

(4.) Sei $t:\alpha$ *eine aktivierte Transitioneninstanz in* M*. Das Schalten von* $t:\alpha$ *in* M *erzeugt eine* Folgemarkierung M' *nach folgender Vorschrift:*

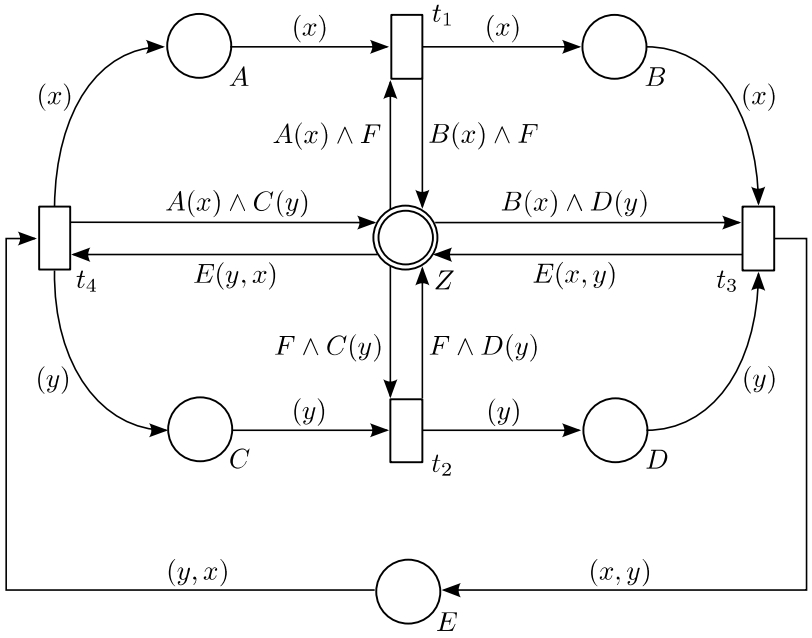
(4.1.) $M'(p) = M(p) - \varepsilon(A_F(p,t):\alpha) + \varepsilon(A_F(t,p):\alpha)$ *für alle* $p \in P_N$.

(4.2.) $M'(\widehat{p}) = M(\widehat{p}) - \sigma(A_F(\widehat{p},t):\alpha) + \sigma(A_F(t,\widehat{p}):\alpha)$ *für alle* $\widehat{p} \in \widehat{P}_N$.

Das Schalten einer Transitioneninstanz $t:\alpha$ *in einer Markierung* M *zu einer Folgemarkierung* M' *wird mit* $M \xrightarrow{t:\alpha}_{ZT} M'$ *oder kurz* $M \xrightarrow{t}_{ZT} M'$ *notiert. Geht* ZT *aus dem Kontext hervor, kann auf den Index verzichtet werden.*

Neben den in Abschnitt 3.1 gezeigten Beispielen, ist mit Abbildung 3.4 ein weiteres Beispiel für ein Ziel/Transitions-Netz gegeben, das die Aktivierung von Transitionen und die Berechnung von Folgemarkierungen veranschaulichen soll. In der Anfangsmarkierung sei $M_0(A) = (1)$, $M_0(C) = (0)$ und $M_0(Z) = A(1) \wedge C(0)$. Die restlichen Stellen seien mit der leeren Multimenge markiert, d.h. sie enthalten keine Marken. Sobald die beiden Anfangsmarken von A und C mit dem Schalten von t_1 und t_2 auf B und D abgelegt wurden, ist t_3 aktiviert. Wenn t_3 schaltet, wird eine zusammengesetzte Marke auf E erzeugt, die wiederum t_4 aktiviert. Mit dem Schalten von t_4 werden die Anfangsmarken daraufhin wieder in umgekehrter Reihenfolge auf A und C abgelegt. Anschließend können alle Transitionen erneut schalten, bis die Anfangsmarkierung des Netzes erreicht ist. Darüber hinaus aktualisiert jede Transition die Formel auf der Ziel-Stelle Z, so dass sie in jedem Zustand die aktuelle Markierung der Prädikat-Stellen in Form einer Konjunktion repräsentiert. Zu diesem Zweck wird beim Schalten von t_1 und t_2 auf eine Formelvariable F zugegriffen, die mit entsprechenden Teilformeln belegt werden kann. Auf der Ziel-Stelle Z kann somit in jeder Markierung der aktuelle Zustand des Netzes abgelesen werden. Die erreichbaren Markierungen und die entsprechenden Transitioneninstanzen des Netzes sind ebenfalls in Abbildung 3.4 dargestellt. Es handelt sich um ein beschränktes und stark lebendiges Netz (siehe Abschnitt 3.4).

Die Kanten von Z zu t_3 und von Z und t_4 in Abbildung 3.4 hätten auch mit einer Formelvariablen G gewichtet werden können. Das Verhalten des Netzes wäre dadurch unverändert geblieben, da die Marke, die von den Transitionen auf Z vorausgesetzt wird, in jeder Markierung vorhanden ist. Die entsprechenden Kantengewichte beschreiben also keine echten Restriktionen für die Aktivierung der Transitionen,



	Markierung						*Folgemarkierungen*
	A	B	C	D	E	Z	
M_0	(1)		(0)			$A(1) \wedge C(0)$	$M_0 \xrightarrow{t_1:\alpha_1} M_1,\ M_0 \xrightarrow{t_2:\alpha_2} M_2$
M_1		(1)	(0)			$B(1) \wedge C(0)$	$M_1 \xrightarrow{t_2:\alpha_3} M_3$
M_2	(1)			(0)		$A(1) \wedge D(0)$	$M_2 \xrightarrow{t_1:\alpha_4} M_3$
M_3		(1)		(0)		$B(1) \wedge D(0)$	$M_3 \xrightarrow{t_3:\alpha_5} M_4$
M_4					(1,0)	$E(1,0)$	$M_4 \xrightarrow{t_4:\alpha_6} M_5$
M_5	(0)		(1)			$A(0) \wedge C(1)$	$M_5 \xrightarrow{t_1:\alpha_7} M_6,\ M_5 \xrightarrow{t_2:\alpha_8} M_7$
M_6		(0)	(1)			$B(0) \wedge C(1)$	$M_6 \xrightarrow{t_2:\alpha_9} M_8$
M_7	(0)			(1)		$A(0) \wedge D(1)$	$M_7 \xrightarrow{t_1:\alpha_{10}} M_8$
M_8		(0)		(1)		$B(0) \wedge D(1)$	$M_8 \xrightarrow{t_3:\alpha_6} M_9$
M_9					(0,1)	$E(0,1)$	$M_9 \xrightarrow{t_4:\alpha_5} M_0$

	Belegung		
	x	y	F
α_1	1		$C(0)$
α_2		0	$A(1)$
α_3		0	$B(1)$
α_4	1		$D(0)$
α_5	1	0	
α_6	0	1	
α_7	0		$C(1)$
α_8		1	$A(0)$
α_9		1	$B(0)$
α_{10}	0		$D(1)$

Abbildung 3.4: Ein Ziel/Transitions-Netz mit einer vollständigen Darstellung aller erreichbaren Markierungen und Transitioneninstanzen.

sondern sehen lediglich das Entfernen der Marke auf Z vor. Eine weitere Modifikation des Netzes könnte darin bestehen, statt Konjunktionen getrennte Formeln zur Beschreibung des aktuellen Zustands zu verwenden. Die Anfangsmarkierung von Z wäre dann durch $M_0(Z) = A(1) + C(0)$ gegeben und in den Kanten von t_3 nach Z und von t_4 nach Z wäre die Konjunktion ebenfalls durch eine Summe zu ersetzen. Das Konjunktionsglied F in den Kanten zwischen t_1 und Z sowie zwischen t_2 und Z

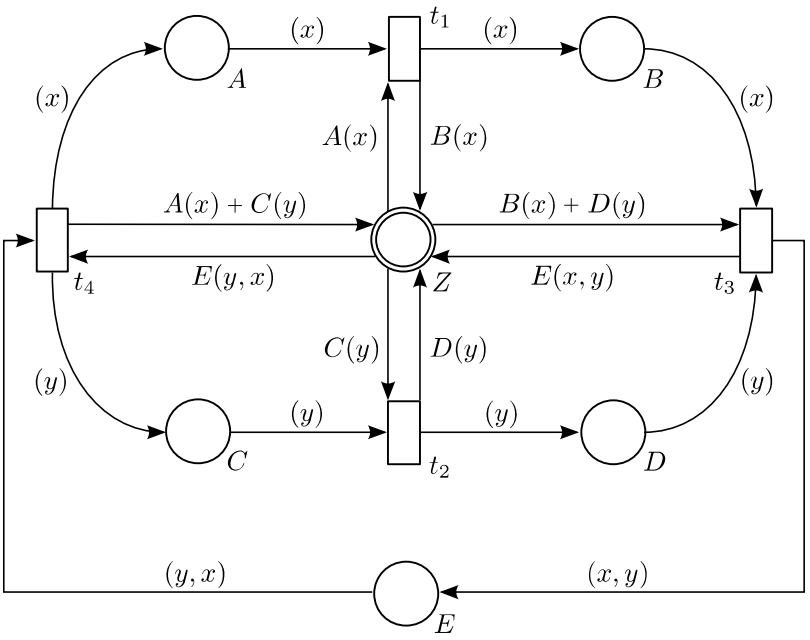


Abbildung 3.5: Modifikation des Netzes aus Abbildung 3.4.

wäre in diesem Fall überflüssig, da nur eine der beiden Marken zu aktualisieren ist. Für die Markierungen der Prädikat-Stellen ergeben sich mit diesen Modifikationen keine Änderungen und für die Markierungen der Ziel-Stelle wären lediglich Summen statt Konjunktionen zu berücksichtigen. Die Transitionen t_1 und t_2 stehen in dieser Variante aber nicht mehr im Konflikt sobald die Stellen A und C markiert sind, sondern sind nebenläufig aktiviert (siehe Abschnitt 3.4). In Abbildung 3.5 ist das modifizierte Netz dargestellt.

3.4 Grundbegriffe

In diesem Abschnitt werden die wichtigsten Grundbegriffe der Petri-Netz-Theorie auf Ziel/Transitions-Netze übertragen.

3.4.1 Erreichbarkeit

Die Menge aller erreichbaren Zustände eines Petri-Netz-Modells wird als Erreichbarkeitsmenge bezeichnet. Der Erreichbarkeitsgraph enthält alle Zustände und Zustandsübergänge in Form eines kantenbeschrifteten Graphen (Automaten).

Definition 3.3 (Erreichbarkeitsmenge, Erreichbarkeitsgraph)
Sei $ZT = (N, A_N, M_0)$ ein Ziel/Transitions-Netz und M sowie M' zwei Markierungen von ZT.

(1.) *M' heißt* erreichbar *aus M genau dann, wenn $M = M'$ oder eine nichtleere Folge von Transitioneninstanzen $t_1 : \alpha_1$ ... $t_n : \alpha_n$ existiert, so dass $M_i \xrightarrow{t_i \alpha_i} M_{i+1}$ für alle $i \in \{1..n\}$ mit $M_1 = M$ und $M_{n+1} = M'$. Die Folge $t_1 : \alpha_1$ ... $t_n : \alpha_n$ heißt* Schaltfolge *in M und die Erreichbarkeit von M' aus M wird mit $M \xrightarrow{*} M'$ notiert.*

(2.) *Die Menge der erreichbaren Markierungen von ZT, genannt* Erreichbarkeitsmenge *von ZT, ist gegeben durch $R_{ZT} = \{M \mid M_0 \xrightarrow{*} M\}$.*

(3.) *Der* Erreichbarkeitsgraph *$G_{ZT} = (V, E)$ von ZT ist gegeben durch die Knotenmenge $V = R_{ZT}$ und die Kantenmenge $E = \{(M, t : \alpha, M') \mid M \xrightarrow{t:\alpha} M'\}$. Ein Tupel $(M, t : \alpha, M')$ beschreibt dabei eine Kante von M nach M', die mit $t : \alpha$ beschriftet ist.*

Ob eine Markierung M' aus einer Markierung M erreichbar ist, lässt sich im Erreichbarkeitsgraphen daran erkennen, ob ein Pfad von Knoten M zu Knoten M' existiert. Ist die Anfangsmarkierung aus jeder Markierung erreichbar, dann wird das Netz als reversibel bezeichnet. Abbildung 3.6 zeigt den Erreichbarkeitsgraphen für das reversible Ziel/Transitions-Netz aus Abbildung 3.4.

Es ist durchaus möglich, dass der Zustandsraum eines Netzes unendlich groß ist. In diesem Fall kann die Erreichbarkeitsmenge nicht vollständig berechnet werden und der entsprechende Erreichbarkeitsgraph ist nicht konstruierbar. Eine Erreichbarkeitsanalyse kann dann nicht alle Zustände des Modells berücksichtigen. In diesem Fall ist es möglicherweise sinnvoll, die Analyse auf eine Teilmenge der Erreichbarkeitsmenge zu beschränken, wenn die zu untersuchenden Eigenschaften dadurch eindeutig entschieden werden können. Eine andere Möglichkeit besteht darin, die Idee der parametrisierten Erreichbarkeitsgraphen [Lin89, Lin93] von Prädikat/-Transitions-Netzen auf Ziel/Transitions-Netze zu übertragen. In parametrisierten

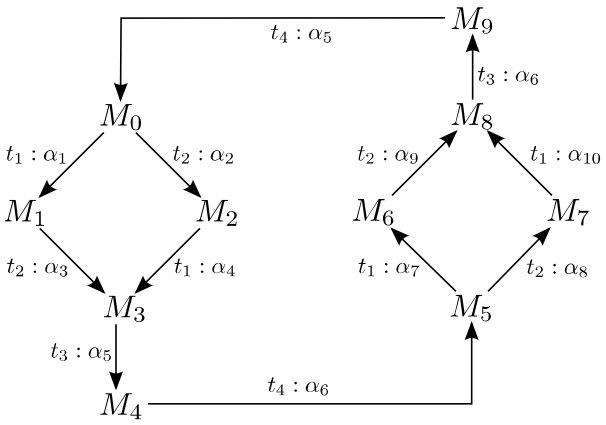


Abbildung 3.6: Der Erreichbarkeitsgraph zu Abbildung 3.4.

Erreichbarkeitsgraphen werden verschiedene Markierungen eines Prädikat/Transitions-Netzes durch die Einführung von Parametern zu parametrisierten Markierungen zusammengefasst. Beim Schalten einer Transition werden ausgewählte Parameter anstelle von konkreten Variablenbelegungen verwendet. Jeder Parameter dient dabei als Platzhalter für eine Menge von Individuen. Auf diese Weise werden alle durch das Schalten der Transition hervorgehenden Markierungen in einem Ausdruck repräsentiert. Durch eine geeignete Fixierung, d.h. eine Substitution der Parameter mit Individuen, kann aus diesem Ausdruck die Menge der konkreten Markierungen extrahiert werden. Parametrisierte Erreichbarkeitsgraphen verallgemeinern damit die Idee von Überdeckbarkeitsgraphen, die bei der Analyse von S/T-Systemen verwendet werden (siehe z.B. [Rei82, Sta90, PW03]). Parametrisierte Erreichbarkeitsgraphen sind in jedem Fall endlich [Lin89, Lin93] und die Idee lässt sich auf Algebraische Petri-Netze im Allgemeinem übertragen [Sch95].

3.4.2 Nebenläufigkeit

Eine Multimenge von Transitioneninstanzen wird genau dann als nebenläufig bei einer Markierung bezeichnet, wenn alle Transitioneninstanzen aktiviert sind und darüber hinaus genug Marken auf den Stellen im Vorbereich vorhanden sind, um alle Transitioneninstanzen »unabhängig voneinander« schalten zu können. Da das Schalten von Transitionen in Ziel/Transitions-Netzen an eine Variablenbelegung gebunden ist, kann der Begriff der Nebenläufigkeit auf Transitioneninstanzen und auf Transitionen erklärt werden.

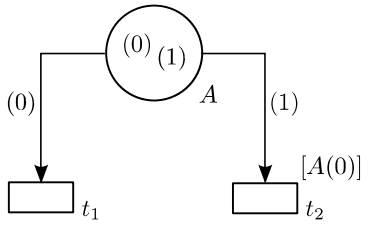


Abbildung 3.7: Ein Ziel/Transitions-Netz: Obwohl die Stelle A genug Marken enthält, um beiden Kantengewichten zu genügen, sind die entsprechenden Transitioneninstanzen aufgrund des Wächters von t_2 nicht nebenläufig aktiviert.

Definition 3.4 (Nebenläufigkeit)
Sei $ZT = (N, A_N, M_0)$ ein Ziel/Transitions-Netz und M eine Markierung von ZT. Eine Multimenge G von Transitioneninstanzen heißt genau dann nebenläufig *bei M, wenn jede Transitioneninstanz $t : \alpha \in G$ aktiviert ist unter M und*

(1.) $\sum_{t:\alpha \in G} G(t:\alpha) \cdot \varepsilon(A_F(p,t):\alpha) \leq M(p)$ *für alle* $p \in P_N$,

(2.) $\sum_{t:\alpha \in G} G(t:\alpha) \cdot \sigma(A_F(\widehat{p},t):\alpha) \leq M(\widehat{p})$ *für alle* $\widehat{p} \in \widehat{P}_N$,

(3.) Jede Folge von Transitioneninstanze aus G ist eine Schaltfolge in M.

Eine Multimenge von Transitionen $\{t_1,...,t_n\}$ heißt genau dann nebenläufig *bei M, wenn eine Multimenge von Belegungen $\{\alpha_1, ..., \alpha_2\}$ existiert, so dass $\{t_1 : \alpha_1, ..., t_2 : \alpha_2\}$ nebenläufig ist bei M.*

Absatz (3.) in Definition 3.4 ist aufgrund der Verwendung von variablen Prädikaten in Wächtern notwendig. In Ziel/Transitions-Netzen können Transitioneninstanzen unabhängig von gemeinsamen Vorbereichsstellen anderen Transitioneninstanzen die Aktivierung entziehen, indem sie die Markierung einer Stelle verändern, die gleichzeitig von einem variablen Prädikat im Wächter einer anderen Transition vorausgesetzt wird. Ein Beispiel zeigt Abbildung 3.7. Aus diesem Grund muss jede Folge von Transitioneninstanzen, die aus G gebildet werden kann, eine Schaltfolge sein.

3.4.3 Konflikt

Wenn eine Multimenge von Transitioneninstanzen unter einer Markierung aktiviert ist, das Schalten einer Transitioneninstanz anderen Transitioneninstanzen die Aktivierung jedoch entzieht, dann liegt ein Konflikt vor. Der Begriff des Konfliktes

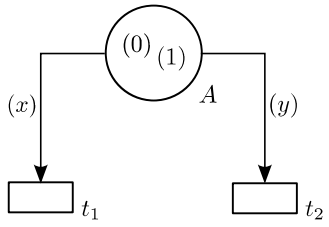


Abbildung 3.8: Ein Ziel/Transitions-Netz: Die Transitionen t_1 und t_2 sind bei der dargestellten Markierung gleichzeitig nebenläufig und im Konflikt, weil z.B. $t_1 : (x \to 0)$ und $t_2 : (y \to 1)$ nebenläufig und $t_1 : (x \to 0)$ und $t_2 : (y \to 0)$ im Konflikt sind.

kann ebenso wie der Begriff der Nebenläufigkeit auf Transitioneninstanzen und auf Transitionen erklärt werden.

Definition 3.5 (Konflikt)
Sei $ZT = (N, A_N, M_0)$ ein Ziel/Transitions-Netz und M eine Markierung von ZT. Eine Multimenge G von Transitioneninstanzen heißt genau dann im Konflikt *bei M, wenn jede Transitioneninstanz $t : \alpha \in G$ aktiviert ist unter M und G nicht nebenläufig ist bei M. Eine Multimenge von Transitionen $\{t_1,...,t_n\}$ heißt genau dann* im Konflikt *bei M, wenn eine Multimenge von Belegungen $\{\alpha_1, ..., \alpha_2\}$ existiert, so dass $\{t_1 : \alpha_1, ..., t_2 : \alpha_2\}$ im Konflikt ist bei M.*

Ein Konflikt in Petri-Netzen deutet auf einen Mangel an Informationen hin, weil das Verhalten des Systems in der entsprechenden Markierung nicht eindeutig spezifiziert ist [Sta90]. Eine solche Situation muss aber nicht zwangsläufig einen Fehler identifizieren, wenn die Modellierung tatsächlich Nichtdeterminismus vorsieht. Im Netz aus Abbildung 3.4 kommt es immer dann zu einem Konflikt zwischen t_1 und t_2, wenn die Stellen A und C markiert sind. Der Konflikt wird durch die Stelle Z verursacht, die stets nur eine Formel enthält, die sowohl von t_1 als auch von t_2 beansprucht wird. Im modifizierten Netz aus Abbildung 3.5 sind t_1 und t_2 stattdessen nebenläufig aktiviert.

Wenn die Variablenbelegungen von Transitionen unberücksichtigt bleiben, schließen sich Nebenläufigkeit und Konflikt in Ziel/Transitions-Netzen nicht gegenseitig aus. Abbildung 3.8 zeigt, dass eine Multimenge von Transitionen im Gegensatz zu einer Multimenge von Transitioneninstanzen bei einer Markierung eines Ziel/Transitions-Netzes gleichzeitig nebenläufig und im Konflikt sein kann.

Situationen, in denen Transitioneninstanzen gleichzeitig aktiviert sind, lassen sich im Erreichbarkeitsgraphen an Knoten erkennen, von denen mehrere ausgehende Kanten verzweigen. Anhand von Verzweigungen ist es jedoch nicht möglich, Nebenläufigkeit und Konflikt zu unterscheiden. Der Informationsverlust basiert auf der Tatsache, dass jeder Pfad im Erreichbarkeitsgraphen einen konkreten Systemablauf repräsentiert, in dem alle Konflikte bereits aufgelöst wurden und für das Schalten der Transitioneninstanzen eine sequentielle Semantik zu Grunde gelegt wird. Sollen Systemabläufe unter diesen Aspekten untersucht werden, so bietet sich die Analyse auf Basis einer kausalen Semantik an. Für Ziel/Transitions-Netze wird eine entsprechende Ablaufdarstellung im nächsten Kapitel entwickelt. In einer kausalen Semantik kommt den Begriffen *kausale Abhängigkeit* und *kausale Unabhängigkeit* von Vorgängen eine besondere Bedeutung zu.

3.4.4 Beschränktheit

Beschränktheit ist eine dynamische Eigenschaft eines Modells, d.h. sie hängt von der Anfangsmarkierung des Netzes ab. Sie stellt sicher, dass die Erreichbarkeitsmenge endlich ist und der Erreichbarkeitsgraph konstruiert und vollständig analysiert werden kann. Unbeschränkte Systeme besitzen einen unendlich großen Zustandsraum und sind überhaupt nicht realisierbar. Solche Systeme sind daher auch nur von geringem praktischen Interesse. Für Ziel/Transitions-Netze bedeutet Beschränktheit, dass die Anzahl der möglichen Marken auf jeder Stelle endlich ist. Im Gegensatz zu S/T-Systemen wird aber nicht die Gesamtzahl der Marken auf einer Stelle begrenzt, sondern die Häufigkeit jedes Tupels bzw. jeder Formel. Als Schranken werden dazu Multimengen über Tupel von Individuen bzw. Multimengen über geschlossenen Formeln verwendet.

Definition 3.6 (Beschränktheit)
Sei $ZT = (N, A_N, M_0)$ ein Ziel/Transitions-Netz. Eine Prädikat-Stelle $p \in P_N$ heißt genau dann beschränkt, *wenn eine endliche Multimenge b über Tupel von Individuen existiert, so dass $M(p) \leq b$ für alle erreichbaren Markierungen $M \in R_{ZT}$. Eine Ziel-Stelle $\widehat{p} \in \widehat{P}_N$ heißt genau dann* beschränkt, *wenn eine endliche Multimenge $\widehat{b}$ über geschlossenen Formeln existiert, so dass $M(\widehat{p}) \leq \widehat{b}$ für alle erreichbaren Markierungen $M \in R_{ZT}$. ZT heißt genau dann* beschränkt, *wenn jede Stelle $p \in P_N \cup \widehat{P}_N$ beschränkt ist.*

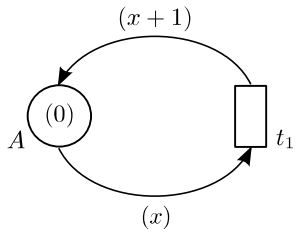


Abbildung 3.9: Ein Ziel/Transitions-Netz: Obwohl niemals mehr als eine Marke auf A vorhanden sein kann, ist die Erreichbarkeitsmenge des Netzes unendlich groß und das Netz damit nicht beschränkt.

Es ist zu beachten, dass die Schranken durch endliche Multimengen repräsentiert werden müssen, um auf eine endliche Erreichbarkeitsmenge des Netzes zu schließen. Abbildung 3.9 zeigt, dass ein Ziel/Transitions-Netz unendlich viele Zustände besitzen kann, wenn nichtendliche Multimengen als Schranken zugelassen werden. Offensichtlich ist ein Netz genau dann beschränkt, wenn sein Erreichbarkeitsgraph endlich ist.

