Vorlesung „Modellierung“

Prof. Janis Voigtländer

Wintersemester 2017/18

Ungefährer Überblick

1. Organisatorisches
2. Einführung in die Modellierung
3. Petrinetze
   - Grundlagen und Erreichbarkeitsgraphen
   - Eigenschaften von Petrinetzen, Überdeckungsgraphen
4. Unified Modeling Language (UML)
   - Einführung & Objekt-Orientierung
   - Klassen- und Objektdiagramme
   - Aktivitätsdiagramme
   - Zustandsdiagramme
   - Weitere UML-Diagramme

Organisatorisches

Mit wem haben Sie es hier zu tun?

Dozent: Prof. Dr. Janis Voigtländer

- Raum LF 233
- E-Mail: `janis.voigtlaender@uni-due.de`
- Sprechstunde: nach Vereinbarung

Übungsleitung: M.Sc. Marcel Fourné

- Raum LF 231B
- E-Mail: `marcel.fourne@uni-due.de`

Wie ist Ihre Zusammensetzung?

Meines Wissens:

- Bachelor-Studierende „Angewandte Informatik“
  (vorwiegend 1. oder 2. Semester)
- Bachelor-Studierende „Komedia“
  (vorwiegend 3. Semester)
- Bachelor-Studierende „ISE / Computer Engineering“
  (vorwiegend 3. oder 5. Semester)

Zur Lehrveranstaltung

Form:

- 2 V + 1 Ü
- Vorlesung: Vortrag großteils mit Beamer-Folien, großteils übernommen aus Vorjahren
- Präsenzübungen:
  Abgaben zu Übungszetteln mit anschließender Korrektur und Besprechung in den Übungsgruppen

Zur Lehrveranstaltung

Form:

- 2 V + 1 Ü
- Vorlesung: Vortrag großteils mit Beamer-Folien, großteils übernommen aus Vorjahren
- Präsenzübungen:
  Abgaben zu Übungszetteln mit anschließender Korrektur und Besprechung in den Übungsgruppen

Vorlesungstermin:

- Mittwoch, 14:15–15:45, in LB 104
- 14-mal

Lernplattform Moodle

Wir verwenden Moodle, um

- die Vorlesungsfolien jeweils aktuell zur Verfügung zu stellen,
- die Aufgabenblätter zur Verfügung zu stellen und
- Übungseinreichungen elektronisch abgeben zu lassen.

Lernplattform Moodle

Wir verwenden Moodle, um

- die Vorlesungsfolien jeweils aktuell zur Verfügung zu stellen,
- die Aufgabenblätter zur Verfügung zu stellen und
- Übungseinreichungen elektronisch abgeben zu lassen.
- Außerdem gibt es im Moodle-Kurs ein Diskussionsforum.

Lernplattform Moodle

Wir verwenden Moodle, um

- die Vorlesungsfolien jeweils aktuell zur Verfügung zu stellen,
- die Aufgabenblätter zur Verfügung zu stellen und
- Übungseinreichungen elektronisch abgeben zu lassen.
- Außerdem gibt es im Moodle-Kurs ein Diskussionsforum.

Link: https://moodle.uni-due.de/course/view.php?id=11514

Bitte legen Sie dort einen Zugang an (falls noch nicht vorhanden) und tragen Sie sich in den Kurs „Modellierung (WS17/18)“ ein (unter Wintersemester 2017/18 → Ingenieurwissenschaften → Informatik und Angewandte Kognitionswissenschaften).

Lernplattform Moodle

Wir verwenden Moodle, um

- die Vorlesungsfolien jeweils aktuell zur Verfügung zu stellen,
- die Aufgabenblätter zur Verfügung zu stellen und
- Übungseinreichungen elektronisch abgeben zu lassen.
- Außerdem gibt es im Moodle-Kurs ein Diskussionsforum.

Link: https://moodle.uni-due.de/course/view.php?id=11514

Bitte legen Sie dort einen Zugang an (falls noch nicht vorhanden) und tragen Sie sich in den Kurs „Modellierung (WS17/18)“ ein (unter Wintersemester 2017/18 → Ingenieurwissenschaften → Informatik und Angewandte Kognitionswissenschaften).

Tun Sie das noch heute, inklusive Ausfüllen der Umfrage, damit Sie sich morgen in die Übungsgruppen einschreiben können!

Hinweise zu den Folien

- Die Folien stehen in verschiedenen Formen zur Verfügung, etwa auch zum Ausdrucken mit Randzeilen für Notizen.
- Es ist nicht sinnvoll, all zu viele Folien im Voraus herunterzuladen, da diese noch nicht endgültig sind.