3.4.5 Lebendigkeit

Wie Beschränktheit zählt auch Lebendigkeit zu den dynamischen Eigenschaften eines Modells. Lebendigkeit und damit verwandte Begriffe können in verschiedenen Ausprägungen für Transitionen, Markierungen und Netze erklärt werden. Dieser Abschnitt beschränkt sich auf die zwei wichtigsten Lebendigkeitsbegriffe für Netze: Die schwache Lebendigkeit garantiert, dass es keinen Zustand im Modell gibt, in dem es zu keinen Zustandsübergängen mehr kommen kann. Das Ausschließen von solchen Verklemmungen ist insbesondere für nebenläufige Systeme, wie sie mit Petri-Netzen typischerweise untersucht werden, von Bedeutung. Die starke Lebendigkeit garantiert darüber hinaus, dass in jedem Zustand des Modells jede Transition die Möglichkeit hat, in einem erreichbaren Folgezustand aktiv zu werden.

Definition 3.7 (Lebendigkeit)
Sei $ZT = (N, A_N, M_0)$ ein Ziel/Transitions-Netz. ZT heißt genau dann schwach lebendig, *wenn in jeder erreichbaren Markierung $M \in R_{ZT}$ eine Transitioneninstanz $t : \alpha$ existiert, die unter M aktiviert ist. ZT heißt genau dann* stark lebendig, *wenn in jeder erreichbaren Markierung $M \in R_{ZT}$ für jede Transition $t \in T_N$ eine Belegung*

α und eine aus M erreichbare Markierung $M' \in R_{ZT}$ existiert, so dass $t : \alpha$ unter M' aktiviert ist.

Die schwache Lebendigkeit eines Netzes kann an Hand von Blättern im Erreichbarkeitsgraphen entschieden werden, d.h. anhand von Knoten, von denen keine Kanten ausgehen. Enthält der Erreichbarkeitsgraph keine Blätter, dann ist das Netz schwach lebendig, da jedes Blatt eine Verklemmung identifiziert. Existiert darüber hinaus ausgehend von jedem Knoten für jede Transition ein Pfad, der mit einer entsprechenden Transitioneninstanz beschriftet ist, dann ist das Netz auch stark lebendig. Die Netze aus Abbildung 3.4 und 3.5 sind beschränkt und stark lebendig (und damit auch schwach lebendig).

3.5 Mächtigkeit

Ziel/Transitions-Netze sind Algebraische Petri-Netze [EPR94] und vereinen die Vorteile von algebraischen Spezifikationen mit den Vorteilen von Petri-Netzen. Sie verallgemeinern die Grundidee von Prädikat/Transitions-Netzen, indem sie variable Prädikate und Formelvariablen für die Bildung von Formeln zulassen und neben Tupeln auch Formeln als Marken im Netz erlauben. Offensichtlich können Prädikat/Transitions-Netze somit als Spezialfall von Ziel/Transitions-Netzen erklärt werden. Zu diesem Zweck ist $\widehat{P}_N = \emptyset$ und die Verwendung von erweiterten Formeln in Wächtern von Transitionen nicht erlaubt. Ein grundlegender Unterschied zwischen Ziel/Transitions-Netzen und anderen Petri-Netz-Arten besteht darin, dass mit der Formulierung eines geeigneten Wächters die Aktivierung einer Transition auch von Stellen abhängig sein kann, die weder im Vor- noch im Nachbereich der Transition enthalten sind. Es ist also möglich, dass eine Transition auf Informationen über die Markierung einer Stelle zugreift, mit der sie nicht oder nur indirekt verbunden ist. Lediglich die Manipulation der vorhandenen Marken erfordert eine direkte Verbindung. Der in Abbildung 3.3 dargestellte Nulltest ist z.B. genauso durchführbar, wenn die Kante zwischen der Transition t und der Stelle A nicht existieren würde.

Prädikat/Transitions-Netze besitzen nicht die selbe Mächtigkeit wie Turingmaschinen, d.h. mit ihnen können nicht alle berechenbaren Funktionen berechnet werden, sofern endliche Signaturen und Strukturen vorausgesetzt werden. Das Problem kann auf die Unfähigkeit von Prädikat/Transitions-Netze zurückgeführt werden, einen Nulltest auf einer Stelle durchzuführen. Es ist zwar möglich, durch Anliegen einer

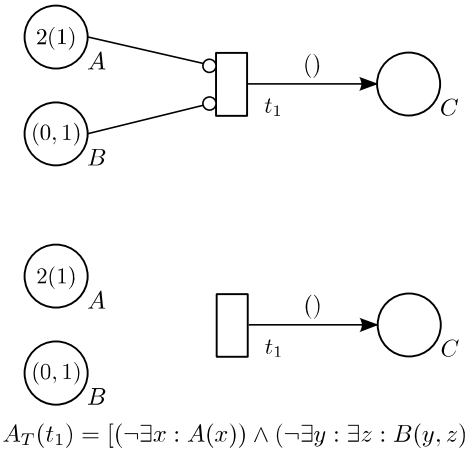
Transition im Nachbereich zu prüfen, ob sich *mindestens* eine Marke auf einer Stelle befindet. Es ist im Allgemeinen aber nicht möglich, zu prüfen, ob sich *keine* Marke auf der Stelle befindet. Eine Transition, die bei einer *sauberen* Stelle aktiviert ist, ist auch bei jeder anderen Anzahl von Marken auf der Stelle aktiviert. Es existieren Erweiterungen wie z.B. Inhibitor-Netze, um diesem Problem zu begegnen. In Inhibitor-Netzen wird durch eine Abbildung $I : T_N \to 2^{P_N}$ für jede Transition eine Menge von Inhibitorkanten festlegt (siehe z.B. [PW03]). Inhibitorkanten verlaufen dabei stets von einer Stelle zu einer Transition und werden mit einem Kreis statt mit einer Pfeilspitze gezeichnet. Mit Inhibitorkanten wird die Aktivierung einer Transition $t \in T_N$ unter einer Markierung M eingeschränkt, indem zusätzlich verlangt wird, dass alle Stellen in $I(t)$ keine Marken enthalten dürfen, d.h. $\forall p \in I(t) : M(p) = 0$.

Wie bereits in Abschnitt 3.1 angedeutet, lassen sich Nulltests ebenfalls mit Ziel/-Transitions-Netzen realisieren. Dazu werden entsprechende Wächter mit variablen Prädikaten eingeführt, die die selben Restriktionen beschreiben wie Inhibitorkanten in Inhibitor-Netzen. Ziel/Transitions-Netze können das Verhalten von Inhibitor-Netzen somit simulieren.

Theorem 3.1
Zu jedem Prädikat/Transitions-Netz H mit Inhibitorkanten existiert ein verhaltensgleiches Ziel/Transitions-Netz $ZT(H)$, d.h. H und $ZT(H)$ haben den selben Erreichbarkeitsgraphen.

Beweis: $ZT(H)$ wird mit dem selben Netz, den selben Beschriftungen und der selben Anfangsmarkierung wie H konstruiert. Für jede Inhibitorkante von einer Stelle $p \in I(t)$ zu einer Transition $t \in T_N$ im Inhibitor-Netz H wird außerdem eine Formel $G_p = \exists x_1 :, ..., \exists x_n : A_P(p)(x_1, ..., x_n)$ in $ZT(H)$ konstruiert mit $(x_1, ..., x_n) \in X_{s_1} \times ... \times X_{s_n}$ für $A_P(p) \in \Pi_{s_1,...,s_n}$. Der Wächter $A_T(t)$ im Ziel/Transitions-Netz $ZT(H)$ wird anschließend um die Konjunktionsglieder $\bigwedge_{p \in I(t)}(\neg G_p)$ erweitert. Befindet sich in einer beliebigen Markierung M mindestens ein Tupel auf einer Stelle $p \in I(t)$ im Inhibitor-Netz H, dann ist das Konjunktionsglied $\neg G_p$ im Wächter von t des entsprechenden Ziel/Transitions-Netzes $ZT(H)$ nicht erfüllt und t ist nicht aktiviert unter M. Der Wächter $A_T(t)$ in $ZT(H)$ repräsentiert somit genau die Restriktionen, die die entsprechenden Inhibitorkanten von t in H beschreiben. Alle Zustände und Zustandsübergänge von H und $ZT(H)$ sind daher identisch. Ein Beispiel ist in Abbildung 3.10 dargestellt. □

Korollar 3.1
Ziel/Transitions-Netze sind turingmächtig.



$$A_T(t_1) = [(\neg\exists x : A(x)) \wedge (\neg\exists y : \exists z : B(y, z))]$$

Abbildung 3.10: Oben: Ein Inhibitor-Netz mit zwei Inhibitorkanten. Unten: Ein verhaltensgleiches Ziel/Transitions-Netz ohne Inhibitorkanten. In beiden Fällen ist t_1 nicht aktiviert, da sowohl die Stelle A als auch die Stelle B Marken enthält.

Beweis: Folgt aus Theorem 3.1 und dem Beweis, dass Inhibitor-Netze im Allgemeinen turingmächtig sind (siehe z.B. [PW03, S. 166]). □

Die Fähigkeit von Ziel/Transitions-Netzen, die Zustände eines Modells in den Zuständen eines Modells zu beschreiben, kann als eine Form des Selbstbezugs verstanden werden. Dieser Selbstbezug wird allerdings durch die Unterscheidung der beiden Stellen-Arten eingeschränkt. Formeln mit variablen Prädikaten können ausschließlich auf den Ziel-Stellen eines Netzes vorhanden sein, während sie gleichzeitig ausschließlich die Markierung von Prädikat-Stellen beschreiben können. Ziel/Transitions-Netze berühren somit nicht die Aspekte der Logik höherer Ordnung.

Die Verwendung von Formeln als Marken in Petri-Netzen wird auch in [BD08] vorgeschlagen. In dieser Arbeit werden *Simple Logic Petri Nets* eingeführt, die ihren Ursprung in der Logikprogrammierung haben. Im Unterschied zu Ziel/Transitions-Netzen werden bei dieser Variante aber nur Literale (Prädikate und deren Negationen) als Markierungen und Kantengewichte zugelassen, d.h. zusammengesetzte Formeln können nicht verwendet werden. Stellen, die gewöhnliche Tupel bzw. Individuen enthalten, sind ebenfalls nicht vorgesehen. Somit existieren auch keine variablen Prädikate. In [BD08] wird gezeigt, wie mit *Simple Logic Petri-Nets* einfache Agenten nach dem BDI-Modell spezifiziert und verifiziert werden können. Die Idee besteht darin, ein gegebenes Netz in einen temporallogischen Formalismus zu überführen und die geforderten Eigenschaften mit Model Checking zu prüfen. Aufgrund

der Einschränkungen bleiben die Darstellungsmöglichkeiten von *Simple Logic Petri Nets* gegenüber Ziel/Transitions-Netzen begrenzt. Eine Modellierung von dynamischen Zielen in Agentensystemen ist nicht möglich.

3.6 Modifikationen

In diesem Abschnitt werden einige Modifikationen der bisher eingeführten Grundform von Ziel/Transitions-Netzen vorgestellt. Die entsprechenden Erweiterungen bzw. Einschränkungen sollen entweder die Darstellungsmöglichkeiten von Ziel/Transitions-Netzen erweitern oder die Realisierung von Werkzeugen vereinfachen.

3.6.1 Indizierte Prädikate

Die erste Idee für eine Modifikation von Ziel/Transitions-Netzen basiert auf der Tatsache, dass mit den bisherigen Konzepten nicht alle Zustände eines Modells eindeutig durch Formeln repräsentiert werden können. In Ziel/Transitions-Netzen werden die Prädikat-Stellen mit Multimengen von Tupeln markiert, weshalb es möglich ist, dass auf einer Stelle in einer Markierung mehrere identische Marken vorhanden sind. Für die Auswertung der variablen Prädikate ist aber nur entscheidend, ob das entsprechende Tupel auf der Stelle existiert oder nicht (siehe Definition 3.2). Es genügt, wenn mindestens ein Tupel die Stelle markiert, die Häufigkeit spielt darüber hinaus keine Rolle. Es ist allerdings denkbar, dass in einer Formel auch auf die Häufigkeit einer Marke Bezug genommen werden soll. In diesem Fall können indizierte Prädikatensymbole verwendet werden. Indizierte Prädikatensymbole sind variable Prädikatensymbole, die durch einen Index die geforderte Häufigkeit einer Marke beschreiben. Ein indiziertes Prädikat ist genau dann erfüllt, wenn das entsprechende Tupel mindestens so häufig auf der assoziierten Stelle vorhanden ist, wie der Index des Prädikates verlangt.

Definition 3.8 (Z/T-Netze mit indizierten Prädikaten)
Sei $ZT = (N, A_N, M_0)$ *ein Ziel/Transitions-Netz,* $\widehat{\pi} \in \widehat{\Pi}_{s_1 \ldots s_n}$ *ein variables Prädikatensymbol und* $k \in \mathbb{N}_+$. *Dann heißt* $\widehat{\pi}_k$ indiziertes Prädikatensymbol. *Für die Auswertung einer erweiterten Formel* F *mit indizierten Prädikatensymbolen bei einer Markierung* M, *wird* $\varepsilon(F)$ *um folgenden Fall erweitert: Für* $F = \widehat{\pi}_k(t_1, ..., t_n)$ *mit* $\widehat{\pi} \in \widehat{\Pi}$ *und* $k \in \mathbb{N}_+$ *ist* $\varepsilon(F) = \top$ *genau dann, wenn* $k \leq M(A_P^{-1}(\pi))(\varepsilon(t_1), ..., \varepsilon(t_n))$.

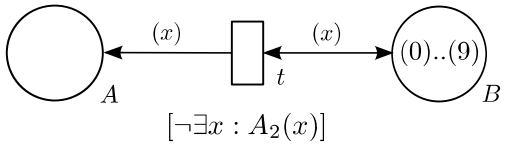


Abbildung 3.11: Ein Ziel/Transitions-Netz mit einem indizierten Prädikat.

ZT heißt Ziel/Transitions-Netz mit indizierten Prädikaten *genau dann, wenn ZT eine Formel mit indizierten Prädikatensymbolen enthält.*

Offensichtlich ist $\widehat{\pi}_1(t_1, ..., t_n) = \widehat{\pi}(t_1, ..., t_n)$ für alle variablen Prädikate $\widehat{\pi} \in \widehat{\Pi}$ und alle zulässigen Terme t_i. Abbildung 3.11 zeigt ein Beispiel für die Verwendung eines indizierten Prädikates. In diesem Netz kann die Transition t solange beliebig Zahlen zwischen 0 und 9 von der Stelle B auf die Stelle A kopieren, bis eine der Zahlen zweimal oder häufiger vorliegt. Wäre die Existenz von genau zwei Marken gefordert, so wäre der Wächter von t um das Konjunktionsglied $\neg A_3(x)$ zu erweitern. Auf ähnliche Weise kann jede Markierung eines Ziel/Transitions-Netzes eindeutig durch eine Formel mit indizierten Prädikaten repräsentiert werden.

Theorem 3.2
Sei $ZT = (N, A_N, M_0)$ ein Ziel/Transitions-Netz. Für jede Markierung M von ZT existiert eine Formel F mit indizierten Prädikaten, so dass $\varepsilon(F) = \top$ bei M und $\varepsilon(F) = \bot$ für alle Markierungen M' mit $M'(p) = M(p)$ für eine Prädikat-Stelle $p \in P_N$.

Beweis: Für jede Prädikat-Stelle $p \in P_N$ wird eine Konjunktion $G_p = \bigwedge_{(d_1,...,d_n) \in M(p)} \widehat{\pi}_k(c_1, ..., c_n) \wedge \neg\widehat{\pi}_{k+1}(c_1, ..., c_n)$ mit $\widehat{\pi} = A_P(p)$, $k = M(p)(d_1, ..., d_n)$ und $\varepsilon(c_i) = d_i$ für alle $i \in \{1..n\}$ konstruiert. Dann ist $\varepsilon(F) = \top$ bei M für $F = \bigwedge_{p \in P_N} G_p$. Für jede Markierung M' mit $M'(p) = M(p)$ für eine Prädikat-Stelle $p \in P_N$ ist die Konjunktion G_p nicht erfüllt, so dass $\varepsilon(F) = \bot$ für M'. □

3.6.2 Kapazitäten

Eine weitere Modifikation von Ziel/Transitions-Netzen besteht in der Einführung von Kapazitäten. Kapazitätsvorschriften in S/T-Systemen beschreiben die Grenzen für die Anzahl von Marken auf einer Stelle und legen somit zusätzliche Restriktionen für die Aktivierung einer Transition fest. Wenn durch das Schalten einer Transition

in der Folgemarkierung mehr Marken auf einer Stelle vorhanden wären, als die Kapazität der Stelle zulässt, dann ist die Transition nicht aktiviert. Die Vernachlässigung von Kapazitäten in S/T-Systemen stellt keine echte Einschränkung dar, da für jedes S/T-System mit Kapazitäten ein äquivalentes S/T-System ohne Kapazitäten konstruiert werden kann (siehe z.B. [PW03, Sta90]). Für höhere Petri-Netze wie z.B. Prädikat/Transitions-Netze oder Gefärbte Petri-Netze wird in der Regel keine Kapazität erklärt, weshalb auch in der Grundform von Ziel/Transitions-Netzen darauf verzichtet wurde. Besteht dennoch der Wunsch, Kapazitätsvorschriften in Ziel/Transitions-Netzen zu hinterlegen, so kann eine Kapazitätsfunktion verwendet werden. Eine Kapazitätsfunktion ordnet jeder Prädikat-Stelle eine endliche Multimenge über Tupel von Individuen und jeder Ziel-Stelle eine endliche Multimenge über geschlossenen, erweiterten Formeln zu. Sie legt auf diese Weise die oberen Schranken für die Anzahl von Marken auf einer Stelle fest. Markierungen dürfen die Kapazität nicht verletzen und für die Aktivierung einer Transition t unter einer Markierung M ist zusätzlich zu fordern, dass auf keiner Stelle im Nachbereich von t durch das Schalten von t die Kapazität in der Folgemarkierung überschritten wird.

Definition 3.9 (Z/T-Netze mit Kapazitäten)
$ZT = (N, A_N, M_0, C_N)$ heißt Ziel/Transitions-Netz mit Kapazitäten *genau dann, wenn (N, A_N, M_0) ein Ziel/Transitions-Netz und C_N eine Abbildung ist, die jeder Prädikat-Stelle $p \in P_N$ mit $A_P(p) \in \widehat{\Pi}_{s_1 \ldots s_n}$ eine endliche Multimenge über Tupel von Individuen über $A_{s_1} \times \ldots \times A_{s_n}$, und jeder Ziel-Stelle $\widehat{p} \in \widehat{P}_N$ eine endliche Multimenge über geschlossenen, erweiterten Formeln zuordnet. Darüber hinaus gilt für die Anfangsmarkierung $M_0(p) \leq C_N(p)$ für alle $p \in P_N \cup \widehat{P}_N$. Für eine aktivierte Transitioneninstanz $t : \alpha$ unter einer Markierung M wird zusätzlich verlangt:*

(1.) $M(p) - \varepsilon(A_F(p,t) : \alpha) + \varepsilon(A_F(t,p) : \alpha) \leq C_N(p)$
für alle Prädikat-Stellen $p \in tF_N \cap P_N$,

(2.) $M(\widehat{p}) - \sigma(A_F(\widehat{p},t) : \alpha) + \sigma(A_F(t,\widehat{p}) : \alpha) \leq C_N(\widehat{p})$
für alle Ziel-Stellen $\widehat{p} \in tF_N \cap \widehat{P}_N$.

Wie in S/T-Systemen ist es möglich, durch die Einführung von *Komplementstellen* zu jedem Ziel/Transitions-Netz mit Kapazitäten ein äquivalentes Netz ohne Kapazitäten zu konstruieren. Dazu wird für jede Stelle $p \in P_N \cup \widehat{P}_N$ eine Komplementstelle $\overline{p}$ mit $M_0(\overline{p}) = C_N(p) - M_0(p)$ als Anfangsmarkierung eingeführt. Um den Zugang von Marken auf p mit der Komplementstelle $\overline{p}$ zu erfassen, wird für jede Kante $(t,p) \in F_N$ von einer Transition $t \in T_N$ eine Kante $(\overline{p},t)$ mit $A_F(\overline{p},t) = A_F(t,p)$

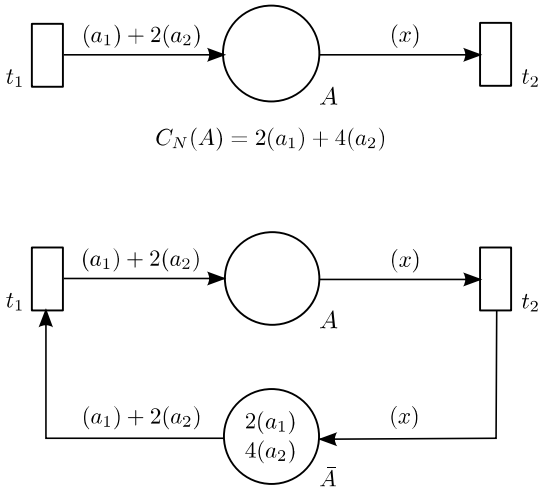


Abbildung 3.12: Oben: Ein Ziel/Transitions-Netz mit Kapazitäten. Unten: Ein äquivalentes Ziel/Transitions-Netz ohne Kapazitäten. $\bar{A}$ stellt dabei die Komplementstelle von A dar.

eingeführt. Entsprechend wird für jede Kante $(p, t) \in F_N$ zu einer Transition $t \in T_N$ eine Kante $(t, \bar{p})$ mit $A_F(t, \bar{p}) = A_F(p, t)$ eingeführt, um den Abgang von Marken zu erfassen. Damit ist $M(p) + M(\bar{p}) = C_N(p)$ konstant für alle Stellen $p \in P_N \cup \widehat{P}_N$ in jeder erreichbaren Markierung M. Abbildung 3.12 zeigt ein Beispiel.

Um auch mit unbegrenzten Kapazitäten umgehen zu können, wäre es möglich, Multimengen mit unendlichen Häufigkeiten als Kapazitätsvorschriften zuzulassen. Allerdings lässt sich in diesem Fall nicht mehr zu jedem Ziel/Transitions-Netz mit Kapazitäten ein äquivalentes Ziel/Transitions-Netz ohne Kapazitäten konstruieren. Das Problem besteht darin, dass für jedes Tupel mit unbegrenzter Kapazität eine ebenfalls unbegrenzte Anfangsmarkierung auf den entsprechenden Komplementstellen vorausgesetzt werden muss. Ziel/Transitions-Netze ohne Kapazitäten können in diesem Fall aber als Spezialfälle von Ziel/Transitions-Netzen mit Kapazitäten erklärt werden. Dazu wird jedem Tupel für jede Stelle eine unbegrenzte Kapazität zugeordnet. Mit Ausnahme von Abbildung 3.12 handelt es sich bei allen anderen Beispielen in dieser Arbeit um Ziel/Transitions-Netze ohne Kapazitäten.

3.6.3 Vollständige Bindung

Eine weitere Modifikation von Ziel/Transitions-Netzen betrifft die Verwendung von Variablen. Die Syntax von Ziel/Transitions-Netzen gestattet es, freie Variablen beliebig in den Termen und Formeln von Wächtern und Kantengewichten zu verwenden. Wird die gleiche Variable an verschiedenen Stellen im Umfeld einer Transition benutzt, so wird sie beim Schalten einer Transitioneninstanz überall mit dem selben Wert belegt. Dadurch lassen sich Gleichheitsbeziehungen implizit festlegen und müssen nicht explizit im Wächter einer Transition ausgedrückt werden. Es kann sinnvoll erscheinen, für Ziel/Transitions-Netze zu fordern, dass alle freien Variablen im Wächter und in den Gewichten der ausgehenden Kanten einer Transition auch in den Gewichten der eingehenden Kanten enthalten sein müssen. Dadurch sind die Werte, mit denen die Variablen im Umfeld einer Transition belegt werden können, an die Marken des Vorbereichs gebunden. Der inhärente Nichtdeterminismus kann auf diese Weise eingeschränkt werden, da sich die Anzahl der in Frage kommenden Belegungen, die eine Transitioneninstanz aktivieren, für gewöhnlich stark reduziert. Dies kann die Realisierung von Werkzeugen vereinfachen.

Definition 3.10 (Z/T-Netze mit vollständiger Bindung)
Sei $ZT = (N, A_N, M_0)$ ein Ziel/Transitions-Netz. ZT heißt Ziel/Transitions-Netz mit vollständiger Bindung *genau dann, wenn für alle Transitionen $t \in T_N$ gilt: $\chi(A_T(t)) \subseteq \bigcup_{p \in Ft} \chi(A_F(p,t))$ und $\bigcup_{p \in tF} \chi(A_F(t,p)) \subseteq \bigcup_{p \in Ft} \chi(A_F(p,t))$.*

Das in Abbildung 3.2 dargestellte Netz zeigt ein Ziel/Transitions-Netz, das nicht vollständig gebunden ist. Die Belegung der Variablen y im ausgehenden Bogen von t_1 ist nicht von der Belegung im Vorbereich abhängig. Da es sich um eine Variable handelt, die natürliche Zahlen repräsentiert, kommen in den entsprechenden Markierungen unendlich viele aktivierte Transitioneninstanzen in Frage. Ob und mit welcher Transitioneninstanz das Netz schaltet, wird nichtdeterministisch entschieden. Auf diese Weise wird ein Zufallsgenerator realisiert, der bei jedem Schaltvorgang willkürlich eine Zahl aus der Menge der natürlichen Zahlen auswählt und die Variable mit ihr belegt. In einem Ziel/Transitions-Netz mit vollständiger Bindung wird dieser Nichtdeterminismus eingeschränkt, da die Variablenbelegungen im Wächter und in den ausgehenden Kanten einer Transition von den Variablenbelegungen der eingehenden Kanten abhängig sind. Mit Ausnahme von Abbildung 3.2 handelt es sich bei allen anderen Beispielen in dieser Arbeit um Ziel/Transitions-Netze mit vollständiger Bindung.

3.7 Anwendungsbeispiel

In diesem Abschnitt wird ein einfacher Agent mit dynamischen Zielen als Ziel/-Transitions-Netz modelliert. Als Beispiel dient eine einfache Variante des klassischen BlocksWorld-Szenarios aus [Woo02]. In diesem Szenario existieren unterschiedliche Klötzchen, die aufeinander gestapelt werden können. Die Aufgabe des Agenten besteht darin, durch Auf- und Abstellen von Klötzchen eine vordefinierte Anordnung zu erzeugen. Dabei soll er jedoch das unnötige Umstellen von Klötzchen vermeiden, d.h. die Lösung soll nicht darin bestehen, zunächst alle vorhandenen Stapel abzubauen. Darüber hinaus wird in diesem Beispiel davon ausgegangen, dass jeder Stapel aus nicht mehr als zwei Klötzchen bestehen kann.

Abbildung 3.13 zeigt ein Ziel/Transitions-Netz, das einen BlocksWorld-Agenten und seine Umgebung modelliert. Die Umgebung des Agenten wird durch die Prädikat-Stelle A repräsentiert, in der die aktuelle Anordnung der Klötzchen fixiert ist. Ein Tupel (x, y) bedeutet, dass das Klötzchen x auf dem Klötzchen y gestapelt wurde. Existiert für ein Klötzchen x kein entsprechendes Tupel, dann befindet sich x auf dem Boden. Die (Teil-)Ziele des Agenten werden durch positive und negative Literale auf der Ziel-Stelle Z festgehalten und beschreiben die gewünschte Anordnung der Klötzchen. Positive Literale legen fest, welche Klötzchen aufeinander zu stapeln sind. Negative Literale legen fest, welche Klötzchen nicht aufeinander gestapelt werden dürfen. Als Zielvorgabe sind dabei auch unvollständige Konfigurationsbeschreibungen zulässig. Das Aufstellen eines Klötzchens findet durch Schalten der Transition *up* statt. Sie entfernt ein Literal der Form $A(x, y)$ von der Ziel-Stelle Z und stellt das Klötzchen x auf das Klötzchen y, indem das Tupel (x, y) auf A produziert wird. Der Wächter von *up* garantiert, dass ein Stapelvorgang nur dann durchgeführt werden kann, wenn x und y nicht verstellt sind und y nicht bereits auf ein anderes Klötzchen gestellt wurde. Die Transition *down* baut einen vorhandenen Stapel (x, y) wieder ab, wenn ein Literal der Form $\neg A(x, y)$ auf Z vorliegt. Dabei wird das Literal mit dem Schalten von *down* ebenfalls wieder von Z entfernt. Sollte ein beliebiges Ziel auf Z bereits erfüllt sein, dann kann es vom Agenten ignoriert und mit dem Schalten der Transition *ignore* gelöscht werden. Die Transitionen $plan_1$, $plan_2$ und $plan_3$ repräsentieren mentale Prozesse des Agenten. Sie sind immer dann aktiviert, wenn ein Ziel auf Z einen Stapel (x, y) verlangt, der Stapel aber zunächst nicht erzeugt werden kann. Dies ist der Fall, wenn ein Klötzchen z das Klötzchen x oder y verstellt ($plan_1$ oder $plan_2$) oder wenn das Klötzchen y bereits auf ein Klötzchen z gestellt wurde ($plan_3$). Die Wächter der Transitio-

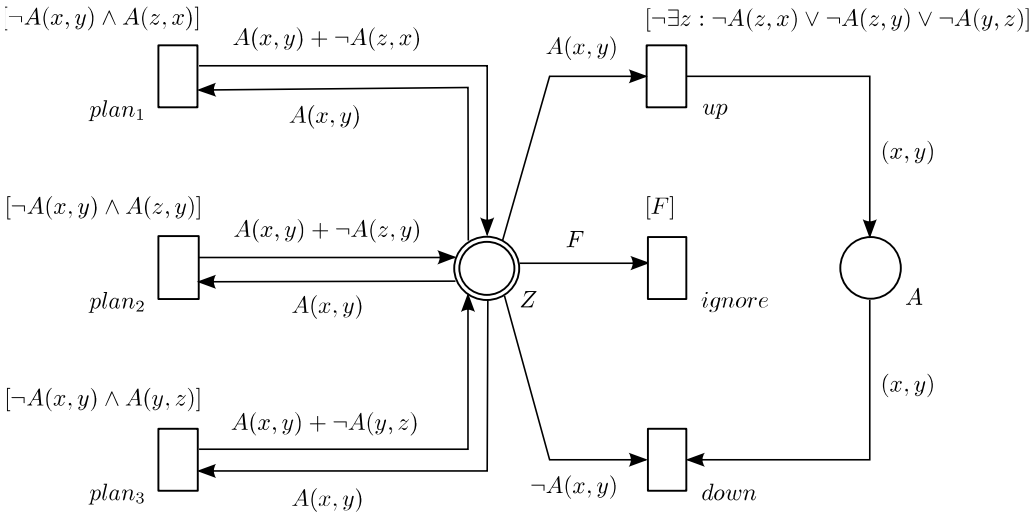


Abbildung 3.13: Ein BlocksWorld-Agent als Ziel/Transitions-Netz: Das Netz ist beschränkt und nicht lebendig für alle Anfangsmarkierungen von A und Z.

nen realisieren die notwendigen Fallunterscheidungen. Darüber hinaus setzen sie durch das Konjunktionsglied $\neg A(x, y)$ voraus, dass der geforderte Stapel noch nicht existiert. Wenn die Transition $plan_1$, $plan_2$ oder $plan_3$ schaltet, werden neue Ziele auf der Ziel-Stelle Z erzeugt, um x und y frei zu stellen. Auf diese Weise kann ein BlocksWorld-Agent ausgehend von einer beliebigen Anfangskonfiguration der Klötzchen in A nacheinander alle notwendigen Schritte durchführen, um die Zielkonfiguration in Z zu erreichen. Abbildung 3.14 zeigt einen möglichen Systemablauf für die Anfangskonfiguration $M_0(A) = (1, 2) + (3, 4) + (5, 6) + (7, 8)$ und die Zielkonfiguration $M_0(Z) = A(4, 1) + A(6, 5) + \neg A(3, 2)$.

Das Beispiel in diesem Abschnitt zeigt einige Elemente, die den Unterschied zu Prädikat/Transitions-Netzen verdeutlichen. So ist die Unterscheidung zwischen dem Aktivwerden der Transitionen $plan_1$, $plan_2$, $plan_3$ und up auf entsprechende Wächter zurückzuführen, die die Beschaffenheit der Markierung von A abfragen. Diese Abfragen werden durch die Verwendung von variablen Prädikaten realisiert und können in dieser Form nicht in Prädikat/Transitions-Netzen ausgedrückt werden. Auch die Darstellung der dynamischen Ziele in Form von positiven und negativen Literalen auf der Ziel-Stelle Z ist nur in Ziel/Transitions-Netzen möglich. Um dieses Verhalten in Prädikat/Transitions-Netzen abzubilden, sind komplexere Netz-Strukturen notwendig, d.h. es müssen viele weitere Stellen, Transitionen und Kanten eingeführt

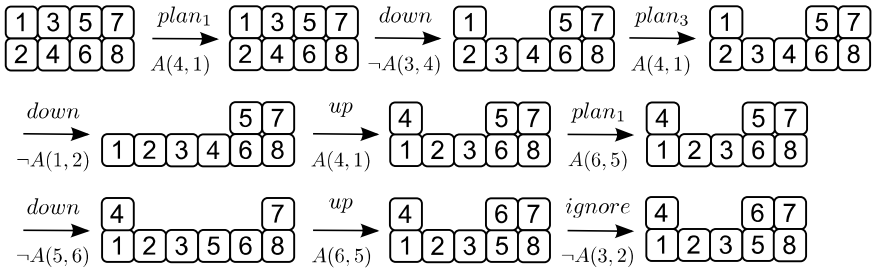


Abbildung 3.14: Ein Ablauf eines BlocksWorld-Agenten. Die gezeigten Klötzchen entsprechen der Markierung der Prädikat-Stelle A. Neben den Transitionen sind auch die Literale der Ziel-Stelle Z dargestellt, die in den einzelnen Schaltvorgängen konsumiert werden.

werden. In Ziel/Transitions-Netzen lassen sich dynamische Ziele eines Agenten stattdessen sehr einfach darstellen.

4

Modellierung von zielorientiertem Verhalten

Ziel/Transitions-Netze ermöglichen es, dynamische Ziele eines Agenten bei der Modellierung zu berücksichtigen. Ein Ziel beschreibt einen gewünschten Systemzustand und kann in Ziel/Transitions-Netzen durch eine Formel mit variablen Prädikaten repräsentiert werden. Eine solche Formel kann als Wächter oder als Marke in einem Netz verwendet werden, um das Modellverhalten zu beeinflussen. Für die Beschreibung und Analyse von Systemabläufen in Ziel/Transitions-Netzen wird in diesem Kapitel eine kausale Ablaufdarstellung entwickelt. Kausale Ablaufdarstellungen wurden bereits von Carl A. Petri verwendet, um Vorgänge in Petri-Netzen zu beschreiben [Pet62, Pet77]. Sie eignen sich als Ausgangspunkt für die Verifikation [Esp94, Rei95] und die Validierung eines Modells [Des00, Des02a, Des02b].

4.1 Grundidee

In sequentiellen Ablaufdarstellungen (z.B. Schaltfolgen) wird das Verhalten eines Modells als Folge von globalen Zuständen und Zustandsübergängen verstanden. Zu diesem Zweck wird eine Totalordnung über den Vorgängen im Modell postuliert. Eine Unterscheidung von Kausalität und Nebenläufigkeit ist bei der Betrachtung eines Ablaufes auf diese Weise nicht mehr möglich. So kann z.B. für das in Abbildung 3.5 dargestellte Netz an Hand der Schaltfolge $t_1 : \alpha_1\ t_2 : \alpha_2$ nicht entschieden werden, ob das Schalten von $t_2 : \alpha_2$ tatsächlich vom Schalten von $t_1 : \alpha_1$ abhängt

oder beide Transitioneninstanzen nebenläufig eintreten. Um diesem Problem zu begegnen, werden für Petri-Netze in der Regel kausale Ablaufdarstellungen bevorzugt. In kausalen Ablaufdarstellungen wird das Schalten von Transitionen bzw. Transitioneninstanzen während eines Ablaufs in Form von Ereignissen erfasst und die Vor- und Nachbedingungen der Ereignisse ebenfalls berücksichtigt. Vorbedingungen beschreiben die Voraussetzungen, die für das Schalten einer Transition bzw. Transitioneninstanz notwendig sind. Durch das Eintreten eines Ereignisses wird ein neuer Zustand erzeugt, in dem die Nachbedingungen des Ereignisses gelten. Diese Nachbedingungen können Vorbedingungen für das Eintreten von weiteren Ereignissen sein. Durch die Beziehungen zwischen dem Eintreten von Ereignissen und ihren Vor- und Nachbedingungen wird eine kausale Struktur festgelegt, die sich in Form eines Netzes darstellen lässt. In einem solchen Netz werden die kausalen Abhängigkeiten als Relation zwischen den Ereignissen und ihren Vor- und Nachbedingungen von entsprechenden Kanten repräsentiert. Durch den transitiven Abschluss dieser Relation entsteht eine Halbordnung, die auch Kausalordnung genannt wird: Zwei Ereignisse sind kausal voneinander abhängig genau dann, wenn zwischen ihnen ein gerichteter Pfad existiert. Ansonsten treten die Ereignisse nebenläufig ein. Ein Netz, das den Ablauf eines Petri-Netzes in dieser Form beschreibt, wird auch Prozess genannt. Bei der Analyse von Systemabläufen haben kausale Ablaufdarstellungen den Vorteil, dass eindeutig entscheidbar ist, zwischen welchen Bedingungen und Ereignissen im Ablauf ein kausaler Zusammenhang besteht und zwischen welchen nicht. In Ziel/-Transitions-Netzen ist es auf diese Weise möglich, die Wechselwirkung zwischen den Zielen und den Zuständen in einem Ablauf zu untersuchen.

Ein verbreiteter Ansatz für eine kausale Ablaufdarstellung ist die Verwendung von Kausalnetzen [GR83, Rei85, Des00, Des02a]. Ein Kausalnetz ist ein zyklenfreies Netz, das den kausalen Zusammenhang zwischen Bedingungen und Ereignissen wie folgt erfasst:

- Jede Bedingung in einem Kausalnetz repräsentiert das Vorhandensein einer Marke während des Ablaufs.
- Jedes Ereignis in einem Kausalnetz repräsentiert das Schalten einer Transition bzw. Transitioneninstanz während des Ablaufs.
- Jede Kante repräsentiert den Konsum bzw. die Produktion einer Marke. Wenn eine Transition bzw. Transitioneninstanz schaltet, verknüpfen entsprechende Kanten die Vor- und Nachbedingungen mit dem jeweiligen Ereignis.

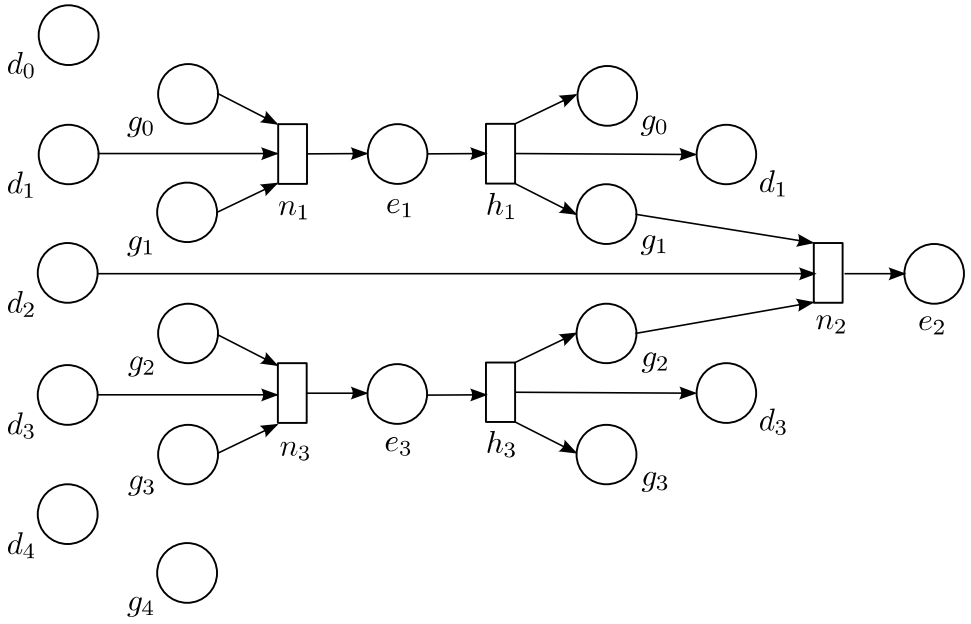


Abbildung 4.1: Ein Kausalnetz für das Petri-Netz aus Abbildung 2.6: Jede Marke während des Ablaufs muss durch eine Bedingung repräsentiert werden.

Ein Beispiel für ein Kausalnetz zeigt Abbildung 4.1. Es protokolliert einen Ablauf, der durch das Schalten der Transitionen n_1, n_3, h_1, h_3 und n_2 im Petri-Netz aus Abbildung 2.6 entsteht.

Während Kausalnetze in B/E-Systemen eine geeignete Ablaufdarstellung sein können, ergeben sich in S/T-Systemen und höheren Petri-Netzen einige entscheidende Nachteile. Weil jede Marke als Bedingung im Netz repräsentiert werden muss, können zu einem Ablauf in der Regel mehrere unterschiedliche Kausalnetze konstruiert werden. Die Netze unterscheiden sich darin, welche Marken als Vor- und Nachbedingungen für die Ereignisse ausgewählt wurden, wenn in einer Markierung mehrere Marken auf einer Stelle zur Verfügung stehen. Die Konstruktion verliert somit an Eindeutigkeit, weil völlig identischen Marken eine Individualität zugesprochen wird, die ihnen in Wirklichkeit nicht zukommt [Cze97]. Darüber hinaus neigen Kausalnetze zu starkem Wachstum, wenn die Zahl der Marken, die im Netz produziert und konsumiert werden, zunimmt.

Eine Alternative zu Kausalnetzen besteht in der Verwendung von Protokollnetzen [Cze97]. Protokollnetze bestehen ebenfalls aus Bedingungen und Ereignissen (genannt B-Elemente bzw. E-Elemente in [Cze97]). Bedingungen stellen allerdings kei-

ne Marken dar, sondern Markierungen von Stellen. Sie werden als Vor- und Nachbedingungen mit den Ereignissen im Protokollnetz verknüpft. Eine Vorbedingung repräsentiert die Markierung einer Stelle bevor das entsprechende Ereignis stattfindet, eine Nachbedingung repräsentiert die Markierung einer Stelle nachdem das entsprechende Ereignis stattgefunden hat. Die Stellen im Einflussbereich einer Transition werden in Protokollnetzen also doppelt berücksichtigt: einmal im Vorbereich und einmal im Nachbereich von Ereignissen. Verzweigungen von Bedingungen beschreiben in diesem Fall den nebenläufigen Zugriff auf die Markierung einer Stelle. Bei der Verwendung von Protokollnetzen ist gewährleistet, dass jeder Ablauf eine eindeutige Darstellung besitzt. Darüber hinaus verlieren Protokollnetze auch mit zunehmender Anzahl von Marken nicht ihre Kompaktheit. Aus diesen Gründen soll in den folgenden Abschnitten die Idee von Protokollnetzen, die als Ablaufdarstellung für S/T-Systeme vorgeschlagen wurden, auf Ziel/Transitions-Netze übertragen werden.

4.2 Protokollnetze

Die Konstruktion eines Protokollnetzes wird induktiv auf der Grundlage einer Multischrittfolge erklärt. Ein Multischritt beschreibt eine Multimenge von Transitioneninstanzen, die in einer Markierung nebenläufig schalten. Eine Multischrittfolge ist eine Sequenz von Multischritten.

Definition 4.1 (Multischritt, Multischrittfolge)
Sei $ZT = (N, A_N, M_0)$ ein Ziel/Transitions-Netz und M eine Markierung von ZT.

(1.) Eine Multimenge G, die Transitioneninstanzen auf natürliche Zahlen abbildet, heißt genau dann Multischritt *in M, wenn G nebenläufig ist bei M.*

(2.) Das Schalten eines Multischrittes G bei M erzeugt eine Folgemarkierung M' nach folgender Vorschrift:

(2.1.) $M'(p) = M(p) - \sum_{t:\alpha \in G} G(t:\alpha) \cdot \varepsilon(A_F(p,t):\alpha) + \sum_{t:\alpha \in G} G(t:\alpha) \cdot \varepsilon(A_F(t,p):\alpha)$ *für alle* $p \in P_N$.

(2.2.) $M'(\widehat{p}) = M(\widehat{p}) - \sum_{t:\alpha \in G} G(t:\alpha) \cdot \sigma(A_F(\widehat{p},t):\alpha) + \sum_{t:\alpha \in G} G(t:\alpha) \cdot \sigma(A_F(t,\widehat{p}):\alpha)$ *für alle* $\widehat{p} \in \widehat{P}_N$.

Das Schalten eines Multischrittes G von M nach M' wird mit $M \xrightarrow{G}_{ZT} M'$ notiert. Geht ZT aus dem Kontext hervor, kann auf den Index verzichtet werden.

(3.) Eine Folge von Multischritten $H = G_1...G_k$ heißt genau dann Multischrittfolge *auf ZT, wenn eine Menge von Markierungen $M_1, ..., M_k \in R_{ZT}$ existiert, so dass $M_{i-1} \xrightarrow{G_i}_{ZT} M_i$ für alle $i \in \{1..k\}$.*

Die Konstruktion eines Protokollnetzes beginnt mit der Darstellung der Anfangsmarkierung. Anschließend wird durch Induktion festgelegt, wie ein bereits konstruierter Teil eines Protokollnetzes um den nächsten Multischritt erweitert wird. Die folgende Definition ist bewusst möglichst nahe an [Cze97] angelehnt. Allerdings werden in dieser Arbeit ausschließlich *Vorwärtsschritte* betrachtet und *Rückwärtsschritte* bei der Beschreibung von Abläufen vernachlässigt. Die Konstruktion besitzt also einen eindeutigen Anfang (*fundierte* Protokollnetze) und findet immer nach »rechts« statt. Darüber hinaus ergeben sich aufgrund der Berücksichtigung von variablen Prädikaten in den Wächtern von Transitionen Änderungen in der Konstruktionsvorschrift.

Definition 4.2 (Protokollnetz)
Sei $ZT = (N, A_N, M_0)$ ein Ziel/Transitions-Netz und $H = G_1...G_k$ eine Multischrittfolge auf ZT mit $M_0 \xrightarrow{G_1}_{ZT} M_1 \xrightarrow{G_2}_{ZT} ... \xrightarrow{G_k}_{ZT} M_k$.

(1.) ***Induktionsbeginn:*** *Der leeren Multischrittfolge wird das Konstrukt $PN_0 = (PB_0, PE_0, PF_0, PZ_0, PP_0)$ zugeordnet mit den Mengen:*

(1.1.) $PB_0 = B_0$ mit $B_0 = \{(p, 0) \mid p \in P_N \cup \widehat{P}_N\}$,

(1.2.) $PE_0 = E_0$ mit $E_0 = \emptyset$,

(1.3.) $PF_0 = F_0$ mit $F_0 = \emptyset$

und den Abbildungen:

(1.4.) $PZ_0 = Z_0$ mit $Z_0(p, 0) = M_0(p)$ für alle $(p, 0) \in B_0$,

(1.5.) $PP_0 = P_0$ mit $P_0(p, 0) = p$ für alle $(p, 0) \in B_0$.

(2.) ***Induktionsschritt:*** *Das Konstrukt $PN_n = (PB_n, PE_n, PF_n, PZ_n, PP_n)$ wird durch den Multischritt G_{n+1} zum Konstrukt $PN_{n+1} = (PB_{n+1}, PE_{n+1}, PF_{n+1}, PZ_{n+1}, PP_{n+1})$ erweitert mit:*

(2.1.) $PB_{n+1} = PB_n \cup B_{n+1}$ mit $B_{n+1} =$
$\{(p, n+1) \mid p \in \bigcup_{t:\alpha \in G_{n+1}} (F_N t \cup t F_N)\} \cup$
$\{(p, n+1) \mid A_P(p) \in \bigcup_{t:\alpha \in G_{n+1}} \{A_T(t) : \alpha\}\}$,

(2.2.) $PE_{n+1} = PE_n \cup E_{n+1}$ *mit* $E_{n+1} =$
$\{(t : \alpha, n+1) \mid t : \alpha \in G_{n+1}\}$,

(2.3.) $PF_{n+1} = PF_n \cup F_{n+1}$ *mit* $F_{n+1} =$
$\{((t : \alpha, n+1), (p, n+1)) \mid t : \alpha \in G_{n+1} \wedge p \in F_N t \cup t F_N\} \cup$
$\{((t : \alpha, n+1), (p, n+1)) \mid t : \alpha \in G_{n+1} \wedge A_P(p) \in A_T(t) : \alpha\} \cup$
$\{((p, i), (t : \alpha, n+1)) \mid t : \alpha \in G_{n+1} \wedge p \in F_N t \cup t F_N \wedge$
$i = max\{j \mid (p, j) \in PB_n\}\} \cup$
$\{((p, i), (t : \alpha, n+1)) \mid t : \alpha \in G_{n+1} \wedge A_P(p) \in A_T(t) : \alpha \wedge$
$i = max\{j \mid (p, j) \in PB_n\}\}$,

(2.4.) $PZ_{n+1} = PZ_n \cup Z_{n+1}$ *mit*
$Z_{n+1}(p, n+1) = M_{n+1}(p)$ *für alle* $(p, n+1) \in B_{n+1}$ *und*
$Z_{n+1}(t : \alpha, n+1) = G_{n+1}(t)$ *für alle* $(t : \alpha, n+1) \in E_{n+1}$,

(2.5.) $PP_{n+1} = PP_n \cup P_{n+1}$ *mit*
$P_{n+1}(p, n+1) = p$ *für alle* $(p, n+1) \in B_{n+1}$ *und*
$P_{n+1}(t : \alpha, n+1) = t : \alpha$ *für alle* $(t : \alpha, n+1) \in E_{n+1}$.