Termine der Übungsgruppen

Übungsgruppen (jeweils 45 Minuten im angegebenen Zeitraum):

Group 1:	Monday, 16:00–17:00, in LF 035 (given in English)
Gruppe 2:	Montag, 17:00–18:00, in LF 035
Gruppe 3:	Dienstag, 10:00–11:00, in LE 120
Gruppe 4:	Dienstag, 11:00–12:00, in LE 120
Gruppe 5:	Dienstag, 12:00–13:00, in LC 140
Gruppe 6:	Dienstag, 13:00–14:00, in LC 140
Gruppe 7:	Mittwoch, 10:00–11:00, in LE 120
Gruppe 8:	Mittwoch, 11:00–12:00, in LE 120
Gruppe 9:	Donnerstag, 14:00–15:00, in LF 035
Gruppe 10:	Donnerstag, 15:00–16:00, in LF 035
Gruppe 11:	Freitag, 10:00–11:00, in LK 052
Gruppe 12:	Freitag, 11:00–12:00, in LK 052

Die Übungen beginnen am 16.10.2017.

Hinweise zu den Übungen

- Die Anmeldung für die Übungen erfolgt über den Moodle-Kurs.

  12.10.2017, 18:00 – 13.10.2017, 23:55

  (Verfügbar für alle, die sich heute eintragen, inklusive Umfrage.)

- Sie müssen sich dort für eine Übungsgruppe anmelden, um an dieser teilnehmen zu können.

- Sie müssen an der ersten Übungssitzung teilnehmen. Anderenfalls wird Ihr Platz neu vergeben.

Hinweise zu den Übungen

- Besuchen Sie die Übungen und machen Sie die Aufgaben!
  Den Stoff kann man sich nur durch regelmäßiges Üben erschließen. Auswendiglernen hilft nicht besonders viel.

Hinweise zu den Übungen

- Besuchen Sie die Übungen und machen Sie die Aufgaben!
  Den Stoff kann man sich nur durch regelmäßiges Üben erschließen. Auswendiglernen hilft nicht besonders viel.
- Weitere Hilfestellung können Sie bei Bedarf im Lern- und Diskussionszentrum (LuDi Informatik) erhalten:
  LF 031, jederzeit als Arbeitsraum nutzbar, demnächst zu bestimmten Zeiten auch betreut (siehe https://udue.de/ludi).

Hinweise zu den Übungen

- Besuchen Sie die Übungen und machen Sie die Aufgaben!
  Den Stoff kann man sich nur durch regelmäßiges Üben erschließen. Auswendiglernen hilft nicht besonders viel.
- Weitere Hilfestellung können Sie bei Bedarf im Lern- und Diskussionszentrum (LuDi Informatik) erhalten:
  LF 031, jederzeit als Arbeitsraum nutzbar, demnächst zu bestimmten Zeiten auch betreut (siehe https://udue.de/ludi).
- Durch die Übungen können bis zu 10% Bonus für die Klausur erreicht werden. Voraussetzung ist nicht nur Einreichen von Lösungen, sondern auch Präsentation in der Übungsgruppe. Näheres dazu in der ersten Übung.

Hinweise zu den Übungen

- Besuchen Sie die Übungen und machen Sie die Aufgaben!
  Den Stoff kann man sich nur durch regelmäßiges Üben erschließen. Auswendiglernen hilft nicht besonders viel.
- Weitere Hilfestellung können Sie bei Bedarf im Lern- und Diskussionszentrum (LuDi Informatik) erhalten:
  LF 031, jederzeit als Arbeitsraum nutzbar, demnächst zu bestimmten Zeiten auch betreut (siehe https://udue.de/ludi).
- Durch die Übungen können bis zu 10% Bonus für die Klausur erreicht werden. Voraussetzung ist nicht nur Einreichen von Lösungen, sondern auch Präsentation in der Übungsgruppe. Näheres dazu in der ersten Übung.
- Die Aufgaben werden auf Deutsch und Englisch gestellt.

Hinweise zu den Übungen

- In der ersten Übung (16.–19.10.2017) werden vor allem organisatorische Dinge besprochen und geregelt.
  Gehen Sie hin!

Hinweise zu den Übungen

- In der ersten Übung (16.–19.10.2017) werden vor allem organisatorische Dinge besprochen und geregelt.
  Gehen Sie hin!
- Abgaben erfolgen in Dreiergruppen, außer für das erste Übungsblatt.