(3.) $PN = (PB, PE, PF, PZ, PP)$ *heißt* Protokollnetz *über* H *auf* ZT *genau dann, wenn* $PB = PB_k$, $PE = PE_k$, $PF = PF_k$, $PZ = PZ_k$ *und* $PP = PP_k$. *Elemente aus* PB *heißen* Bedingungen *und Elemente aus* PE *heißen* Ereignisse.

Bei der graphischen Darstellung eines Protokollnetzes werden die Elemente aus PB als Kreise und die Elemente aus PE als Rechtecke gezeichnet. Die durch PF protokollierten Zusammenhänge zwischen den Bedingungen und den Ereignissen werden als unbeschriftete Pfeile dargestellt, während die Bilder von PZ als Beschriftungen in den Kreisen und Rechtecken verwendet werden. In Anlehnung an die Notation von Ziel/Transitions-Netzen werden die Bedingungen, die Markierungen von Ziel-Stellen repräsentieren, durch einen doppelten Rand hervorgehoben.

Abbildung 4.2 zeigt ein Beispiel für ein Protokollnetz von Abbildung 3.5. Anhand der Multischrittfolge $H_1 = \{t_1 : \alpha_1\}\ \{t_2 : \alpha_3\}\ \{t_3 : \alpha_5\}$ können folgende Element- und Abbildungsmengen konstruiert werden:

(1.) **n = 0:**
$B_0 = \{(A, 0), (B, 0), (C, 0), (D, 0), (E, 0), (Z, 0)\}$,
$E_0 = \emptyset$,
$F_0 = \emptyset$,

$Z_0(A,0) = (1), Z_0(B,0) = 0, Z_0(C,0) = (0),$
$Z_0(D,0) = 0, Z_0(E,0) = 0, Z_0(Z,0) = A(1) + C(0).$

(2.) **n = 1:**
$B_1 = \{(A,1),(B,1),(Z,1)\},$
$E_1 = \{(t_1 : \alpha_1, 1)\},$
$F_1 = \{((t_1 : \alpha_1, 1),(A,1)),((t_1 : \alpha_1, 1),(B,1)),$
$((t_1 : \alpha_1, 1),(Z,1)),((A,0),(t_1 : \alpha_1, 1)),$
$((B,0),(t_1 : \alpha_1, 1)),((Z,0),(t_1 : \alpha_1, 1))\},$
$Z_1(A,1) = 0, Z_1(B,1) = (1), Z_1(Z,1) = B(1) + C(0).$

(3.) **n = 2:**
$B_2 = \{(C,2),(D,2),(Z,2)\},$
$E_2 = \{(t_2 : \alpha_3, 2)\},$
$F_2 = \{((t_2 : \alpha_3, 2),(C,2)),((t_2 : \alpha_3, 2),(D,2)),$
$((t_2 : \alpha_3, 2),(Z,2)),((C,0),(t_2 : \alpha_3, 2)),$
$((D,0),(t_2 : \alpha_3, 2)),((Z,1),(t_2 : \alpha_3, 2))\},$
$Z_2(C,2) = 0, Z_2(D,2) = (0), Z_2(Z,2) = B(1) + D(0).$

(4.) **n = 3:**
$B_3 = \{(B,3),(D,3),(E,3),(Z,3)\},$
$E_3 = \{(t_3 : \alpha_5, 3)\},$
$F_3 = \{((t_3 : \alpha_5, 3),(B,3)),((t_3 : \alpha_5, 3),(D,3)),$
$((t_3 : \alpha_5, 3),(E,3)),((t_3 : \alpha_5, 3),(Z,3)),$
$((B,1),(t_3 : \alpha_5, 3)),((D,2),(t_3 : \alpha_5, 3)),$
$((E,0),(t_3 : \alpha_5, 3)),((Z,2),(t_3 : \alpha_5, 3))\},$
$Z_3(B,3) = 0, Z_3(D,3) = 0, Z_3(E,3) = (1,0), Z_3(Z,3) = E(1,0).$

Abbildung 4.3 zeigt ein weiteres Beispiel für ein Protokollnetz von Abbildung 3.5. Anhand der Multischrittfolge $H_2 = \{t_1 : \alpha_1, t_2 : \alpha_3\}\ \{t_3 : \alpha_5\}\ \{t_4 : \alpha_6\}$ können folgende Element- und Abbildungsmengen konstruiert werden:

(1.) **n = 0:**
$B_0 = \{(A,0),(B,0),(C,0),(D,0),(E,0),(Z,0)\},$
$E_0 = \emptyset,$
$F_0 = \emptyset,$
$Z_0(A,0) = (1), Z_0(B,0) = 0, Z_0(C,0) = (0),$
$Z_0(D,0) = 0, Z_0(E,0) = 0, Z_0(Z,0) = A(1) + C(0).$

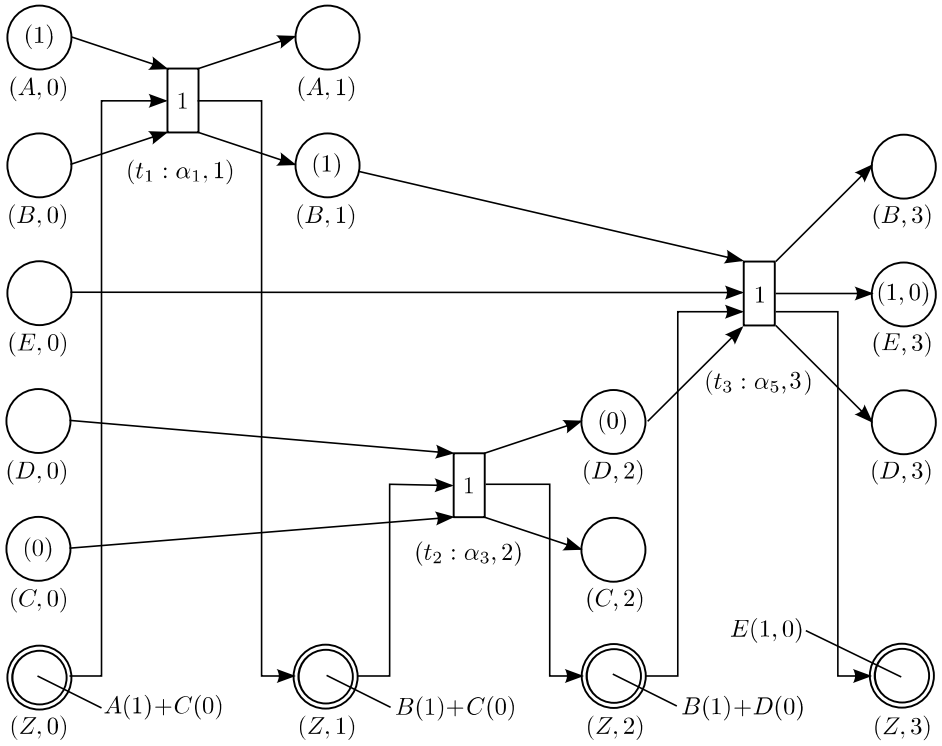


Abbildung 4.2: Protokollnetz über H_1 auf dem Ziel/Transitions-Netz aus Abbildung 3.5.

(2.) **n = 1:**
$B_1 = \{(A,1),(B,1),(C,1),(D,1),(Z,1)\}$,
$E_1 = \{(t_1 : \alpha_1, 1),(t_2 : \alpha_3, 1)\}$,
$F_1 = \{((t_1 : \alpha_1, 1),(A,1)),((t_1 : \alpha_1, 1),(B,1)),$
$((t_1 : \alpha_1, 1),(Z,1)),((t_2 : \alpha_3, 1),(C,1)),$
$((t_2 : \alpha_3, 1),(D,1)),((t_2 : \alpha_3, 1),(Z,1)),$
$((A,0),(t_1 : \alpha_1, 1)),((B,0),(t_1 : \alpha_1, 1)),$
$((Z,0),(t_1 : \alpha_1, 1)),((C,0),(t_2 : \alpha_3, 1)),$
$((D,0),(t_2 : \alpha_3, 1)),((Z,0),(t_2 : \alpha_3, 1))\}$,
$Z_1(A,1) = 0, Z_1(B,1) = (1), Z_1(C,1) = 0,$
$Z_1(D,1) = (0), Z_1(Z,1) = B(1) + D(0).$

(3.) **n = 2:**
$B_2 = \{(B,2),(D,2),(E,2),(Z,2)\}$,
$E_2 = \{(t_3 : \alpha_5, 2)\}$,
$F_2 = \{((t_3 : \alpha_5, 2),(B,2)),((t_3 : \alpha_5, 2),(D,2)),$

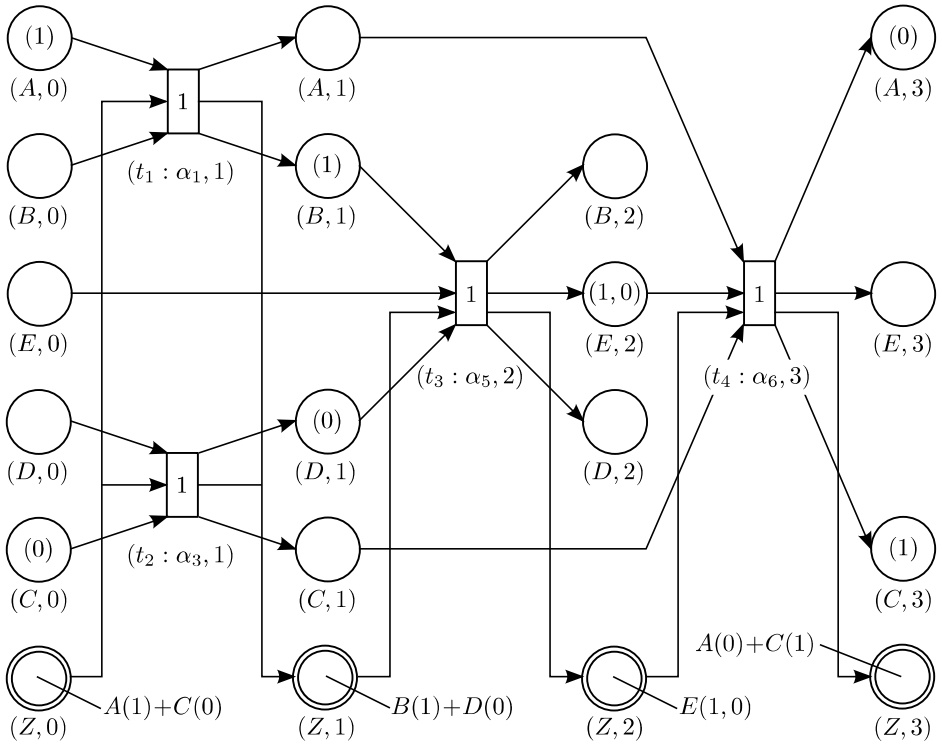


Abbildung 4.3: Protokollnetz über H_2 auf dem Ziel/Transitions-Netz aus Abbildung 3.5.

$((t_3 : \alpha_5, 2), (E, 2)), ((t_3 : \alpha_5, 2), (Z, 2)),$
$((B, 1), (t_3 : \alpha_5, 2)), ((D, 1), (t_3 : \alpha_5, 2)),$
$((E, 0), (t_3 : \alpha_5, 2)), ((Z, 1), (t_3 : \alpha_5, 2))\},$
$Z_2(B, 2) = 0, Z_2(D, 2) = 0, Z_2(E, 2) = (1, 0), Z_2(Z, 2) = E(1, 0).$

(4.) **n = 3:**
$B_2 = \{(A, 3), (C, 3), (E, 3), (Z, 3)\},$
$E_2 = \{(t_4 : \alpha_6, 3)\},$
$F_2 = \{((t_4 : \alpha_6, 3), (A, 3)), ((t_4 : \alpha_6, 3), (C, 3)),$
$((t_4 : \alpha_6, 3), (E, 3)), ((t_4 : \alpha_6, 3), (Z, 3)),$
$((A, 1), (t_4 : \alpha_6, 3)), ((C, 1), (t_4 : \alpha_6, 3)),$
$((E, 2), (t_4 : \alpha_6, 3)), ((Z, 2), (t_4 : \alpha_6, 3))\},$
$Z_3(A, 3) = (0), Z_3(C, 3) = (1), Z_3(E, 3) = 0, Z_3(Z, 3) = A(0) + C(1).$

Theorem 4.1
Zu jeder Multischrittfolge existiert genau ein Protokollnetz und zu jedem Protokollnetz existiert genau eine Multischrittfolge.

Beweis: Nach Definition 4.2 wird zu jeder Multischrittfolge genau ein Protokollnetz konstruiert. Weil die Indizes der Multischritte darüber hinaus in den Ereignissen eines Protokollnetzes enthalten sind, führen verschiedene Multischrittfolgen immer zu verschiedenen Protkollnetzen. □

Definition 4.3 (Minimale und maximale Elemente)
Sei $PN = (PB, PE, PF, PZ, PP)$ ein Protokollnetz und PN_n das im Konstruktionsschritt $n \in \mathbb{N}$ entstandene Konstrukt von PN. Die Menge der minimalen Elemente *von PN_n ist gegeben durch $\mathrm{Min}PN_n = B_0$. Die Menge der* maximalen Elemente *von PN_n ist gegeben durch $\mathrm{Max}PN_n = B_0$ für $n = 0$ und $\mathrm{Max}PN_n = B_n \cup (\mathrm{Max}PN_{n-1} \setminus \bigcup_{(t:\alpha,n)\in E_n} PF_n(t:\alpha,n))$ für $n > 0$.*

4.3 Grundeigenschaften

Aus der Konstruktionsvorschrift lassen sich die Grundeigenschaften von Protokollnetzen ableiten.

Lemma 4.1
Sei $PN = (PB, PE, PF, PZ, PP)$ ein Protokollnetz und PN_n das im Konstruktionsschritt $n \in \mathbb{N}$ entstandene Konstrukt von PN.

(1.) Bedingungen und Ereignisse sind endliche und disjunkte Mengen:
PE_n und PB_n endlich, $PE_n \cap PB_n = \emptyset$.

(2.) Die Vor- und Nachbereiche aller Bedingungen und Ereignisse sind endlich und mit Ausnahme von $\mathrm{Min}PN_n$ und $\mathrm{Max}PN_n$ nicht leer:

(2.1.) $\forall (t:\alpha, l) \in PE_n : PF_n(t:\alpha, l)$ *endlich* $\wedge\ PF_n(t:\alpha, l) = \emptyset$,

(2.2.) $\forall (t:\alpha, l) \in PE_n : (t:\alpha, l)PF_n$ *endlich* $\wedge\ (t:\alpha, l)PF_n = \emptyset$.

(2.3.) $\forall (p, i) \in PB_n \setminus \mathrm{Min}PN_n : PF_n(p, i)$ *endlich*
$\wedge\ PF_n(p, i) = \emptyset$,

(2.4.) $\forall (p, i) \in PB_n \setminus \mathrm{Max}PN_n : (p, i)PF_n$ *endlich*
$\wedge\ (p, i)PF_n = \emptyset$,

(2.5.) $\forall (p,i) \in \mathrm{Min}PN_n : PF_n(p,i) = \emptyset$,

(2.6.) $\forall (p,i) \in \mathrm{Max}PN_n : (p,i)PF_n = \emptyset$.

(3.) *Zwei Bedingungen auf einem Pfad sind über mindestens ein Ereignis miteinander verbunden, und zwei Ereignisse auf einem Pfad sind über mindestens eine Bedingung miteinander verbunden:*

(3.1.) $\forall (p,i),(p',j) \in PB_n : ((p,i){PF_n}^+(p',j) \Rightarrow$
$\exists (t:\alpha,l) \in PE_n : (p,i){PF_n}^+(t:\alpha,l){PF_n}^+(p',j))$.

(3.2.) $\forall (t:\alpha,j),(t':\alpha',l) \in PE_n : ((t:\alpha,j){PF_n}^+(t':\alpha',l) \Rightarrow$
$\exists (p,i) \in PB_n : (t:\alpha,j){PF_n}^+(p,i){PF_n}^+(t':\alpha',l))$.

(4.) *Zu jeder Vorbedingung eines Ereignisses existiert eine Nachbedingung, und zu jeder Nachbedingung eines Ereignisses existiert eine Vorbedingung:*

(4.1.) $\forall (t:\alpha,l) \in PE_n : \forall (p,i) \in PF_n(t:\alpha,l) : \exists (p,j) \in PB_n :$
$(p,j) \in (t:\alpha,l)PF_n$,

(4.2.) $\forall (t:\alpha,l) \in PE_n : \forall (p,i) \in (t:\alpha,l)PF_n : \exists (p,j) \in PB_n :$
$(p,j) \in PF_n(t:\alpha,l)$.

(5.) *Der Index eines Ereignisses ist gleich dem Index seiner Nachbedingungen und größer als der Index seiner Vorbedingungen:*

(5.1.) $\forall (t:\alpha,l) \in PE_n : \forall (p,i) \in (t:\alpha,l)PF_n : i = l$,

(5.2.) $\forall (t:\alpha,l) \in PE_n : \forall (p,i) \in PF_n(t:\alpha,l) : i < l$.

(6.) *Die Indizes der Vorbedingungen eines Ereignisses sind maximal für alle Bedingungen, die vor dem Ereignis vorkommen:*
$\forall (t:\alpha,l) \in E_n : \forall (p,i) \in PF_n(t:\alpha,l) : \forall (p,j) \in PB_{n-1} : i \geq j$.

(7.) *Alle Ereignisse im Vorbereich einer Bedingung haben den selben Index und alle Ereignisse im Nachbereich einer Bedingung haben den selben Index:*

(7.1.) $\forall (p,i) \in PB_n : \forall (t:\alpha,j),(t':\alpha',l) \in PF_n(p,i) : j = l$.

(7.2.) $\forall (p,i) \in PB_n : \forall (t:\alpha,j),(t':\alpha',l) \in (p,i)PF_n : j = l$.

(8.) *Die Indizes werden nicht kleiner entlang eines Pfades:*
$\forall (x,i),(y,j) \in PB_n \cup PE_n : (x,i){PF_n}^+(y,j) \Rightarrow i \leq j$.

(9.) Vor- und Nachbereich eines Ereignisses haben die selbe Mächtigkeit: $\forall(t:\alpha,l) \in PE_n : |PF_n(t:\alpha,l)| = |(t:\alpha,l)PF_n|$.

(10.) PN_n *ist zyklenfrei:* $\forall(x,i) \in PB_n \cup PE_n : \neg(x,i){PF_n}^+(x,i)$.

(11.) $(PB_n \cup PE_n, {PF_n}^*)$ *ist eine Halbordnung.*

Beweis: Die Eigenschaften (1.) bis (6.) folgen unmittelbar aus Definition 4.2, insbesondere aus (2.3.).
Zu (7.1.): Angenommen, für zwei Ereignisse $(t:\alpha,j),(t':\alpha',l)$ im Vorbereich einer Bedingung (p,i) ist $j = l$. Widerspruch zu (5.1.), wonach $j = i$ und $l = i$ ist und somit $j = l$.
Zu (7.2.): Induktionsanfang: Nach (1.2.) in Definition 4.2 ist $PE_n = \emptyset$ bei $n = 0$ und die Behauptung gilt. Induktionsschritt: Die Behauptung gelte für PN_n mit $n \geq 0$. Die Annahme, für zwei beliebige Ereignisse $(t:\alpha,j),(t':\alpha',l) \in PE_{n+1}$ im Nachbereich einer Bedingung $(p,i) \in PB_{n+1}$ ist $j = l$, führt in allen Fällen zum Widerspruch. 1. Fall: $(t:\alpha,j),(t':\alpha',l) \in PE_n$. Widerspruch zur Induktionsvoraussetzung, wonach $j = l$ ist. 2. Fall: $(t:\alpha,j),(t':\alpha',l) \in E_{n+1}$. Widerspruch zu (2.2.) in Definition 4.2, wonach $j = l = n+1$ ist. 3. Fall: Sei o.B.d.A. $(t:\alpha,j) \in PE_n$ und $(t':\alpha',l) \in E_{n+1}$. In diesem Fall existiert für $(t:\alpha,j)$ nach (4.1.) eine Nachbedingung $(p,k) \in PB_n$. Nach (5.) ist $k = j$ sowie $i < j$ und somit $i < k$. Widerspruch zu (6.), wonach $i \geq k$ ist, da (p,i) Vorbedingung von $(t':\alpha',l)$ ist.
Zu (8.): Nach (5.) ist $i \leq j$ für zwei beliebige $(x,i),(y,j) \in PB_n \cup PE_n$ mit $(x,i)PF_n(y,j)$. Aufgrund der definitionsgemäßen Transitivität von ${PF_n}^+$ ist damit $i \leq j$ für zwei beliebige $(x,i),(y,j) \in PB_n \cup PE_n$ mit $(x,i){PF_n}^+(y,j)$.
Zu (9.): Nach (2.3.) in Definition 4.2 existiert für jeden Fall, in dem ein Ereignis mit einer Nachbedingung verbunden wird, genau ein Fall, der das selbe Ereignis mit einer Vorbedingung verbindet, und umgekehrt.
Zu (10.): Angenommen, $(x,i){PF_n}^+(y,j){PF_n}^+(x,i)$ ist ein Zyklus in PN_n. Wegen (8.) ist daher $i = j$. Nach (5.) ist eine Verbindung zwischen zwei Knoten mit gleichen Indizes nur von einem Ereignis zu einer Bedingung möglich. Dann ist einerseits (x,i) ein Ereignis und (y,j) eine Bedingung, anderseits ist (y,j) ein Ereignis und (x,i) eine Bedingung sein. Widerspruch zu (1.), wonach Bedingungen und Ereignisse disjunkte Mengen sind.
Zu (11.): ${PF_n}^*$ ist defintionsgemäß reflexiv und transitiv. Die Antisymmetrie von ${PF_n}^*$ folgt aus (10.). □

4.4 Linien

Durch ein Protokollnetz PN bzw. ein Konstrukt PN_n ist eine kausale Ordnung über den Bedingungen und Ereignissen eines Ablaufs festgelegt. Kausal abhängige Elemente bilden so genannte *Ketten*, wobei maximale Ketten zu *Linien* zusammengefasst werden.

Definition 4.4 (Kette, Linie)
Sei $PN = (PB, PE, PF, PZ, PP)$ ein Protokollnetz und PN_n das im Konstruktionsschritt $n \in \mathbb{N}$ entstandene Konstrukt von PN. Seien $(x, i), (y, j) \in PB_n \cup PE_n$ zwei beliebige Elemente von PN_n.

(1.) Die Relation $\mathrm{li}_n \subseteq (PB_n \cup PE_n) \times (PB_n \cup PE_n)$ *ist gegeben durch:* $(x, i)\mathrm{li}_n(y, j) \Leftrightarrow (x, i){PF_n}^+(y, j) \vee (y, j){PF_n}^+(x, i) \vee (x, i) = (y, j)$.

(2.) Eine Menge $L_n \subseteq PB_n \cup PE_n$ heißt Kette *genau dann, wenn* $\forall (x, i), (y, j) \in L_n : (x, i)\mathrm{li}_n(y, j)$.

(3.) Eine Menge $L_n \subseteq PB_n \cup PE_n$ heißt Linie *genau dann, wenn L_n eine Kette ist und* $\forall (x, i) \in (PB_n \cup PE_n) \setminus L_n : \exists (y, j) \in L_n : \neg(x, i)\mathrm{li}_n(y, j)$.

Die Relation li_n identifiziert kausal abhängige Elemente. Ketten bzw. Linien beschreiben Mengen bzw. maximale Mengen von Bedingungen und Ereignissen, die während eines Ablaufs in einem Ursache-Wirkungs-Zusammenhang stehen. Ein Beispiel für eine Linie im Protokollnetz von Abbildung 4.2 ist in Abbildung 4.4 dargestellt. Alle Elemente aus $L = \{(B, 0), (t_1 : \alpha_1, 1), (B, 1), (t_3 : \alpha_5, 3), (B, 3)\}$ sind kausal voneinander abhängig: Die leere Markierung von B am Ende des Ablaufs ist eine unmittelbare Folge des Schaltens der Transitioneninstanz $t_3 : \alpha_5$ im letzten Multischritt. Für das Schalten von $t_3 : \alpha_5$ wurde eine entsprechende Marke auf B vorausgesetzt, die im ersten Multischritt von $t_1 : \alpha_1$ erzeugt wurde. Vor dem Schalten von $t_1 : \alpha_1$ zu Beginn des Ablaufs war die Markierung von B leer (siehe Abbildung 3.5).

Im Gegensatz zu Kausalnetzen sind in Protokollnetzen Verzweigungen von Bedingungen möglich. Eine Bedingung kann also direkt von mehreren Ereignissen abhängig sein, während mehrere Ereignisse direkt von einer Bedingung abhängig sein können. Ein Beispiel zeigt Abbildung 4.3, in der die Bedingungen $(Z, 0)$ und $(Z, 1)$ durch die Ereignisse $(t_1 : \alpha_1, 1)$ und $(t_2 : \alpha_3, 1)$ verzweigt sind. In Protokollnetzen entstehen Verzweigungen von Bedingungen aber nur dann, wenn die entsprechenden

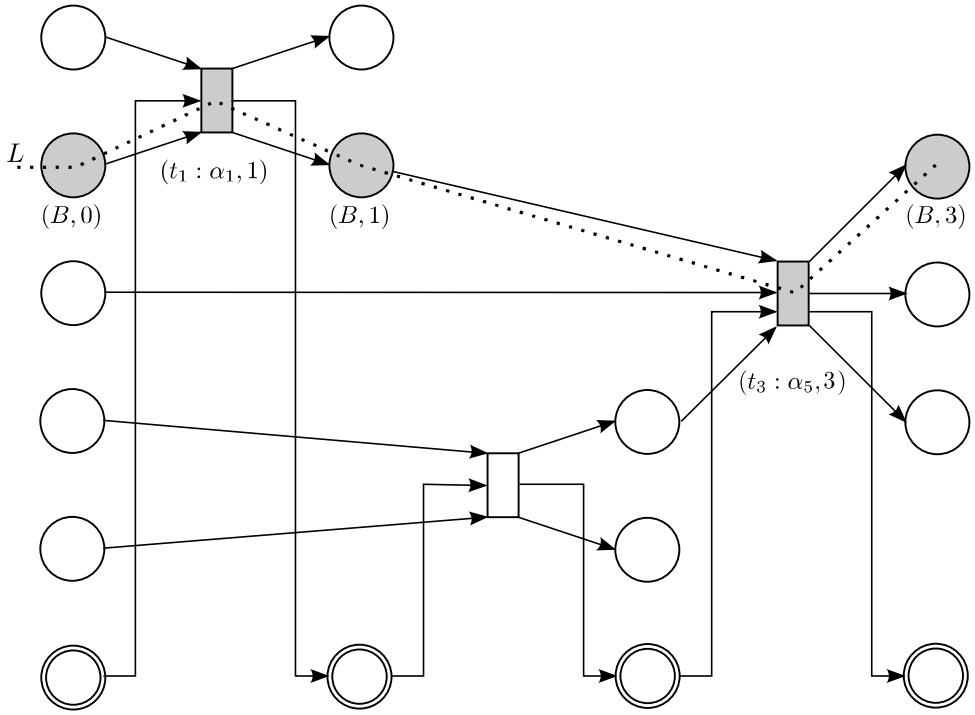


Abbildung 4.4: Eine Linie im Protokollnetz aus Abbildung 4.2.

Ereignisse aus dem selben Multischritt stammen (siehe (7.1.) und (7.2.) in Lemma 4.1) und damit einen nebenläufigen Zugriff auf die Stelle beschreiben.

Kausale Abhängigkeiten zwischen den Bedingungen und Ereignissen eines Protokollnetzes ergeben sich aus der Produktion und dem Konsum von Marken während des Ablaufs. Aufgrund der Verwendung von variablen Prädikaten in Wächtern von Transitionen können kausale Abhängigkeiten in Ziel/Transitions-Netzen aber auch unabhängig von der Bewegung von Marken entstehen, wenn die Aktivierung einer Transitioneninstanz von der Markierung einer Stelle abhängt, zu der keine direkte Verbindung existiert. Ein Beispiel zeigt Abbildung 4.5. Das Protokollnetz wurde auf dem Ziel/Transitions-Netz aus Abbildung 3.2 über der Multischrittfolge $H_3 = \{t_1 : \alpha_1\}\ \{t_1 : \alpha_2\}\ \{t_2 : \alpha_3\}$ mit $\alpha_1 = (x \to 0, y \to 2, F \to A(1))$, $\alpha_2 = (x \to 2, y \to 1, F \to A(1))$ und $\alpha_3 = (z \to 1)$ konstruiert. Beim Eintreten des Ereignisses $(t_2 : \alpha_3, 3)$ wird die Marke $A(1)$ von der Ziel-Stelle Z entfernt und die Marke $A(2)$ erzeugt, was durch die Vorbedingung $(Z, 2)$ und die Nachbedingung $(Z, 3)$ protokolliert ist. Die Prädikat-Stelle A ist nicht mit der Transition t_2 im Ziel/Transitions-Netz verbunden, d.h. t_2 kann die Markierung von A nicht

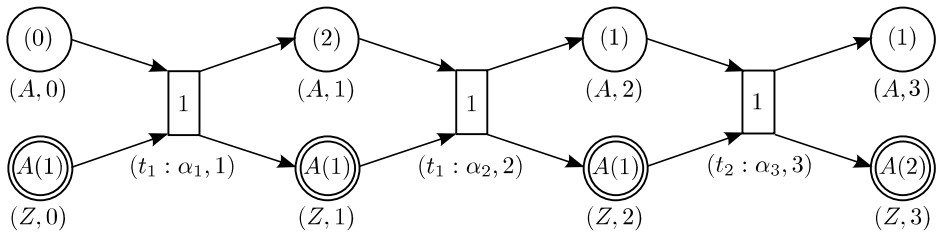


Abbildung 4.5: Ein Protokollnetz auf dem Ziel/Transitions-Netz aus Abbildung 3.2.

verändern. Dennoch greift der Wächter von t_2 über ein variables Prädikat auf die Markierung von A zu. Das Schalten von t_2 ist somit kausal von der Markierung auf A abhängig, was durch die Vorbedingung $(A, 2)$ und die Nachbedingung $(A, 3)$ protokolliert ist. Um diesen Fall zu berücksichtigen, wurden zwei entsprechende Fälle für die Kantenkonstruktion in (2.3.) in Definition 4.2 berücksichtigt. Wenn eine kausale Abhängigkeit zwischen einem Ereignis und einer Bedingung in einem Protokollnetz existiert, die entsprechende Transition aber nicht mit der jeweiligen Stelle im Ziel/Transitions-Netz verbunden ist, dann muss diese Verbindung auf die Verwendung von variablen Prädikaten im Wächter der Transition zurückzuführen sein. Der Wächter beschreibt dabei nur einen "lesenden" Zugriff auf die Markierung der Stelle und verändert diese nicht, d.h. die Markierung der Bedingung im Vorbereich ist mit der Markierung der Bedingung im Nachbereich identisch, sofern keine anderen Transitioneninstanzen im selben Multischritt die Markierung der Stelle verändern.

Theorem 4.2
Sei $PN = (PB, PE, PF, PZ, PP)$ ein Protokollnetz eines Ziel/Transitions-Netzes ZT und PN_n das im Konstruktionsschritt $n \in \mathbb{N}$ entstandene Konstrukt von PN.

(1.) Sei $(t : \alpha, l) \in PE_n$ ein Ereignis und $(p, i) \in PB_n$ eine Bedingung mit $(p, i) \in PF_n(t : \alpha, l)$ oder $(p, i) \in (t : \alpha, l)PF_n$. Wenn $(p, t), (t, p) \notin F_N$, dann ist $A_P(p) \in A_T(t) : \alpha$.

(2.) Seien $(p, i), (p, j) \in PB_n$ zwei Bedingungen mit $(p, i)PF_n = PF_n(p, j)$. Wenn $(p, t), (t, p) \notin F_N$ für alle Ereignisse $(t : \alpha, l) \in (p, i)PF_n$, dann ist $PZ_n(p, i) = PZ_n(p, j)$.

Beweis: Zu (1.): Weil $p \notin F_N t \cup tF_N$, sind nach (2.3.) in Definition 4.2 nur zwei (von vier) Fällen für die Konstruktion einer Kante $((t : \alpha, l), (p, i)) \in PF_n$ oder $((p, i), (t : \alpha, l)) \in PF_n$ möglich. In beiden Fällen ist $A_P(p) \in A_T(t) : \alpha$.

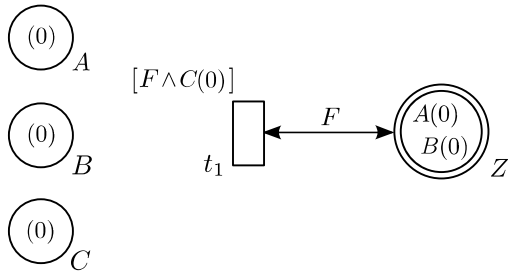


Abbildung 4.6: Ein Ziel/Transitions-Netz mit variablen kausalen Abhängigkeiten.

Zu (2.): Es ist zu zeigen, dass $M_i(p) = M_j(p)$. Alle Ereignisse $(t : \alpha, l) \in (p, i)PF_n$ im Nachbereich von (p, i) haben nach (7.) in Lemma 4.1 den selben Index l und stammen somit aus dem selben Multischritt G_l. Nach Voraussetzung ist $p \notin F_N t \cup tF_N$ und das Schalten einer Transitioneninstanz $t : \alpha$ aus G_l kann die Folgemarkierung von p nach Definition 3.2 nicht beeinflussen, da $A_F(p, t) = A_F(t, p) = 0$. Somit hat p in M_i die selbe Markierung wie nach dem Schalten von G_l in M_l, d.h. $M_i(p) = M_l(p)$. Weiterhin ist nach Voraussetzung jedes Ereignis $(t : \alpha, l)$ im Nachbereich von (p, i) gleichzeitig Ereignis im Vorbereich von (p, j). Nach (5.) in Lemma 4.1 ist dann $l = j$ und somit $M_i(p) = M_j(p)$. □

Kausale Abhängigkeiten zwischen einem Ereignis und einer Bedingung in einem Protokollnetz können von der Variablenbelegung der entsprechenden Transitioneninstanz abhängig sein. Dies ist der Fall, wenn der Zusammenhang ausschließlich auf die Verwendung von variablen Prädikaten im Wächter der Transition zurückzuführen ist, die variablen Prädikate aber nicht struktureller Bestandteil des Wächters sind, sondern erst durch die Belegung von Formelvariablen entstehen. Die Verwendung von Formalvariablen kann somit variable Kausalzusammehänge implizieren. Ein einfaches Beispiel zeigt Abbildung 4.6. Je nach Belegung der Formelvariable F entsteht eine kausale Abhängigkeit zwischen t_1 und der Prädikat-Stelle A oder zwischen t_1 und der Prädikat-Stelle B. Die Abhängigkeiten zwischen t_1 und der Prädikat-Stelle C sind strukturell vom Wächter vorgegeben, während die Abhängigkeiten zwischen t_1 und der Ziel-Stelle Z durch die Struktur des Netzes bedingt sind.

4.5 Schnitte

Während kausal abhängige Elemente durch Ketten und Linien identifiziert werden, beschreiben Antiketten und Schnitte kausal unabhängige Elemente in einem Protokollnetz PN bzw. einem Konstrukt PN_n.

Definition 4.5 (Antikette, Schnitt)
Sei $PN = (PB, PE, PF, PZ, PP)$ ein Protokollnetz und PN_n das im Konstruktionsschritt $n \in \mathbb{N}$ entstandene Konstrukt von PN. Seien $(x, i), (y, j) \in PB_n \cup PE_n$ zwei beliebige Elemente von PN_n.

(1.) Die Relation $\mathrm{co}_n \subseteq (PB_n \cup PE_n) \times (PB_n \cup PE_n)$ *ist gegeben durch:* $(x, i)\mathrm{co}_n(y, j) \Leftrightarrow \neg(x, i)\mathrm{li}_n(y, j) \vee (x, i) = (y, j)$.

(2.) Eine Menge $C_n \subseteq PB_n \cup PE_n$ heißt Antikette *genau dann, wenn* $\forall (x, i), (y, j) \in C_n : (x, i)\mathrm{co}_n(y, j)$.

(3.) Eine Menge $C_n \subseteq PB_n \cup PE_n$ heißt Schnitt *genau dann, wenn C_n eine Antikette ist und* $\forall (x, i) \in (PB_n \cup PE_n) \setminus C_n : \exists (y, j) \in C_n : \neg(x, i)\mathrm{co}_n(y, j)$.

Die Relation co_n identifiziert kausal unabhängige Elemente in einem Protokollnetz. Eine Antikette bzw. ein Schnitt beschreibt eine Menge bzw. eine maximale Menge von Bedingungen und Ereignissen, die während eines Ablaufs in keinem Ursache-Wirkungs-Zusammenhang stehen. Ein Beispiel für einen Schnitt im Protokollnetz von Abbildung 4.3 ist $C_1 = \{(t_1 : \alpha_1, 1), (t_2 : \alpha_3, 1)\}$. Beide Ereignisse sind kausal unabhängig voneinander, da die entsprechenden Transitioneninstanzen nebenläufig eintreten. Auch die Menge $C_2 = \{(A, 1), (B, 1), (C, 1), (D, 1), (E, 0), (Z, 1)\}$ bildet einen Schnitt. Keine Bedingung aus C_2 ist Voraussetzung oder Folge einer anderen Bedingung aus C_2. Darüber hinaus sind auch Schnitte möglich, in denen Bedingungen und Ereignisse gemeinsam vorkommen.

Besondere Bedeutung für die Analyse von Protokollnetzen haben Schnitte, die ausschließlich aus Bedingungen bestehen. Solche Schnitte werden *Scheiben* genannt und stellen Momentaufnahmen eines Ablaufs dar. In Kausalnetzen sind alle Stellen eines Petri-Netzes durch entsprechende Bedingungen in einer Scheibe vertreten, so dass jede Scheibe vollständig einen Zustand des Modells während des Ablaufs wiedergibt. In Protokollnetzen können durch die möglichen Verzweigungen von Bedingungen auch Scheiben entstehen, die nur unvollständig eine Markierung eines Netzes beschrei-

ben. Aus diesem Grund wird der Begriff der *regulären Scheibe* in Protokollnetzen verwendet.

Definition 4.6 (Reguläre Scheibe)
Sei $PN = (PB, PE, PF, PZ, PP)$ ein Protokollnetz.

(1.) Die Relation $R \subseteq PE \times PE$ ist gegeben durch: $(t : \alpha, k)R(t' : \alpha', k) \Leftrightarrow PF(t : \alpha, k) \cap PF(t' : \alpha', k) = \emptyset$. Die reflexiv-transitive Hülle R^* ist eine Äquivalenzrelation auf PE.

(2.) Für ein $(t : \alpha, k) \in PE$ bezeichnet $[(t : \alpha, k)] = \{(t' : \alpha', k) \mid (t : \alpha, k)R^*(t' : \alpha', k)\}$ die R^*-Äquivalenzklasse von $(t : \alpha, k)$. Die Menge aller R^*-Äquivalenzklassen von PE wird mit $[PE]$ notiert.

(3.) Ein Schnitt D von PN heißt Scheibe *genau dann, wenn $D \subseteq PB$. Eine Scheibe D heißt* regulär *genau dann, wenn $PF^*[(t : \alpha, k)] \cap D = \emptyset \vee [(t : \alpha, k)]PF^* \cap D = \emptyset$ für alle Äquivalenzklassen $[(t : \alpha, k)] \in [PE]$.*

Eine R^*-Äquivalenzklasse beschreibt eine Menge von Ereignissen, die durch den selben Multischritt entstanden sind und sich gemeinsame Vorbedingungen teilen. Im Protokollnetz von Abbildung 4.3 können $\{(t_1 : \alpha_1, 1), (t_2 : \alpha_3, 1)\}, \{(t_3 : \alpha_5, 2)\}$ und $\{(t_4 : \alpha_6, 3)\}$ als R^*-Äquivalenzklassen identifiziert werden. Eine reguläre Scheibe ist ein Schnitt aus Bedingungen, der nicht gleichzeitig Elemente aus dem Vorbereich und dem Nachbereich einer R^*-Äquivalenzklasse enthält. Auf diese Weise repräsentieren Reguläre Scheiben die Systemzustände während des Ablaufes, wobei die Markierungen der einzelnen Stellen von der Abbildung PZ protokolliert werden. MaxPN_n und MinPN_n bilden immer reguläre Scheiben: MinPN_n repräsentiert die Anfangsmarkierung des Netzes, während MaxPN_n den Zustand des Systems nach der Ausführung des letzten Multischrittes beschreibt.

Theorem 4.3
Sei $PN = (PB, PE, PF, PZ, PP)$ ein Protokollnetz eines Ziel/Transitions-Netzes ZT. Wenn $D \subseteq PB$ eine reguläre Scheibe von PN ist, dann ist $PZ(D) = \bigcup_{(p,i)\in D} PZ(p, i) = M$ eine erreichbare Markierung von ZT.

Dieser Zusammenhang lässt sich durch Induktion über die Anzahl der Multischritte beweisen, auf deren Grundlage das Protkollnetz konstruiert wurde. Die regulären Scheiben entstehen in jedem Multischritt, indem die Vorbereiche der R*-Äquivalenzklassen durch die Nachbereiche ersetzt werden. Für einen ausführlichen Beweis siehe [Cze97, S. 38 - 46].

In Abbildung 4.7 werden alle regulären Scheiben dargestellt, die im Protokollnetz von Abbildung 4.3 identifiziert werden können. Sie geben alle Markierungen des Ziel/Transitions-Netzes während des Ablaufs vollständig wieder:

- $D_0 = \{(A,0),(B,0),(C,0),(D,0),(E,0),(Z,0)\}$
  mit $PZ(D_0) = \bigcup_{(p,i)\in D_0} PZ(p,i) = M_0$,
- $D_1 = \{(A,1),(B,1),(C,1),(D,1),(E,0),(Z,1)\}$
  mit $PZ(D_1) = \bigcup_{(p,i)\in D_1} PZ(p,i) = M_3$,
- $D_2 = \{(A,1),(B,2),(C,1),(D,2),(E,2),(Z,2)\}$
  mit $PZ(D_2) = \bigcup_{(p,i)\in D_2} PZ(p,i) = M_4$,
- $D_3 = \{(A,3),(B,2),(C,3),(D,2),(E,3),(Z,3)\}$
  mit $PZ(D_3) = \bigcup_{(p,i)\in D_3} PZ(p,i) = M_5$.

Die Scheiben $D_4 = \{(A,0), (B,0), (C,1), (D,1), (E,0)\}$ und $D_5 = \{(A,1), (B,1), (C,0), (D,0), (E,0)\}$ sind zum Vergleich nicht regulär und repräsentieren keine Zustände des Systems. In einem Protokollnetz, das über der Multischrittfolge $H = \{t_1 : \alpha_1, t_2 : \alpha_3\}\ \{t_3 : \alpha_5\}\ \{t_4 : \alpha_6\}\ \{t_1 : \alpha_7, t_2 : \alpha_8\}\ \{t_3 : \alpha_6\}\ \{t_4 : \alpha_5\}$ konstruiert wurde, können insgesamt zehn reguläre Scheiben identifiziert werden, die die Erreichbarkeitsmenge des Ziel/Transitions-Netzes vollständig wiedergeben.

Die Halbordnung über den Bedingungen und Ereignissen eines Protokollnetzes lässt sich auf reguläre Scheiben übertragen. Auf diese Weise wird eine kausale Struktur über den Systemzuständen, die von den regulären Scheiben repräsentiert werden, gebildet.

Definition 4.7 (Folgescheibe)
Sei $PN = (PB, PE, PF, PZ, PP)$ ein Protokollnetz eines Ziel/Transitions-Netzes ZT und $D, D' \subseteq PB$ zwei reguläre Scheiben von PN. D' heißt genau dann Folgescheibe *von D, notiert mit $D \preceq D'$, wenn $\forall (p,i) \in D : \forall (p',j) \in D' : p = p' \Rightarrow i \leq j$.*

Theorem 4.4
Sei $PN = (PB, PE, PF, PZ, PP)$ ein Protokollnetz eines Ziel/Transitions-Netzes ZT, $D, D' \subseteq PB$ zwei reguläre Scheiben von PN und $M = PZ(D)$ und $M' = PZ(D')$ zwei Markierungen von ZT. Wenn $D \preceq D'$, dann ist $M \xrightarrow{*} M'$.

Beweis: Das Protokollnetz sei über der Multischrittfolge $H = G_1 ... G_k$ entstanden. Da es sich bei D und D' um reguläre Scheiben handelt, sind nach Theorem 4.3 $M = PZ(D)$ und $M' = PZ(D')$ in GT erreichbar. Nun sei $u = \max\{i \mid (p,i) \in D\}$

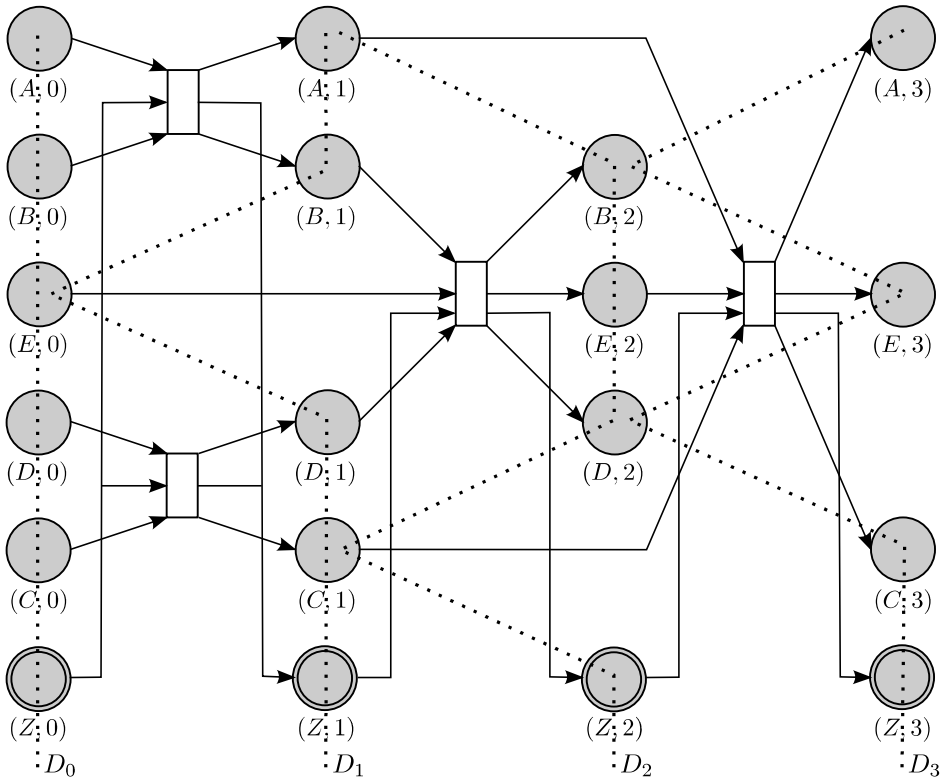


Abbildung 4.7: Reguläre Scheiben im Protokollnetz aus Abbildung 4.3.

der maximale Multischrittindex in D und $v = \min\{i \mid (p, i) \in D'\}$ der minimale Multischrittindex in D'. Nach Voraussetzung ist $D \preceq D'$ und somit $u \leq v$. Für alle Ereignisse $(t : \alpha, l)$ mit $l \leq u$ ist nach (8.) in Lemma 4.1 auch $l \leq v$, d.h. alle Ereignisse, die vor D vorkommen, kommen auch vor D' vor. Die Transitioneninstanzen, die das Anfangsstück von H bilden und M_0 in M überführen, sind somit auch der Beginn einer Multischrittfolge, die M_0 in M' überführt. Weil M und M' von M_0 erreichbar sind, muss daher auch eine Multischrittfolge existieren, die M in M' überführt. Dabei ist G_u der letzte Multischritt, der die Markierung M erzeugt, und G_v der letzte Multischritt, der die Markierung M' erzeugt. □

Für die regulären Scheiben aus Abbildung 4.7 ergibt sich $D_0 \preceq D_1 \preceq D_2 \preceq D_3$. Es gilt $M_0 \xrightarrow{*} M_3 \xrightarrow{*} M_4 \xrightarrow{*} M_5$ mit $M_0 = PZ(D_0)$, $M_3 = PZ(D_1)$, $M_4 = PZ(D_2)$ und $M_5 = PZ(D_3)$.

4.6 Zielorientierte Abläufe

Für die Überprüfung von zielorientiertem Verhalten in den Abläufen eines Ziel/-Transitions-Netzes müssen die Formeln auf den Ziel-Stellen eines Netzes mit den Zuständen des Netzes in einen Zusammenhang gebracht werden. Die Verwendung einer kausalen Ablaufdarstellung hat gegenüber einer sequentiellen Ablaufdarstellung dabei den Vorteil, dass die Untersuchung ausschließlich auf kausalen Zusammenhängen basiert und keine *künstliche* Reihenfolge von nebenläufigen Ereignissen angenommen werden muss.

Bei der Betrachtung von Agenten sind grundsätzlich zwei unterschiedliche Arten von Zielen zu berücksichtigen [Woo02]. In *Achievement tasks* besteht die Aufgabe eines Agenten darin, mit seinen Aktionen bestimmte Systemzustände zu erreichen. In *Maintenance tasks* muss ein Agent stattdessen versuchen, bestimmte Systemzustände zu bewahren, d.h. er muss unerwünschte Systemzustände vermeiden. Beide Arten von Zielen sind unterschiedlich zu behandeln und werden im Folgenden von den Begriffen *Ziel* und *Antiziel* repräsentiert. Ziele beschreiben Systemzustände, die eintreten sollen, während Antiziele Systemzustände beschreiben, die nicht eintreten dürfen.