Hinweise zu den Übungen

- In der ersten Übung (16.–19.10.2017) werden vor allem organisatorische Dinge besprochen und geregelt.
  Gehen Sie hin!
- Abgaben erfolgen in Dreiergruppen, außer für das erste Übungsblatt.
- Die Dreiergruppen werden während des ersten Übungstermins geformt, sie erstrecken sich nicht über Übungsgruppengrenzen hinweg.

Hinweise zu den Übungen

- In der ersten Übung (16.–19.10.2017) werden vor allem organisatorische Dinge besprochen und geregelt.
  Gehen Sie hin!
- Abgaben erfolgen in Dreiergruppen, außer für das erste Übungsblatt.
- Die Dreiergruppen werden während des ersten Übungstermins geformt, sie erstrecken sich nicht über Übungsgruppengrenzen hinweg.
- Details zu Abgabeterminen etc. erhalten Sie noch.

Hinweise zu den Übungen

- In der ersten Übung (16.–19.10.2017) werden vor allem organisatorische Dinge besprochen und geregelt.
  Gehen Sie hin!
- Abgaben erfolgen in Dreiergruppen, außer für das erste Übungsblatt.
- Die Dreiergruppen werden während des ersten Übungstermins geformt, sie erstrecken sich nicht über Übungsgruppengrenzen hinweg.
- Details zu Abgabeterminen etc. erhalten Sie noch.
- Plagiate oder das Kopieren alter Musterlösungen sind selbstverständlich nicht erlaubt!
  In solchen Fällen vergeben wir keine Punkte.

Klausur

Die Lehrveranstaltung wird durch eine 90-minütige schriftliche Klausur am Ende des Semesters geprüft.

Der derzeitige Planungsstand für den Klausurtermin ist Montag, 5. März, 10 Uhr.

Die Anmeldung erfolgt über das Prüfungsamt.

Klausur

Die Lehrveranstaltung wird durch eine 90-minütige schriftliche Klausur am Ende des Semesters geprüft.

Der derzeitige Planungsstand für den Klausurtermin ist Montag, 5. März, 10 Uhr.

Die Anmeldung erfolgt über das Prüfungsamt.

Auch die Klausur wird in Deutsch und in Englisch angeboten.

Literatur

Wolfgang Reisig.
Petrinetze –
Modellierungstechnik,
Analysemethoden,
Fallstudien.
Springer, 2010

https://dx.doi.org/10.1007/978-3-8348-9708-4
(elektronische Version über den Uni-Account)

Literatur

Lutz Priese, Harro Wimmel.
Petri-Netze.
Springer, 2008

https://dx.doi.org/10.1007/978-3-540-76971-2
(elektronische Version über den Uni-Account)

Literatur

Tadao Murata.
Petri Nets: Properties, Analysis and Applications.
Proc. of the IEEE, 77(4), pages 541–580, 1989

Petri Nets: Properties, Analysis and Applications

TADAO MURATA, FELLOW, IEEE

Invited Paper

This is an invited tutorial-review paper on Petri nets—a graphical and mathematical modeling tool. Petri nets are a promising tool for describing and studying information processing systems that are characterized as being concurrent, asynchronous, distributed, parallel, nondeterministic, and/or stochastic.

The paper starts with a brief review of the history and the application areas considered in the literature. It then proceeds with introductory modeling examples, behavioral and structural properties, three methods of analysis, subclasses of Petri nets and their analysis. In particular, one section is devoted to marked graphs—the concurrent system model most amenable to analysis. In addition, the paper presents introductory discussions on stochastic nets with their application to performance modeling, and on high-level nets with their application to logic programming. Also included are recent results on reachability criteria. Suggestions are provided for further reading on many subject areas of Petri nets.

I. INTRODUCTION

Petri nets are a graphical and mathematical modeling tool applicable to many systems. They are a promising tool for describing and studying information processing systems that are characterized as being concurrent, asynchronous, distributed, parallel, nondeterministic, and/or stochastic. As a graphical tool, Petri nets can be used as a visual communication aid similar to flow charts, block diagrams, and networks. In addition, tokens are used in these nets to simulate the dynamic and concurrent activities of systems. As a mathematical tool, it is possible to set up state equations, algebraic equations, and other mathematical models governing the behavior of systems. Petri nets can be used by both practitioners and theoreticians. Thus, they provide a powerful medium of communication between these practitioners can learn from theoreticians how to make their models more methodical, and theoreticians can learn from practitioners how to make their models more realistic.

Historically speaking, the concept of the Petri net has its origin in Carl Adam Petri's dissertation [1], submitted in 1962 to the faculty of Mathematics and Physics at the Technical University of Darmstadt, West Germany. The dissertation was prepared while C. A. Petri worked as a scientist at the University of Bonn. Petri's work [1], [2] came to the attention of A. W. Holt, who later led the Information System Theory Project of Applied Data Research, Inc., in the United States. The early developments and applications of Petri nets (or their predecessors) are found in the reports [3]–[8] associated with this project, and in the Record [9] of the 1970 Project MAC Conference on Concurrent Systems and Parallel Computation. From 1970 to 1975, the Computation Structure Group at MIT was most active in conducting Petri-net related research, and produced many reports and theses on Petri nets. In July 1975, there was a conference on Petri Nets and Related Methods at MIT, but no conference proceedings were published. Most of the Petri-net related papers written in English before 1980 are listed in the annotated bibliography of the first book [10] on Petri nets. More recent papers up until 1984 and those works done in Germany and other European countries are annotated in the appendix of another book [11]. Three tutorial articles [12]–[14] provide a complimental, easy-to-read introduction to Petri nets.

Since the late 1970's, the Europeans have been very active in organizing workshops and publishing conference proceedings on Petri nets. In October 1979, about 135 researchers mostly from European countries assembled in Hamburg, West Germany, for a two-week advanced course on General Net Theory of Processes and Systems. The 17 lectures given in this course were published in its proceedings [15], which is currently out of print. The second advanced course was held in Bad Honnef, West Germany, in September 1980. The proceedings [16], [17] of this course contain 34 articles, including two recent articles by C. A. Petri; one [18] is concerned with his axioms of concurrency theory and the other [19] with his suggestions for further research. The first European Workshop on Applications and Theory of Petri Nets was held in 1980 at Strasbourg, France. Since then, this series of workshops has been held every year at different locations in Europe: 1981, Bad Honnef, West Germany; 1982, Varenna, Italy; 1983, Toulouse, France; 1984, Aarhus, Denmark; 1985, Espoo, Finland; 1986, Oxford, Great

Manuscript received May 28, 1988; revised November 4, 1988. This work was supported by the National Science Foundation under Grant DMC-8518088.

The author is with the Department of Electrical Engineering and Computer Science, University of Illinois, Chicago, IL 60680, USA.

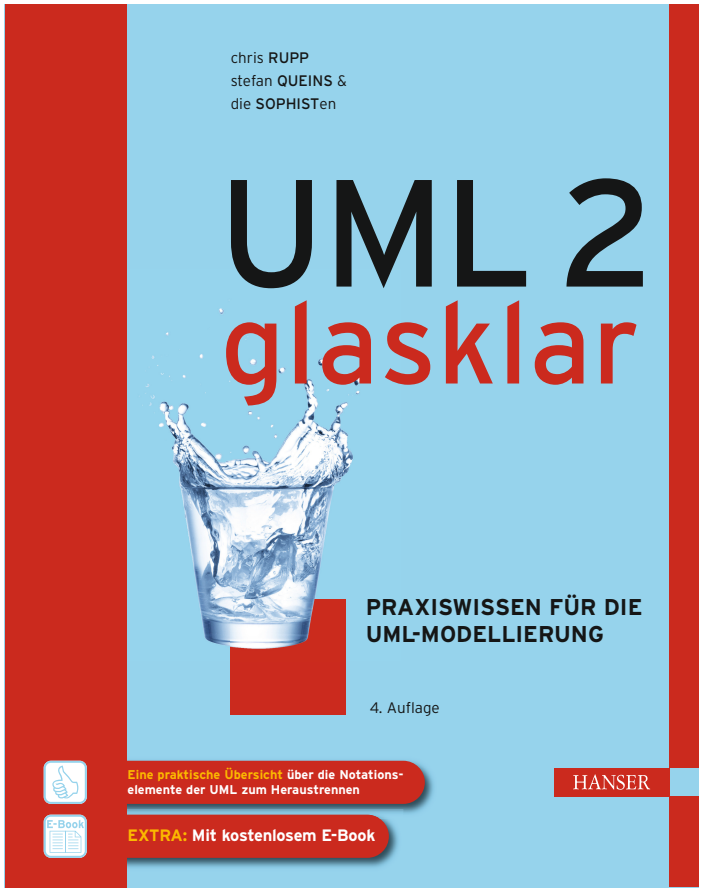
IEEE Log Number 8826788.