Für die Unterscheidung von Zielen und Antizielen in Ziel/Transitions-Netzen werden aus der Menge der Ziel-Stellen zwei disjunkte Teilmengen gebildet, so dass jede Ziel-Stelle immer nur eine Art von Zielen enthalten kann. Die Vereinigung beider Mengen muss dabei nicht notwendigerweise alle Ziel-Stellen eines Ziel/Transitions-Netzes umfassen.

Definition 4.8 (Ziel, Antiziel)
Sei $PN = (PB, PE, PF, PZ, PP)$ ein Protokollnetz für ein Ziel/Transitions-Netz ZT. Für ZT sei die Menge $\widehat{P}_+ \subseteq \widehat{P}_N$ und die Menge $\widehat{P}_- \subseteq \widehat{P}_N$ mit $\widehat{P}_+ \cap \widehat{P}_- = \emptyset$ gegeben.

(1.) Eine erweiterte Formel F heißt genau dann Ziel *von PN, wenn eine Bedingung $(p, i) \in PB$ existiert mit $F \in PZ(p, i)$ und $p \in \widehat{P}_+$.*

(2.) Eine erweiterte Formel F heißt genau dann Antiziel *von PN, wenn eine Bedingung $(p, i) \in PB$ existiert mit $F \in PZ(p, i)$ und $p \in \widehat{P}_-$.*

Ziele und Antiziele haben ihren Ursprung in erweiterten Formeln, die die entsprechenden Ziel-Stellen während eines Ablaufs markieren. Ein Ziel ist in einem Zustand

genau dann erfüllt, wenn die entsprechende Formel erfüllt ist. Ein Antiziel ist in einem Zustand genau dann erfüllt, wenn die entsprechende Formel nicht erfüllt ist. Wie bereits im vorherigen Abschnitt dargestellt, werden die Zustände eines Ziel/-Transitions-Netzes in Protokollnetzen von regulären Scheiben repräsentiert. Mit den Begriffen Ziel, Antiziel und der Erfüllung von Zielen ist es schließlich möglich, zielorientiertes Verhalten in den Abläufen von Ziel/Transitions-Netzen formal zu erfassen.

Definition 4.9 (Zielorientierter Ablauf)
Sei $PN = (PB, PE, PF, PZ, PP)$ ein Protokollnetz für ein Ziel/Transitions-Netz ZT und $D \subseteq PB$ eine reguläre Scheibe von PN.

(1.) Ein Ziel F von PN heißt erfüllt *in D genau dann, wenn $\varepsilon(F) = \top$ bei $M = PZ(D)$.*

(2.) Ein Antiziel F von PN heißt erfüllt *in D genau dann, wenn $\varepsilon(F) = \bot$ bei $M = PZ(D)$.*

(3.) PN heißt zielorientiert *für ein Ziel F von PN in D genau dann, wenn eine reguläre Scheibe $D' \subseteq PB$ mit $D \preceq D'$ existiert, so dass F in D' erfüllt ist oder F in D' kein Ziel ist.*

(4.) PN heißt zielorientiert *für ein Antiziel F von PN in D genau dann, wenn für alle regulären Scheiben $D' \subseteq PB$ mit $D \preceq D'$ gilt, dass F in D' erfüllt ist oder F in D' kein Antiziel ist.*

(5.) PN heißt zielorientiert *genau dann, wenn PN für alle Ziele und Antiziele von PN zielorientiert ist.*

In einem zielorientierten Ablauf wird jedes Ziel, das in einem Zustand erzeugt wurde, in einem Folgezustand erfüllt oder wieder verworfen. Gleichzeitig ist jedes Antiziel nach seiner Erzeugung in jedem Folgezustand erfüllt, solange es nicht wieder verworfen wurde. Das Verwerfen von Zielen und Antizielen stellt dabei keinen Widerspruch zu zielorientiertem Verhalten dar, weil es viele Gründe geben kann, warum ein Agent seine Ziele und Antiziele während des Ablaufs ändert (vgl. Abschnitt 2.2.3).

Abbildung 4.8 zeigt ein Ziel/Transitions-Netz und einen Ablauf in Form eines Protokollnetzes. Das Protokollnetz wurde über der Multischrittfolge $H = \{t_1\}\ \{t_2\}\ \{t_1\}\ \{t_2\}$ konstruiert. Auf die Darstellung der Variablenbelegungen wurde dabei verzichtet, weil das Netz keine Alternativen in den einzelnen Multischritten zulässt. Für die Analyse des Ablaufs werden Formeln auf Z_+ als Ziele und Formeln auf Z_- als

Antiziele interpretiert, d.h. $\widehat{P}_+ = \{Z_+\}$ und $\widehat{P}_- = \{Z_-\}$. Die Zustände des Ablaufs werden von fünf regulären Scheiben des Protokollnetzes repräsentiert:

- $D_0 = \{(A,0),(Z_+,0),(Z_-,0)\}$
  mit $PZ(D_0) = \bigcup_{(p,i)\in D_0} PZ(p,i) = M_0$,
- $D_1 = \{(A,1),(Z_+,0),(Z_-,0)\}$
  mit $PZ(D_1) = \bigcup_{(p,i)\in D_1} PZ(p,i) = M_1$,
- $D_2 = \{(A,2),(Z_+,2),(Z_-,2)\}$
  mit $PZ(D_2) = \bigcup_{(p,i)\in D_2} PZ(p,i) = M_2$,
- $D_3 = \{(A,3),(Z_+,2),(Z_-,2)\}$
  mit $PZ(D_3) = \bigcup_{(p,i)\in D_3} PZ(p,i) = M_3$,
- $D_4 = \{(A,4),(Z_+,4),(Z_-,4)\}$
  mit $PZ(D_4) = \bigcup_{(p,i)\in D_4} PZ(p,i) = M_4$,

Für die Ziele des Ablaufs gilt: Das Protokollnetz ist zielorientiert für das Ziel $A(1)$ in den Scheiben D_0 und D_1, weil es in der Scheibe D_1 erfüllt ist. Dies gilt auch für $A(0)$ in D_2, das bereits in der selben Scheibe erfüllt ist. Das Ziel $A(0)$ in D_3 wird in der Scheibe D_4 verworfen und durch das Ziel $A(1)$ ersetzt. Das Protokollnetz ist somit auch für $A(0)$ in D_3 zielorientiert. Das Ziel $A(1)$ in der letzten Scheibe D_4 verhindert jedoch die Zielorientierung des Protokollnetzes, weil keine Folgescheibe existiert, die das Ziel erfüllt.

Für die Antiziele des Ablaufs gilt: Der Ablauf ist nicht zielorientiert für das Antiziel $A(1)$ in den Scheiben D_0 und D_1, weil es in der Scheibe D_1 nicht erfüllt ist. Dies gilt auch für $A(0)$ in D_2, das bereits in der selben Scheibe nicht erfüllt ist. Für das Antiziel $A(0)$ in der Scheibe D_3 ist das Protokollnetz allerdings zielorientiert, weil es in D_3 erfüllt ist und das Antiziel in der Folgescheibe wieder verworfen wird. Die Zielorientierung kann auch für das Antiziel $A(1)$ in der letzten Scheibe D_4 festgestellt werden, weil es in D_4 erfüllt ist und keine Folgescheibe existiert, in der es nicht erfüllt sein kann.

Für die Überprüfung von zielorientiertem Verhalten in den Abläufen von Ziel/Transitions-Netzen kann die Anzahl der Multischritte, über denen ein Protokollnetz konstruiert wurde, entscheidend sein. Ein unerfülltes Ziel, das die Zielorientierung eines Ablaufs verhindert, ist möglicherweise in einem Folgezustand erfüllt, der nicht im Ablauf enthalten ist. Genauso kann ein erfülltes Antiziel in einem Folgezustand, der nicht im Ablauf enthalten ist, unerfüllt sein und somit die Zielorientierung eines

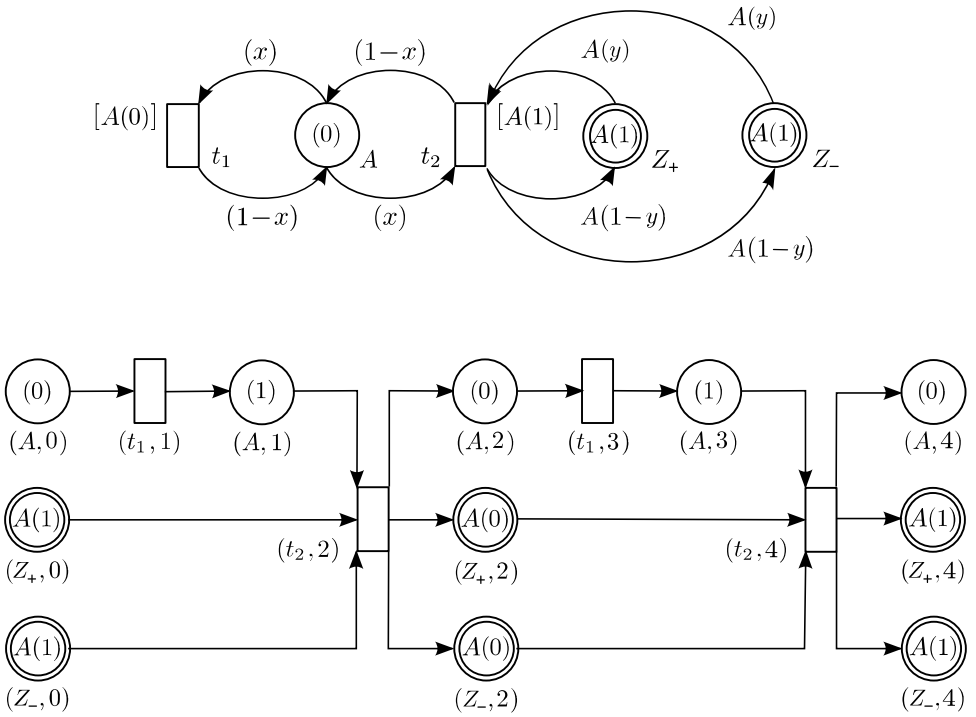


Abbildung 4.8: Ein Ziel/Transitions-Netz (oben) und ein Ablauf in Form eines Protokollnetzes (unten).

Ablaufs verhindern. Für die Analyse von Systemabläufen ist also der Ausschnitt des Modellverhaltens, der von einem Protokollnetz erfasst wurde, zu berücksichtigen. Ein Beispiel zeigt das Protokollnetz in Abbildung 4.8. Der Ablauf ist für das Antiziel $A(1)$ in D_4 zielorientiert, für das Ziel $A(1)$ in D_4 allerdings nicht. Wenn die Multischrittfolge des Protokollnetzes um einen zusätzlichen Multischritt $\{t_1\}$ erweitert wird, kehrt sich diese Situation in das Gegenteil um. Es entsteht eine neue reguläre Scheibe $D_5 = \{(A,5),(Z_+,4),(Z_-,4)\}$, die das Ziel $A(1)$ erfüllt und das Antiziel $A(1)$ nicht erfüllt.

Für das Ziel/Transitions-Netz aus Abbildung 3.5 sind alle Abläufe zielorientiert, wenn Formeln auf Z als Ziele interpretiert werden und keine Antiziele zu berücksichtigen sind, d.h. wenn $\widehat{P}_+ = \{Z\}$ und $\widehat{P}_- = \emptyset$.

4.7 Anwendungsbeispiel

In diesem Abschnitt wird ein einfacher Agent mit dynamischen Zielen als Ziel/-Transitions-Netz modelliert und ein Ablauf des Netzes auf seine Zielorientierung untersucht. Als Beispiel dient eine vereinfachte Variante des klassischen MarsWorld-Szenarios [Ste90a], das in in Abbildung 4.9 dargestellt ist. Die Aufgabe des Agenten besteht darin, einen entfernten Planten zu erforschen. Dazu wird der Agent nach der Landung ausgesetzt, um die Umgebung zu erkunden, geeignete Gesteinsproben einzusammeln und zur Landebasis zu bringen. Das Gelände kann in verschiedene Sektoren aufgeteilt werden, wobei sich die Landebasis und der Ausgangspunkt des Agenten im Sektor 0 befinden. Eine Karte über das Vorkommen von geeigneten Proben ist nicht vorhanden. Es ist allerdings bekannt, dass sich geeignete Steine lokal häufen und der Agent kann ermitteln, ob sich überhaupt noch Proben in Reichweite befinden oder nicht.

Abbildung 4.10 zeigt ein Ziel/Transitions-Netz als Modell für einen MarsWorld-Agenten. Die Umgebung des Agenten wird durch die Prädikat-Stelle *Sample* repräsentiert, auf der die Positionen der Proben als Marken hinterlegt sind. Das aktuelle Ziel des Agenten wird durch eine Formel auf der Ziel-Stelle $Goal_A$ festgehalten. Zu Beginn besteht das Ziel darin, alle Proben im Sektor 0 (falls vorhanden) einzusammeln, was durch die Formel $\neg Sample(0)$ ausgedrückt wird. Die Bewegung des Agenten wird über die Transition *move* realisiert. Sie entfernt die aktuelle Positionsmarke von der Prädikat-Stelle *Position* und legt die neue Position des Agenten anhand des aktuellen Ziels fest. Gleichzeitig wird eine entsprechende Marke auf

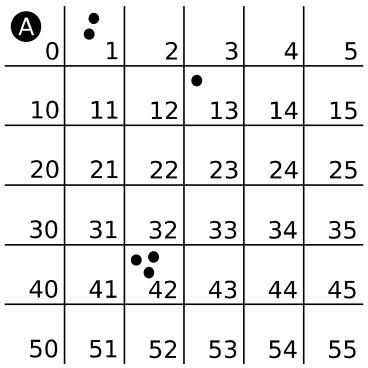


Abbildung 4.9: Schematische Darstellung des MarsWorld-Szenarios.

der Prädikat-Stelle *Sensor* abgelegt, um anzudeuten, dass der Agent den Sektor anschließend auf das Vorhandensein einer geeigneten Probe untersuchen kann. Findet der Agent keine Probe vor, dann ist das aktuelle Ziel erfüllt und der Vorgang kann durch Schalten der Transition *abort* abgebrochen werden. Das Ziel auf der Ziel-Stelle $Goal_A$ wird entfernt und durch ein neues Ziel ausgetauscht. Dazu wird die Funktion $next : \mathbb{N} \to \mathbb{N}$ verwendet, die (zufällig oder systematisch) den nächsten Sektor ermittelt. Findet der Agent stattdessen eine Probe im aktuellen Sektor vor, wird sie mit dem Schalten der Transition *collect* aufgenommen. Dabei wird die entsprechende Marke von der Prädikat-Stelle *Sample* entfernt und auf der Stelle *Load* abgelegt. Nach dem Aufnehmen der Probe kehrt der Agent durch Schalten der Transition $return_1$ oder $return_2$ zur Landebasis im Sektor 0 zurück, um die Probe abzuliefern. Befinden sich noch weitere Proben im letzten Sektor ($return_1$), dann behält der Agent das aktuelle Ziel unverändert bei, um den Sektor nach seiner Rückkehr zur Landebasis wieder aufzusuchen. Sind die Proben vollständig abgetragen ($return_2$), wird das aktuelle Ziel von $Goal_A$ entfernt und ebenfalls mit der Funktion $next : \mathbb{N} \to \mathbb{N}$ ein neues Ziel generiert. Die Fallunterscheidungen zwischen $return_1$ und $return_2$ sowie zwischen *collect* und *abort* werden durch entsprechende Wächter realisiert, die das variable Prädikat *Sample* verwenden. Ein weiterer Wächter ist der Transition *move* zugeordnet, der verhindert, dass der Agent das Gelände erkundet, während sich Marken auf der Stelle *Sensor* oder *Load* befinden. Der Wächter realisiert somit einen klassischen Nulltest. Gleichzeitig wird mit Hilfe der Formelvariablen F auf die Marke der Ziel-Stelle $Goal_U$ zugegriffen, die das Endziel des Agenten repräsentiert. Nur wenn das Endziel noch nicht erreicht ist und sich Proben in Reichweite befinden, kann die Transition *move* schalten. Ansonsten verbleibt der Agent nach seiner letzten Rückkehr in der Landebasis und das Netz erreicht eine tote Markierung. Das Ziel/Transitions-Netz ist somit beschränkt und nicht lebendig. Aufgrund der Wächter existieren darüber hinaus keine Konflikte und keine Nebenläufigkeiten im Netz.

Die Abbildungen 4.11 und 4.12 zeigen einen Ablauf eines MarsWorld-Agenten in Form eines Protokollnetzes, das über der Multischrittfolge $H = \{move\}\ \{abort\}\ \{move\}\ \{collect\}\ \{return_1\}\ \{move\}\ \{collect\}\ \{return_2\}$ konstruiert wurde. Der Ablauf beschreibt einen Agenten, der im ersten Sektor erfolglos nach einer Probe sucht und anschließend im nächsten Sektor zwei Proben nacheinander zur Landebasis befördern kann. Die Zustände während des Ablaufs werden von neun regulären Scheiben im Protokollnetz repräsentiert:

- $D_0 = \{(\text{Goal}_U, 0), (\text{Goal}_A, 0), (\text{Position}, 0), (\text{Sensor}, 0), (\text{Load}, 0), (\text{Sample}, 0)\}$,

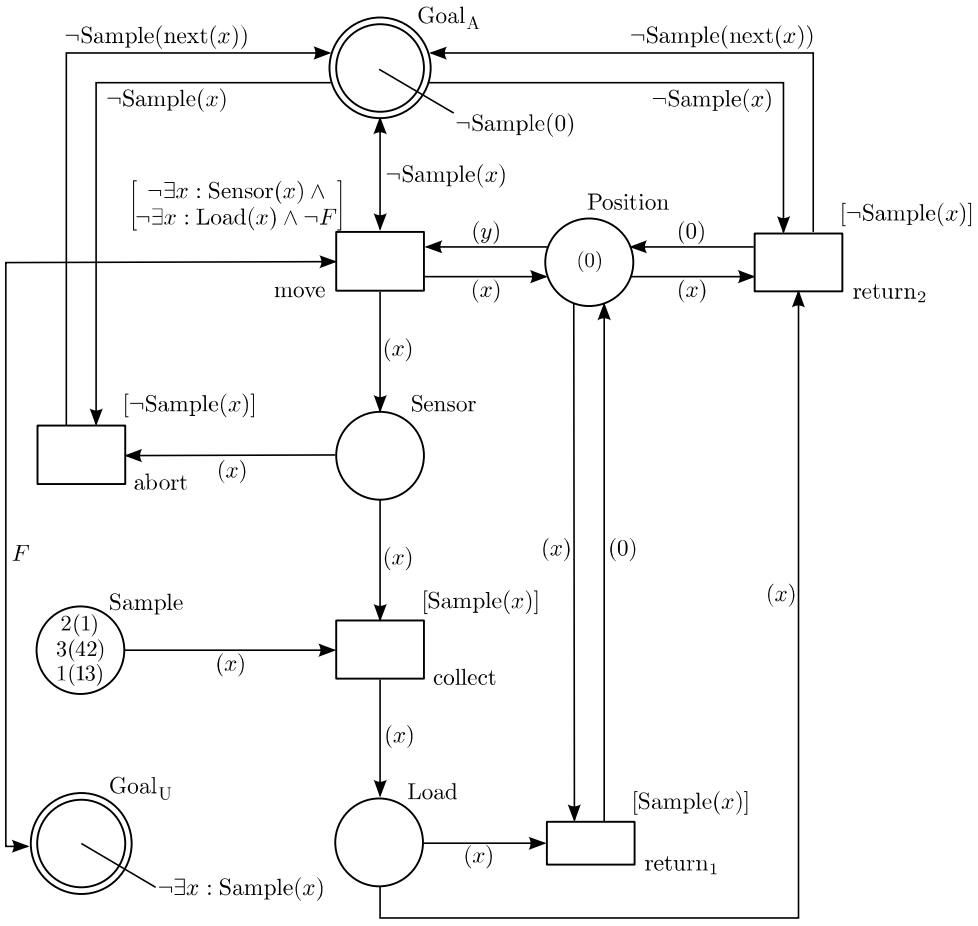


Abbildung 4.10: Ein MarsWorld-Agent als Ziel/Transitions-Netz.

- $D_1 = \{(\text{Goal}_U, 1), (\text{Goal}_A, 1), (\text{Position}, 1), (\text{Sensor}, 1), (\text{Load}, 1), (\text{Sample}, 1)\}$,
- $D_2 = \{(\text{Goal}_U, 1), (\text{Goal}_A, 2), (\text{Position}, 1), (\text{Sensor}, 2), (\text{Load}, 1), (\text{Sample}, 2)\}$,
- $D_3 = \{(\text{Goal}_U, 3), (\text{Goal}_A, 3), (\text{Position}, 3), (\text{Sensor}, 3), (\text{Load}, 3), (\text{Sample}, 3)\}$,
- $D_4 = \{(\text{Goal}_U, 3), (\text{Goal}_A, 3), (\text{Position}, 3), (\text{Sensor}, 4), (\text{Load}, 4), (\text{Sample}, 4)\}$,
- $D_5 = \{(\text{Goal}_U, 3), (\text{Goal}_A, 3), (\text{Position}, 5), (\text{Sensor}, 4), (\text{Load}, 5), (\text{Sample}, 5)\}$,
- $D_6 = \{(\text{Goal}_U, 6), (\text{Goal}_A, 6), (\text{Position}, 6), (\text{Sensor}, 6), (\text{Load}, 6), (\text{Sample}, 6)\}$,
- $D_7 = \{(\text{Goal}_U, 6), (\text{Goal}_A, 6), (\text{Position}, 6), (\text{Sensor}, 7), (\text{Load}, 7), (\text{Sample}, 7)\}$,
- $D_8 = \{(\text{Goal}_U, 6), (\text{Goal}_A, 8), (\text{Position}, 8), (\text{Sensor}, 7), (\text{Load}, 8), (\text{Sample}, 8)\}$.

Für die Analyse des Ablaufs werden Formeln auf $Goal_A$ und $Goal_U$ als Ziele interpretiert. Antiziele werden nicht berücksichtigt, d.h. $\widehat{P}_+ = \{Goal_A, Goal_U\}$ und $\widehat{P}_- = \emptyset$. Der Ablauf ist zielorientiert für das Ziel $\neg Sample(0)$ in den regulären Scheiben D_0 und D_1, da sich bereits zu Beginn des Ablaufs keine Proben im Sektor 0 befinden. In D_2 wechselt das Ziel zu $\neg Sample(1)$, das bis in D_7 unverändert beibehalten wird. Das Ziel wird in der regulären Scheibe D_7 erfüllt, nachdem der Agent die zweite und letzte Probe im Sektor 1 eingesammelt hat. Auch das Ziel $\neg Sample(2)$ in D_8 verhindert die Zielorientierung des Ablaufs nicht, da es bereits zum selben Zeitpunkt erfüllt ist. Andernfalls wäre der protokollierte Ablauf zu kurz und hätte bis zur vollständigen Abtragung des entsprechenden Sektors erweitert werden müssen, um zielorientiertes Verhalten feststellen zu können. Das Ziel $\neg\exists x : Sample(x)$, das in allen regulären Scheiben die Ziel-Stelle $Goal_U$ markiert, wird nicht erfüllt. Nur in einem Ablauf, in dem der Agent alle Proben in Reichweite einsammeln konnte, kann das Ziel erfüllt werden. Ein Ablauf, der das Verhalten des Agenten bis zu seinem endgültigen Verbleib in der Landebasis protokolliert, ist für alle Ziele zielorientiert.

Während variable Prädikate in den Wächtern von Transitionen die Ausdrucksmöglichkeiten von Ziel/Transitions-Netzen erhöhen, scheint die Verwendung von Formel-Marken wie in diesem Beispiel nicht immer zwingend notwendig. Im Gegensatz zum BlocksWorld-Szenario hätten die Ziele des Agenten im MarsWorld-Szenario auch von einfachen Tupeln einer Prädikat-Stelle repräsentiert werden können. Mit der Darstellung von Systemabläufen in Form von Protokollnetzen wird jedoch ein zentraler Vorteil der Verwendung von Formel-Marken deutlich: Ziele und Zustände können direkt in Beziehung gesetzt werden, so dass ihre Wechselwirkung untersucht werden kann.

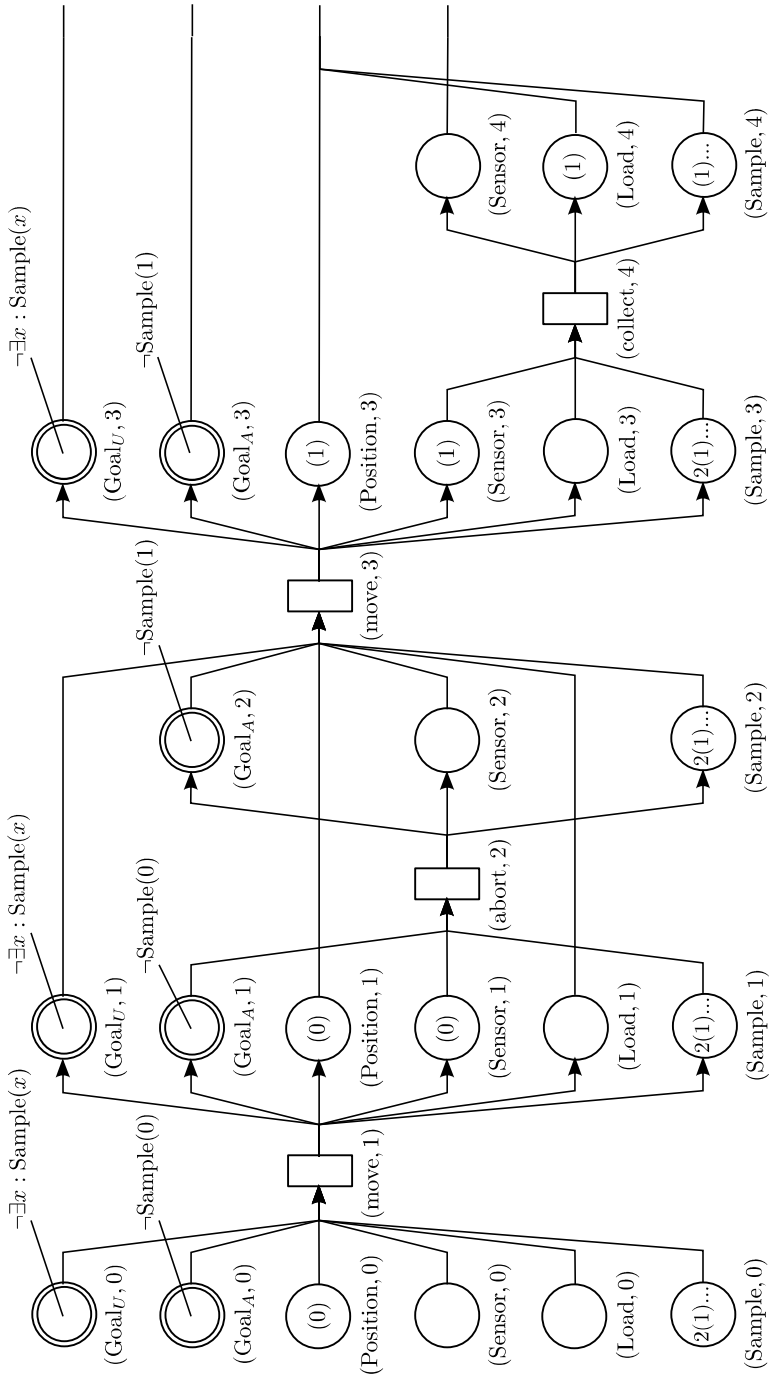


Abbildung 4.11: Ein Ablauf eines MarsWorld-Agenten.

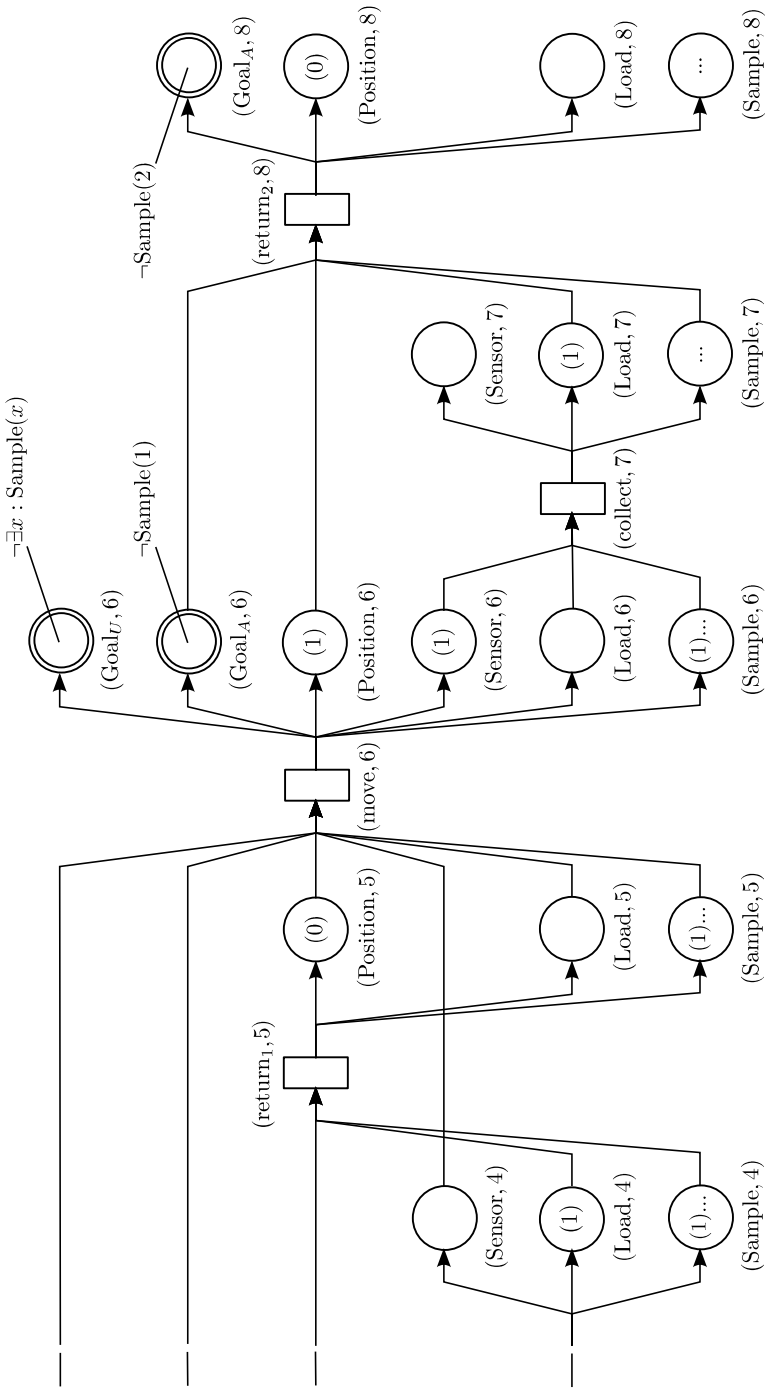


Abbildung 4.12: Ein Ablauf eines MarsWorld-Agenten (Fortsetzung).

5

Schlussbetrachtung

5.1 Zusammenfassung

In dieser Arbeit wurde ein Formalismus entwickelt, um Ziele und zielorientiertes Verhalten in Agentensystemen formal zu erfassen. Zu diesem Zweck wurden zunächst die grundlegenden Ideen und Begriffe der Agentenorientierung herausgearbeitet. Als zentrale Merkmale eines Agenten wurden Autonomie, Reaktivität, Proaktivität und Sozialität identifiziert. Insbesondere die Bedeutung von dynamischen Zielen, die zur Laufzeit eines Agenten Änderungen unterliegen können, wurde betont. Geeignete Ansätze zur Modellierung von Agentensystemen unterstützen idealerweise all diese Charakteristika. Anschließend wurden die in der Literatur verbreiteten Ansätze zur Modellierung von Agentensystemen in Bezug auf die Anforderungen Formalität, Anschaulichkeit und Agentenorientierung untersucht und verglichen. Eine Zusammenfassung zeigt Abbildung 5.1. Neben Graphtransformationen wurden Petri-Netze als viel versprechender Ausgangspunkt identifiziert. Insbesondere mit höheren Petri-Netzen ist es möglich, Agenten formal und anschaulich zu beschreiben. Die in der Literatur verbreiteten Ansätze zur Modellierung von Agentensystemen, die auf Graphtransformationen und Petri-Netze basieren, berücksichtigen jedoch nicht die Darstellung von dynamischen Zielen. Aus diesem Grund wurde ein Ansatz vorgeschlagen, um diese Lücke zu schließen.

Aufgrund der Vielzahl von Ansätzen, die eine Modellierung von Agentensystemen mit Petri-Netzen vorschlagen, wurde in dieser Arbeit ebenfalls ein Petri-Netz-orientierter Ansatz verfolgt. Auf diese Weise lassen sich die bisherigen Ergebnisse mit den

Ausgangspunkt	Formalität	Anschaulichkeit	Agentenorientierung
UML		+	
Agent UML		+	+
Automaten	+	+	
Zelluläre Automaten	+	+	
Algebr. Spezifikation	+		
Modal-/Temporallogik	+		+
Graphtransformationen	+	+	+
Petri-Netze	+	+	+

Abbildung 5.1: Ansätze zur Modellierung von Agentensystemen im Vergleich.

Ergebnissen dieser Arbeit vereinen. Zu diesem Zweck wurde der Formalismus der Ziel/Transitions-Netze entwickelt. Die Grundidee bestand darin, den Zusammenhang zwischen Formeln mit variablen Prädikaten und Markierungen eines Netzes auszunutzen und in der Modellebene abzubilden. So lassen sich die Zustände eines Modells in den Zuständen des Modells beschreiben, was die Darstellung von dynamischen Zielen auf der Modellebene ermöglicht. Anschließend wurden die wichtigsten Grundbegriffe der Petri-Netz-Theorie auf Ziel/Transitions-Netze übertragen, Aspekte zur Mächtigkeit diskutiert und einige Modifikationen vorgestellt. Anhand eines klassischen Anwendungsbeispiels der Agenten-Literatur wurde die Modellierung von dynamischen Zielen eines Agenten mit Ziel/Transitions-Netzen demonstriert.

Für die Darstellung und Analyse von Systemabläufen in Ziel/Transitions-Netzen wurde eine kausale Ablaufdarstellung entwickelt. Zu diesem Zweck wurde der Begriff der Protokollnetze, der für S/T-Systeme vorgeschlagen wurde, auf Ziel/Transitions-Netze (und damit auf höhere Petri-Netze im Allgemeinen) übertragen. Die Markierungen von Ziel-Stellen wurden dabei genauso behandelt wie die Markierungen von Prädikat-Stellen und durch entsprechende Bedingungen im Vorbereich und im Nachbereich von Ereignissen repräsentiert. Die Verwendung von variablen Prädikaten in den Wächtern von Transitionen wurde durch kausale Abhängigkeiten zwischen den Transitionen und den assoziierenden Stellen aufgelöst. Auf diese Weise bleibt die charakteristische Form von Protkollnetzen erhalten und alle Eigenschaften, die in [Cze97] für Protokollnetze von S/T-Systemen gezeigt werden, lassen sich direkt auf Protokollnetze für Ziel/Transitions-Netze übertragen. Dies betrifft insbesondere den Begriff der Dichte sowie die Komposition und Zerlegung von Protkollnetzen. Anschließend wurde die Bildung von Linien und Schnitten, die zentrale

Begriffe in Protokollnetzen darstellen, diskutiert und Änderungen, die sich aufgrund der Verwendung von variablen Prädikaten ergeben, behandelt.

Auf der Grundlage der entwickelten Ablaufdarstellung wurde schließlich ein Zusammenhang zwischen den Zielen eines Agenten und den Zuständen des Systems erklärt. So lies sich der Begriff der Zielorientierung von Systemabläufen formal erfassen. Zu diesem Zweck wurden Formeln auf den Ziel-Stellen eines Ziel/Transitions-Netzes als Ziele oder als Antiziele interpretiert. An einem weiteren klassischen Anwendungsbeispiel der Agenten-Literatur wurde gezeigt, wie zielorientiertes Verhalten in den Abläufen eines Agenten anhand von Protkollnetzen überprüft werden kann.

Statt Ziele und Antiziele *eines* Agenten zu untersuchen, ist es darüber hinaus auch möglich, gemeinsame Ziele und Antiziele in einem Multiagentensystem zu formulieren und die Zielorientierung des gemeinsamen Handelns zu untersuchen. Die entsprechenden Agenten teilen sich in diesem Fall eine oder mehrere Ziel-Stellen.

5.2 Ausblick

In Zukunft steht vor allem die Evaluation des vorgeschlagenen Ansatzes in komplexeren Anwendungsbeispielen im Vordergrund. In diesem Zusammenhang kann auch die Entwicklung eines Werkzeugs für die Modellierung von Ziel/Transitions-Netzen von Bedeutung sein. Ein solches Werkzeug sollte zudem die Darstellung von Systemabläufen in Form von Protkollnetzen unterstützen und die entsprechenden Abläufe automatisch auf ihre Zielorientierung untersuchen können. Auch die Einbeziehung von Rückwärtsschritten bei der Konstruktion von Protokollnetzen für Ziel/Transition-Netze könnte in zukünftigen Arbeiten untersucht werden. Darüber hinaus besteht weiterer Forschungsbedarf in der Integration von anderen Ansätzen zur Modellierung von Agentensystemen mit Petri-Netzen. Dabei scheinen insbesondere die Arbeiten, die auf Referenz-Netzen basieren, vielversprechend.

Literaturverzeichnis

[AEH+99] M. Andries, G. Engels, A. Habel, B. Hoffmann, H.-J. Kreowski, S. Kuske, D. Plump, A. Schürr, and G. Taentzer. Graph transformation for specification and programming. *Science of Computer Programming*, 34(1):1–54, 1999.

[Age79] T. Agerwala. Putting petri nets to work. *IEEE Computer*, 12(12):85–94, 1979.

[Age11a] The Foundation for Intelligent Physical Agents. *FIPA Official Site*, 2011. `http://www.fipa.org/`.

[Age11b] The Foundation for Intelligent Physical Agents. *Publicly Available Implementations of FIPA Specifications*, 2011. `http://www.fipa.org/resources/livesystems.html`.

[Age11c] The Foundation for Intelligent Physical Agents. *Standard Status Specifications*, 2011. `http://www.fipa.org/repository/standardspecs.html`.

[Aus62] J. L. Austin. *How To Do Things With Words*. Oxford University Press, 1962.

[Bal96] H. Balzert. *Lehrbuch der Software-Technik, Band 1, Software-Entwicklung*. Spektrum, 1996.

[Bau96] B. Baumgarten. *Petri-Netze: Grundlagen und Anwendungen*. Spektrum Akademischer Verlag, 1996.

[BB05] M. Bourahla and M. Benmohamed. Model checking multi-agent systems. *Informatica Slovenia*, 29(2):189–198, 2005.

[BBMO02] B. Bauer, F. Bergenti, P. Massonet, and J. Odell. Agents and the UML: A unified notation for agents and multi-agent systems? In M. Wooldridge, G. Weiss, and P. Ciancarini, editors, *Agent-Oriented Software Engineering II: Second International Workshop*, volume 2222 of *Lecture Notes in Computer Science*, pages 101–118. Springer, 2002.

[BD08] T. M. Behrens and J. Dix. Model checking multi-agent systems with logic based petri nets. *Annals of Mathematics and Artificial Intelligence*, 51(2-4):81–121, 2008.

[Ber04] M. Bergman. Petri nets as modeling tool for emergent agents. In M. G. Hinchey, J. L. Rash, W. Truszkowski, and C. Rouff, editors, *Formal Approaches to Agent-Based Systems: Third International Workshop*, volume 3228 of *Lecture Notes in Computer Science*, pages 271–274. Springer, 2004.

[BGS98] M. Benerecetti, F. Giunchiglia, and L. Serafini. A model checking algorithm for multi-agent systems. In J. P. Müller, M. P. Singh, and A. Rao, editors, *Intelligent Agents V: Fifth International Workshop on Agent Theories, Architectures, and Languages*, volume 1555 of *Lecture Notes in Computer Science*, pages 163–176. Springer, 1998.

[BLMJ+01] M. Beaudouin-Lafon, W. E. Mackay, M. Jensen, P. Andersen, P. Janecek, H. Lassen, K. Lund, K. Mortensen, S. Munck, A. V. Ratzer, K. Ravn, S. Christensen, and K. Jensen. CPN/Tools: A tool for editing and simulating coloured petri nets. In T. Margaria and W. Yi, editors, *Tools and Algorithms for the Construction and Analysis of Systems: 7th International Conference*, volume 2031 of *Lecture Notes in Computer Science*, pages 574–577. Springer, 2001.

[BM03] B. Bauer and J. P. Müller. Using UML in the context of agent-oriented software engineering: State of the art. In P. Giorgini, J. P. Müller, and J. Odell, editors, *Agent-Oriented Software Engineering IV: 4th International Workshop*, volume 2935 of *Lecture Notes in Computer Science*, pages 1–24. Springer, 2003.

[BM04] M. Bidoit and P. D. Mosses. *CASL User Manual - Introduction to Using the Common Algebraic Specification Language*, volume 2900 of *Lecture Notes in Computer Science*. Springer, 2004.

[BMO00] B. Bauer, J. Müller, and J. Odell. Agent UML: A formalism for specifying multiagent software systems. In P. Ciancarini and M. J. Wooldridge, editors, *Agent-Oriented Software Engineering: First International Workshop*, volume 1957 of *Lecture Notes in Computer Science*, pages 91–104. Springer, 2000.

[Boo93] G. Booch. *Object-Oriented Analysis and Design with Applications.* Addison-Wesley, 1993.

[BP01] L. Baresi and M. Pezzè. On formalizing UML with high-level petri nets. In G. Agha, F. de Cindio, and G. Rozenberg, editors, *Concurrent Object-Oriented Programming and Petri Nets. Advances in Petri Nets*, volume 2001 of *Lecture Notes in Computer Science*, pages 276–304. Springer, 2001.

[Bra87] M. E. Bratman. *Intentions, Plans, and Practical Reason.* Harvard University Press, 1987.

[Bro91a] R. A. Brooks. Intelligence without reason. In J. Myopoulos and R. Reiter, editors, *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 569–595. Morgan Kaufmann, 1991.

[Bro91b] Rodney A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47(1-3):139–159, 1991.

[BVF+03] R. H. Bordini, W. Visser, M. Fisher, C. Pardavila, and M. Wooldridge. Model checking multi-agent programs with CASP. In W. A. Hunt Jr. and F. Somenzi, editors, *Computer Aided Verification: 15th International Conference*, volume 2725 of *Lecture Notes in Computer Science*, pages 110–113. Springer, 2003.

[Cas97] C. Castelfranchi. To be or not to be an agent. In J. Müller, M. Wooldridge, and N. Jennings, editors, *Intelligent Agents III: Third International Workshop on Agent Theories, Architectures, and Languages*, volume 1193 of *Lecture Notes in Computer Science*, pages 37–39. Springer, 1997.

[CCF+00] R. Scott Cost, Y. Chen, T. W. Finin, Y. Labrou, and Y. Peng. Modeling agent conversations with colored petri nets. In F. Dignum and M. Greaves, editors, *Issues in Agent Communication*, volume 1916 of *Lecture Notes in Computer Science*, pages 178–192. Springer, 2000.

[CDFL92] B. Corbara, A. Drogoul, D. Fresneau, and S. Lalande. Simulating the sociogenesis process in ant colonies with MANTA. In P. Bourgine and F. J. Varela, editors, *Toward A Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, pages 224–235. MIT Press, 1992.

[CH94] S. Christensen and N. D. Hansen. Coloured petri nets extended with channels for synchronous communication. In R. Valette, editor, *Application and Theory of Petri Nets 1994: 15th International Conference*, volume 815 of *Lecture Notes in Computer Science*, pages 159–178. Springer, 1994.

[CHSB96] W. Chainbi, C. Hanachi, and C. Sibertin-Blanc. The multi-agent prey/predator problem: A petri net solution. In P. Borne, J. C. Gentina, E. Craye, and S. El Khattabi, editors, *Proceedings of the CESA'96 Conference, Computational Engineering in Systems Applications*, pages 291–299. IEEE Computer Society, 1996.

[CL90] P. R. Cohen and H. J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42(2-3):213–261, 1990.

[CM04] L. Cabac and D. Moldt. Formal semantics for AUML agent interaction protocol diagrams. In J. Odell, P. Giorgini, and J. P. Müller, editors, *Agent-Oriented Software Engineering V: 5th International Workshop*, volume 3382 of *Lecture Notes in Computer Science*, pages 47–61. Springer, 2004.

[CMR03] L. Cabac, D. Moldt, and H. Rölke. A proposal for structuring petri net-based agent interaction protocols. In W. Aalst and E. Best, editors, *Applications and Theory of Petri Nets 2003: 24th International Conference*, volume 1871 of *Lecture Notes in Computer Science*, pages 102–120. Springer, 2003.

[CW01] P. Ciancarini and M. Wooldridge, editors. *Agent-Oriented Software Engineering: First International Workshop*, volume 1957 of *Lecture Notes in Computer Science*. Springer, 2001.

[Cze97] S. Czernik. *Protokollnetze - Eine Verallgemeinerung des Prozessbegriffs für Petrinetze*. PhD thesis, Universität Bremen, Fachbereich Mathematik und Informatik, 1997.

[Des00] J. Desel. Validation of process models by construction of process nets. In W. M. P. van der Aalst, J. Desel, and A. Oberweis, editors, *Business Process Management: Models, Techniques, and Empirical Studies*, volume 1806 of *Lecture Notes in Computer Science*, pages 110–128. Springer, 2000.

[Des02a] J. Desel. Model validation - a theoretical issue? In J. Esparza and C. Lakos, editors, *Application and Theory of Petri Nets 2002: 23rd International Conference*, volume 2360 of *Lecture Notes in Computer Science*, pages 23–43. Springer, 2002.

[Des02b] J. Desel. Validierug und Verifikation von Prozessmodellen. In J. Desel and M. Weske, editors, *Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen: GI-Workshop*, pages 78–80, 2002.

[dFL+97] M. d'inverno, M. Fisher, A. Lomuscio, M. Luck, M. de Rijke, M. Ryan, and M. Wooldridge. Formalisms for multi-agent systems. *The Knowledge Engineering Review*, 12(3):315–321, 1997.

[DH00] R. Depke and R. Heckel. Formalizing the development of agent-based systems using graph processes. In *Proceedings of the ICALP2000 Workshop on Graph Transformation and Visual Modelling Techniques*, pages 419–426, 2000.

[DH04] R. Depke and R. Heckel. Modelling and analysis of agents' goal-driven behavior. In M. D. Ryan, J.-J. C. Meyer, and H.-D. Ehrichs, editors, *Objects, Agents, and Features: International Seminar*, volume 2975 of *Lecture Notes in Computer Science*, pages 81–97. Springer, 2004.

[DHK01] R. Depke, R. Heckel, and J. Malte Küster. Agent-oriented modeling with graph transformation. In P. Ciancarini and Michael Wooldridge, editors, *Agent-Oriented Software Engineering: First International Workshop*, volume 1957 of *Lecture Notes in Computer Science*, pages 105–120. Springer, 2001.

[DHK02] R. Depke, R. Heckel, and J. M. Küster. Formal agent-oriented modeling with graph transformation. *Science of Computer Programming*, 44:229–252, 2002.

[DP04] G. Denaro and M. Pezzè. Petri nets and software engineering. In J. Desel, W. Reisig, and G. Rozenberg, editors, *Lectures on Concurrency and Petri Nets. Advances in Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 439–466. Springer, 2004.

[DR98] J. Desel and W. Reisig. Place/transition petri nets. In W. Reisig and G. Rozenberg, editors, *Lectures on Petri Nets I: Basic Models. Advances in Petri Nets*, volume 1491 of *Lecture Notes in Computer Science*, pages 122–173. Springer, 1998.

[Dur99] E. H. Durfee. Distributed problem solving and planing. In G. Weiss, editor, *Multiagent Systems: A Modern Approach to Distributed Artifical Intelligence*, pages 121–164. MIT Press, 1999.

[Dur01] E. H. Durfee. Distributed problem solving and planning. In M. Luck, V. Marik, O. Stepankova, and R. Trappl, editors, *Multi-Agent Systems and Applications: 9th ECCAI Advanced Course, ACAI 2001 and Agent Link's 3rd European Agent Systems Summer School*, volume 2086 of *Lecture Notes in Computer Science*, pages 118–149. Springer, 2001.

[EEKR99] H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 2: Applications, Languages and Tools.* World Scientific, 1999.

[EKMR99] H. Ehrig, H.-J. Kreowski, U. Montanari, and G. Rozenberg. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 3: Concurrency, Parallelism, and Distribution.* World Scientific, 1999.

[EM85] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification I. Equations and Initial Semantic.* Springer, 1985.

[EM90] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification II. Module specifications and constraints.* Springer, 1990.

[Eme90] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science (vol. B): Formal Models and Semantics*, pages 995–1072. MIT Press, 1990.

[EPR94] Hartmut Ehrig, Julia Padberg, and Leila Ribeiro. Algebraic high level nets: Petri nets revisited. In Hartmut Ehrig, editor, *Recent Trends*

in Data Type Specification: 9th Workshop on Specification of Abstract Data Types, volume 785 of *Lecture Notes in Computer Science*, pages 188–206. Springer, 1994.

[Esp94] J. Esparza. Model checking using net unfoldings. *Science of Computer Programming*, 23(2-3):151–195, 1994.

[Etz97] O. Etzioni. Moving up the information food chain: Deploying softbots on the world wide. *AI Magazine*, 18(2):11–18, 1997.

[FB98] J. M. Fernandes and O. Belo. Modeling of multi-agent system activities through colored petri nets. In *Proceedings of the 16th International Conference on Applied Informatics*, pages 17–20, 1998.

[Fer01] J. Ferber. *Multiagentensysteme - Eine Einführung in die verteilte künstliche Intelligenz*. Addison-Wesley, 2001.

[FG97] S. Franklin and A. Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In J. Müller, M. Wooldridge, and N. Jennings, editors, *Intelligent Agents III: Third International Workshop on Agent Theories, Architectures, and Languages*, volume 1193 of *Lecture Notes in Computer Science*, pages 21–35. Springer, 1997.

[Fis97] M. Fisher. If Z is the answer, what could the question possibly be? In J. Müller, M. Wooldridge, and N. Jennings, editors, *Intelligent Agents III: Third International Workshop on Agent Theories, Architectures, and Languages*, volume 1193 of *Lecture Notes in Computer Science*, pages 65–66. Springer, 1997.

[Gar70] M. Gardner. Mathematical games: The fantastic combinations of john conway's new solitaire game "life". *Scientific American*, 223:120–123, 1970.

[Gen87] H. J. Genrich. Predicate/transition nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Central Models and Their Properties. Advances in Petri Nets*, volume 254 of *Lecture Notes in Computer Science*, pages 207–247. Springer, 1987.

[GHB00] M. Greaves, H. Holmback, and J. Bradshaw. What is a conversation policy? In F. Dignum and M. Greaves, editors, *Issues in Agent Communication*, volume 1916 of *Lecture Notes in Computer Science*, pages

118–131. Springer, 2000.

[GJM91] C. Ghezzi, M. Jazayeri, and D. Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, 1991.

[GL81] H. J. Genrich and K. Lautenbach. System modelling with high-level petri nets. *Theoretical Computer Science*, 13:109–136, 1981.

[GL83] H. J. Genrich and K. Lautenbach. S-invariance in predicate/transition nets. In Anastasia Pagnoni and Grzegorz Rozenberg, editors, *Applications and Theory of Petri Nets: Selected Papers from the 3rd European Workshop on Applications and Theory of Petri Nets*, volume 66 of *Informatik-Fachberichte*, pages 89–111. Springer, 1983.

[GM00] J. Goguen and G. Malcolm, editors. *Software Engineering with OBJ: Algebraic Specification in Action*. Springer, 2000.

[GMO03] P. Giorgini, J. P. Müller, and J. Odell, editors. *Agent-Oriented Software Engineering IV: 4th International Workshop*, volume 2935 of *Lecture Notes in Computer Science*. Springer, 2003.

[GOW03] F. Giunchiglia, J. Odell, and G. Weiß, editors. *Agent-Oriented Software Engineering III: Third International Workshop*, volume 2585 of *Lecture Notes in Computer Science*. Springer, 2003.

[GR83] U. Goltz and W. Reisig. Processes of place/transition-nets. In J. Díaz, editor, *Automata, Languages and Programming*, volume 154 of *Lecture Notes in Computer Science*, pages 264–277. Springer, 1983.

[Gro11] The Object Management Group. *UML Infrastructure and Superstructure Specification 2.3*, 2011. `http://www.omg.org/spec/UML/2.3/`.

[GS95] M. Gerhardt and H. Schuster. *Das digitale Universum: Zelluläre Automaten als Modelle der Natur*. Vieweg, 1995.

[GTM76] H. J. Genrich and G. Thieler-Mevissen. The calculus of facts. In A. W. Mazurkiewicz, editor, *Mathematical Foundations of Computer Science 1976: 5th Symposium*, volume 45 of *Lecture Notes in Computer Science*, pages 588–595. Springer, 1976.

[HHT96] A. Habel, R. Heckel, and G. Taentzer. Graph grammars with negative application conditions. *Fundamenta Informaticae*, 26(3/4):287–313,

1996.

[Hin62] J. Hintikka. *Knowledge and Belief*. Cornell University Press, 1962.

[HKHK+07] K. Hölscher, R. Klempien-Hinrichs, P. Knirsch, H.-J. Kreowski, and S. Kuske. Autonomous units: Basic concepts and semantic foundation. In M. Hülsmann and K. Windt, editors, *Understanding Autonomous Cooperation and Control in Logistics - The Impact of Autonomy on Management, Information, Communication and Material Flow*, pages 103–120. Springer, 2007.

[HKK06] K. Hölscher, H.-J. Kreowski, and S. Kuske. Autonomous units and their semantics - the sequential case. In A. Corradini, H. Ehrig, U. Montanari, L. Ribeiro, and G. Rozenberg, editors, *Graph Transformations: Third International Conference*, volume 4178 of *Lecture Notes in Computer Science*, pages 245–259. Springer, 2006.

[HKK09] K. Hölscher, H.-J. Kreowski, and S. Kuske. Autonomous units to model interacting sequential and parallel processes. *Fundamenta Informaticae*, 92(3):233–257, 2009.

[Hoa69] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–583, 1969.

[Hol95] T. Holvoet. Agents and petri nets. *Petri net Newsletter*, 49, 1995.

[Hol03] G. J. Holzmann. *The Spin Model Checker: Primer and Reference Manual*. Addison-Wesley, 2003.

[HS99a] M. N. Huhns and M. Singh. A multiagent treatment of agenthood. *Applied Artificial Intelligence*, 13:3–10, 1999.

[HS99b] M. N. Huhns and L. M. Stephens. Multiagent systems and societies of agents. In G. Weiss, editor, *Multiagent Systems: A Modern Approach to Distributed Artifical Intelligence*, pages 79–120. MIT Press, 1999.

[HV91] Joseph Y. Halpern and Moshe Y. Vardi. Model checking vs. theorem proving: A manifesto. In J. Allen, R. E. Fikes, and E. Sandewall, editors, *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, pages 325–334. Morgan Kaufmann, 1991.

[HWP01] M. Hannebauer, J. Wendler, and E. Pagello. Balancing reactivity and social deliberation in multi-agent systems - a short guide to the contributions. In M. Hannebauer, J. Wendler, and E. Pagello, editors, *Balancing Reactivity and Social Deliberation in Multi-Agent Systems: From RoboCup to Real-World Applications*, volume 2103 of *Lecture Notes in Computer Science*, pages 3–8. Springer, 2001.

[Jen81] K. Jensen. Coloured petri nets and the invariant method. *Theoretical Computer Science*, 14:317–336, 1981.

[Jen87] K. Jensen. Coloured petri nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Central Models and Their Properties. Advances in Petri Nets*, volume 254 of *Lecture Notes in Computer Science*, pages 248–299. Springer, 1987.

[Jen92] K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, volume 1. Springer, 1992.

[Jen94] K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, volume 2. Springer, 1994.

[Jen97] K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, volume 3. Springer, 1997.

[Jon90] C. B. Jones. *Systematic Software Development Using VDM*. Prentice Hall, 2nd edition, 1990.

[JSW98] N. R. Jennings, K. Sycara, and M. J. Wooldridge. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1:7–38, 1998.

[JW98] N. R. Jennings and M. J. Wooldridge, editors. *Agent technology: Foundations, Applications and Markets*. Springer, 1998.

[KG97] D. Kinny and M. Georgeff. Modelling and design of multi-agent systems. In J. Müller, M. Wooldridge, and N. Jennings, editors, *Intelligent Agents III: Third International Workshop on Agent Theories, Architectures, and Languages*, volume 1193 of *Lecture Notes in Computer Science*, pages 41–43. Springer, 1997.

[KGKZ09] S. Kuske, M. Gogolla, H.-J. Kreowski, and P. Ziemann. Towards an integrated graph-based semantics for uml. *Software and Systems Mo-*

deling, 8(3):403–422, 2009.

[KK00] P. Knirsch and H.-J. Kreowski. A note on modeling agent systems by graph transformation. In M. Nagl, A. Schurr, and M. Munch, editors, *Applications of Graph Transformations with Industrial Relevance: International Workshop*, volume 1779 of *Lecture Notes in Computer Science*, pages 79–86. Springer, 2000.

[KK07] H.-J. Kreowski and S. Kuske. Autonomous units and their semantics - the parallel case. In J. Fiadeiro and P. Schobbens, editors, *Recent Trends in Algebraic Development Techniques, 18th International Workshop*, volume 4409 of *Lecture Notes in Computer Science*, pages 56–73. Springer, 2007.

[KK10] H.-J. Kreowski and S. Kuske. Autonomous units and their semantics - the concurrent case. In G. Engels, C. Lewerentz, W. Schäfer, A. Schürr, and B. Westfechtel, editors, *Graph Transformations and Model-Driven Engineering - Essays Dedicated to Manfred Nagl on the Occasion of his 65th Birthday*, volume 5765 of *Lecture Notes in Computer Science*, pages 102–120. Springer, 2010.

[Klü04] F. Klügl. Applications of software agents. *Künstliche Intelligenz*, 18(2):5–10, 2004.

[KMR01] M. Köhler, D. Moldt, and H. Rölke. Modelling the structure and behaviour of petri net agents. In J. M. Colom and M. Koutny, editors, *Applications and Theory of Petri Nets 2001: 22nd International Conference*, volume 2075 of *Lecture Notes in Computer Science*, pages 224–241. Springer, 2001.

[KMR03] M. Köhler, D. Moldt, and H. Rölke. Modelling mobility and mobile agents using nets within nets. In W. Aalst and E. Best, editors, *Applications and Theory of Petri Nets 2003: 24th International Conference*, volume 2679 of *Lecture Notes in Computer Science*, pages 121–139. Springer, 2003.

[KR91] L.P. Kaelbling and S.J Rosenschein. Action and planing in embedded agents. In P. Maes, editor, *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*, pages 35–48. MIT Press, 1991.

[Kre91] H.-J. Kreowski. *Logische Grundlagen der Informatik*. Oldenbourg, 1991.

[Kre02] H.-J. Kreowski. A sight-seeing tour of the computational landscape of graph transformations. In W. Brauer, H. Ehrig, J. Karhumäki, and A. Salomaa, editors, *Formal and Natural Computing - Essays Dedicated to Grzegorz Rozenberg*, volume 2300 of *Lecture Notes in Computer Science*, pages 119–137. Springer, 2002.

[KRH03] J.-L. Koning and I. Romero-Hernandez. Generating machine processable representations of textual representations of AUML. In F. Giunchiglia, J. Odell, and G. Weiß, editors, *Agent-Oriented Software Engineering III: Third International Workshop*, volume 2585 of *Lecture Notes in Computer Science*, pages 126–137. Springer, 2003.

[Kum02] O. Kummer. *Referenznetze*. Logos, 2002.

[KWD+04] O. Kummer, F. Wienberg, M. Duvigneau, J. Schumacher, M. Köhler, D. Moldt, H. Rölke, and R. Valk. An extensible editor and simulation engine for petri nets: Renew. In J. Cortadella and W. Reisig, editors, *Applications and Theory of Petri Nets 2004: 25th International Conference*, volume 3099 of *Lecture Notes in Computer Science*, pages 484–493. Springer, 2004.

[LB79] B. H. Liskov and V. Berzinsr. An appraisal of program specifications. In P. Wegner, editor, *Research Directions in Software Technology*, pages 276–301. MIT, 1979.

[LGS09] M. Luck and J. J. Gomez-Sanz, editors. *Agent-Oriented Software Engineering IX: 9th International Workshop*, volume 5386 of *Lecture Notes in Computer Science*. Springer, 2009.

[Lin89] M. Lindqvist. Parameterized reachability trees for predicate/transition nets. *Acta Polytechnica Scandinavica: Mathematics and Computer Science Series*, 54:1–120, 1989.

[Lin93] Markus Lindqvist. Parameterized reachability trees for predicate/transition nets. In Grzegorz Rozenberg, editor, *Advances in Petri Nets 1993: 12th International Conference on Applications and Theory of Petri Nets*, volume 674 of *Lecture Notes in Computer Science*, pages 301–324. Springer, 1993.

[LM04] K. Lehmann and D. Moldt. Modelling and analysis of agent protocols with petri nets. In G. Lindemann, J. Denzinger, I. J. Timm, and R. Unland, editors, *Multiagent System Technologies: Second German Conference*, volume 3187 of *Lecture Notes in Computer Science*, pages 85–98. Springer, 2004.

[LP08] M. Luck and L. Padgham, editors. *Agent-Oriented Software Engineering VIII: 8th International Workshop*, volume 4951 of *Lecture Notes in Computer Science*. Springer, 2008.

[Mae95] P. Maes. Artificial life meets entertainment: Life like autonomous agents. *Communications of the ACM*, 38(11):108–114, 1995.

[Mev82] H. Mevissen. *Algebraische Bestimmung von S-Invarianten in Prädikat/Transitions-Netzen*. ISF-Report 81.01, Gesellschaft für Mathematik und Datenverarbeitung, 2nd edition, 1982.

[MK96] T. Miyamoto and S. Kumagai. A multi agent net model of autonomous distributed systems. In *Proceedings of CESA'96, Symposium on Disrcete Events and Manufacturing Systemes*, pages 619–623, 1996.

[MK99] T. Miyamoto and S. Kumagai. A multi agent net model and the realization of software environment. In *Proceedings of the Workshop of Petri Nets to Intelligent System Development in Application with 20th International Conference on Application and Theory of Petri Nets*, pages 83–92, 1999.

[MK06] S. Kühling M. Kreuzer. *Logik für Informatiker*. Pearson Studium, 2006.

[MML07] T. Mossakowski, C. Maeder, and K. Lüttich. The heterogeneous tool set, hets. In O. Grumberg and M. Huth, editors, *Tools and Algorithms for the Construction and Analysis of Systems: 13th International Conference*, volume 4424 of *Lecture Notes in Computer Science*, pages 519–522. Springer, 2007.

[MP99] M. V. Costa Miranda and A. Perkusich. Modeling and analysis of a multi-agent system using colored petri nets. In *Proceedings of the Workshop of Petri Nets to Intelligent System Development in Application with 20th International Conference on Application and Theory of Petri Nets*, pages 68–78, 1999.

[MPHG02] T. Menzies, A. R. Pearce, C. Heinze, and S. Goss. What is an agent and why should i care? In M. G. Hinchey, J. L. Rash, W. Truszkowski, C. Rouff, and D. F. Gordon-Spears, editors, *Formal Approaches to Agent-Based Systems: Second International Workshop*, volume 2699 of *Lecture Notes in Computer Science*, pages 439–466. Springer, 2002.

[MW97] D. Moldt and F. Wienberg. Multi-agent-systems based on coloured petri nets. In P. Azema and G. Balbo, editors, *Application and Theory of Petri Nets 1997: 18th International Conference*, volume 1248 of *Lecture Notes in Computer Science*, pages 241–249. Springer, 1997.

[MZ06] J. P. Müller and F. Zambonelli, editors. *Agent-Oriented Software Engineering VI: 6th International Workshop*, volume 3950 of *Lecture Notes in Computer Science*. Springer, 2006.

[Mül93] J. Müller, editor. *Verteilte künstliche Intelligenz: Methoden und Anwendungen*. BI-Wissenschaftsverlag, 1993.

[New62] A. Newell. Some problems of the basic organization in problem-solving programs. In M.C. Yovits, G.T. Jocobi, and G.D. Goldstien, editors, *Proceedings of the Second Conference on Self-Organizing Systems*, pages 393–423, 1962.

[NPC01] M. Nowostawski, M. Purvis, and S. Cranefield. A layered approach for modelling agent conversations. In *Proceedings of the 2nd International Workshop on Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems*, pages 163–170, 2001.

[NV86] Y. Narahari and N. Viswanadham. On the invariants of coloured petri nets. In G. Rozenberg, editor, *Application and Theory of Petri Nets 1986: 6th International Conference*, volume 222 of *Lecture Notes in Computer Science*, pages 330–345. Springer, 1986.

[OGM04] J. Odell, P. Giorgini, and J. P. Müller, editors. *Agent-Oriented Software Engineering V: 5th International Workshop*, volume 3382 of *Lecture Notes in Computer Science*. Springer, 2004.

[Par99] H. V. D. Parunak. Industrial and practical applications of DAI. In G. Weiss, editor, *Multiagent Systems: A Modern Approach to Distributed Artifical Intelligence*, pages 377–421. MIT Press, 1999.

[Pet62] C. A. Petri. *Kommunikation mit Automaten*. Institut für Instrumentelle Mathematik, Universität Bonn, 1962.

[Pet77] C. A. Petri. *Nicht-sequentielle Prozesse*. Bericht ISF-76-6, Gesellschaft für Mathematik und Datenverarbeitung, 3. edition, 1977.

[Pet96] C. J. Petrie. Agent-based engineering, the web, and intelligence. *IEEE Expert*, 11(6):24–29, 1996.

[Pet97] C. J. Petrie. What is an agent? In J. Müller, M. Wooldridge, and N. Jennings, editors, *Intelligent Agents III: Third International Workshop on Agent Theories, Architectures, and Languages*, volume 1193 of *Lecture Notes in Computer Science*, pages 41–43. Springer, 1997.

[Pet11] Petri Nets World. *Petri Nets Tools and Software*, 2011. `http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/`.

[PGK+97] B. Pell, E. Gat, R. Keesing, N. Muscettola, and B. D. Smith. Robust periodic planning and execution for autonomous spacecraft. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 1234–1239. Morgan Kaufmann, 1997.

[Pre04] R. S. Pressman. *Software Engineering: A Practitioner's Approach*. Mcgraw-Hill, 5. edition, 2004.

[PW03] L. Priese and H. Wimmel. *Theoretische Informatik: Petri-Netze*. Spinger, 2003.

[PZ07] L. Padgham and F. Zambonelli, editors. *Agent-Oriented Software Engineering VII: 7th International Workshop*, volume 4405 of *Lecture Notes in Computer Science*. Springer, 2007.

[Ram98] D. Ramamonjisoa. Multi-agent coordination with OASIS. In T. Ishida, editor, *Multiagent Platforms: First Pacific Rim International Workshop on Multi-Agents*, pages 103–117, 1998.

[Rei82] W. Reisig. *Petrinetze: Eine Einführung*. Springer, 1982.

[Rei85] Wolfgang Reisig. On the semantics of petri nets. *Formal Models in Programming*, pages 347–372, 1985.

[Rei95] W. Reisig. Petri net models of distributed algorithms. In J. van Leeuwen, editor, *Computer Science Today: Recent Trends and Deve-*

lopments, volume 1000 of *Lecture Notes in Computer Science*, pages 441–454. Springer, 1995.

[RG91] A. S. Rao and M. P. Georgeff. Modeling rational agents within a BDI-architecture. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, pages 473–484. Morgan Kaufmann, 1991.

[RJB04] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Springer, 2nd edition, 2004.

[RN04] S. Russell and P. Norvig. *Künstliche Intelligenz - Ein moderner Ansatz*. Pearson Studium, 2004.

[Roz97] G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*. World Scientific, 1997.

[Sch95] K. Schmidt. Parameterized reachability trees for algebraic petri nets. In G. De Michelis and M. Diaz, editors, *Application and Theory of Petri Nets 1995: 16th International Conference*, volume 935 of *Lecture Notes in Computer Science*, pages 392–411. Springer, 1995.

[Sea90] J. R. Searle. *Speech Acts: an Essay in the Philosophy of Language*. Cambridge University Press, 1990.

[Sho93] Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60(1):51–92, 1993.

[Smi88] R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. In A. H. Bond and L. Gasser, editors, *Readings in Distributed Artificial Intelligence*. Morgan Kaufmann, 1988.

[Spi92] J. M. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall, 2nd edition, 1992.

[SRG99] M. P. Singh, A. S. Rao, and M. P. Georgeff. Formal methods in DAI: Logic-based representation and reasoning. In G. Weiss, editor, *Multiagent Systems: A Modern Approach to Distributed Artifical Intelligence*, pages 331–376. MIT Press, 1999.

[Sta90] P. Starke. *Anaylse von Petri-Netz-Modellen.* Teubner, 1990.

[Ste90a] L. Steels. Cooperation between distributed agents through self-organisation. In Y. Demazeau and J. P. Müller, editors, *Decentralized AI: Proceedings of the First European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, pages 175–196. Elsevier, 1990.

[Ste90b] L. Steels. Cooperation between distributed agents through self-organization. In Y. Demazeau and J.-P. Müller, editors, *Decentralized AI: Proceedings of the First European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, pages 175–196, 1990.

[Sum07] I. Summerville. *Software Engineering.* Pearson Studium, 8. edition, 2007.

[The11] The DARPA Knowledge Sharing Initiative. *KQML Knowledge Query and Manipulation Language*, 2011. `http://www.cs.umbc.edu/research/kqml/`.

[Tve01] A. Tveit. A survey of agent-oriented software engineering. In *Proceedings of the First NTNU Computer Science Graduate Student Conference*, 2001.

[UML11] Agent UML. *AUML Web Site*, 2011. `http://www.auml.org/`.

[Val98] R. Valk. Petri nets as token objects: An introduction to elementary object nets. In J. Desel and M. Silva, editors, *Application and Theory of Petri Nets 1998: 19th International Conference*, volume 1420 of *Lecture Notes in Computer Science*, pages 1–25. Springer, 1998.

[vdH01] W. van der Hoek. Logical foundations of agent-based computing. In M. Luck, V. Marik, O. Stepankova, and R. Trappl, editors, *Multi-Agent Systems and Applications: 9th ECCAI Advanced Course, ACAI 2001 and Agent Link's 3rd European Agent Systems Summer School*, volume 2086 of *Lecture Notes in Computer Science*, pages 50–73. Springer, 2001.

[VW95] R. Vollmar and T. Worsch. *Modelle der Parallelverarbeitung.* Teubner, 1995.

[VW06] G. Vossen and K.-U. Witt. *Grundkurs Theoretische Informatik.* Viweg, 4th edition, 2006.

[Wag97] G. Wagner. Practical theory and theory-based practice. In J. Müller, M. Wooldridge, and N. Jennings, editors, *Intelligent Agents III: Third International Workshop on Agent Theories, Architectures, and Languages*, volume 1193 of *Lecture Notes in Computer Science*, pages 41–43. Springer, 1997.

[WC00] M. Wooldridge and P. Ciancarini. Agent-oriented software engineering: The state of the art. In P. Ciancarini and M. J. Wooldridge, editors, *Agent-Oriented Software Engineering: First International Workshop*, volume 1957 of *Lecture Notes in Computer Science*, pages 105–119. Springer, 2000.

[WHFP06] M. Wooldridge, M.-P. Huget, M. Fisher, and S. Parsonse. Model checking multi-agent systems: The MABLE language and its applications. *International Journal on Artificial Intelligence Tools*, 15(2):195–225, 2006.

[Win90] J. M. Wing. A specifier's introduction to formal methods. *Computer*, 23(9):8–23, 1990.

[WJ95] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10:115–152, 1995.

[WJ04] G. Weiss and R. Jacob. *Agentenorientierte Softwareentwicklung: Methoden und Tools*. Springer, 2004.

[WJK00] M. Wooldridge, N. R. Jennings, and D. Kinny. The Gaia methodology for agent-oriented analysis and design. *International Journal on Artificial Intelligence Tools*, 3(3):285–312, 2000.

[Woo92] M. Wooldridge. *The Logical Modelling of Computational Multi-Agent Systems*. PhD thesis, University of Manchester, Department of Computation, 1992.

[Woo98] M. Wooldridge. Agents and software engineering. *AI*IA Notizie*, XI:31–37, 1998.

[Woo99] M. Wooldridge. Intelligent agents. In G. Weiss, editor, *Multiagent Systems: A Modern Approach to Distributed Artifical Intelligence*, pages 27–77. MIT Press, 1999.

[Woo02] M. Wooldridge. *An Introduction to MultiAgent Systems*. Wiley, 2002.

[WPHT02] M. Winikoff, L. Padgham, J. Harland, and J. Thangarajah. Declarative and procedural goals in intelligent agent systems. In *Proceedings of the 8th International Conference on Principles of Knowledge Representation and Reasoning*. Morgan Kaufman, 2002.

[WWC02] M. Wooldridge, G. Weiß, and P. Ciancarini, editors. *Agent-Oriented Software Engineering II: Second International Workshop*, volume 2222 of *Lecture Notes in Computer Science*. Springer, 2002.

[XD00] D. Xu and Y. Deng. Modeling mobile agent systems with high level petri nets. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*. IEEE Computer Society, 2000.

[XS01] H. Xu and S. M. Shatz. A framework for modeling agent-oriented software. In *Proceedings of the 21th International Conference on Distributed Computing Systems*, 2001.

[XS03] H. Xu and S. M. Shatz. A framework for model-based design of agent-oriented software. *IEEE Transactions on Software Engineering*, 29:15–30, 2003.

[XVIY02] D. Xu, R. Volz, T. Ioerger, and J. Yen. Modeling and verifying multi-agent behaviors using predicate transition nets. In *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering*, pages 193–200, 2002.

[XVIY03] D. Xu, R. Volz, T. Ioerger, and J. Yen. Modeling and analyzing multi-agent behaviors using predicate transition nets. *International Journal of Software Engineering and Knowledge Engineering*, 13:103–124, 2003.

[YC79] E. Yourdon and L.L. Constantine. *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*. Prentice Hall, 1979.

[ZHG05] P. Ziemann, K. Hölscher, and M. Gogolla. From UML models to graph transformation systems. *Electronic Notes in Theoretical Computer Science*, 127(4):17–33, 2005.