0018-9219/89/0400-0541$01.00 © 1989 IEEE

https://dx.doi.org/10.1109/5.24143
(elektronische Version über den Uni-Account)

Literatur

Chris Rupp, Stefan Queins.
UML 2 glasklar.
Hanser Fachbuch, 2012



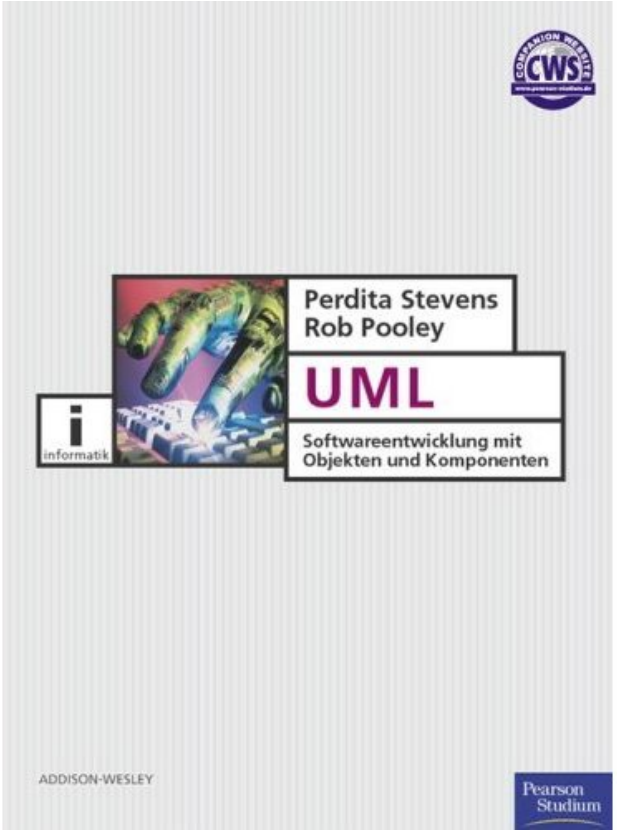
Literatur

Russ Miles, Kim Hamilton.
Learning UML 2.0.
O'Reilly, 2008

Urheberrechtlich geschütztes Material

Literatur

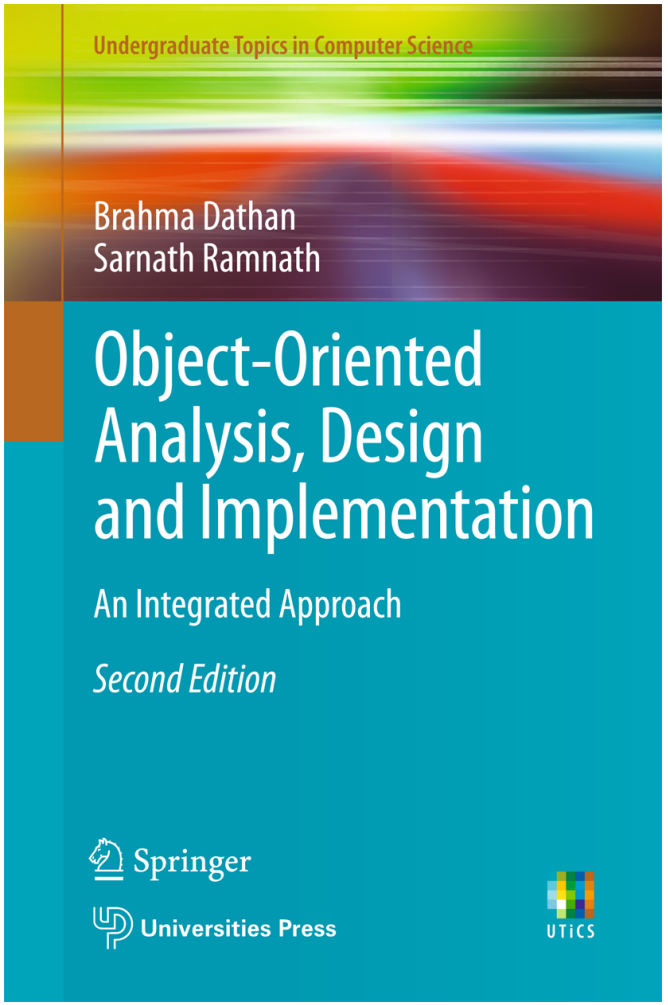
Perdita Stevens, Rob Pooley.
UML – Softwareentwicklung mit Objekten und Komponenten.
Pearson, 2001



Buch (vor allem das englische Original) ist in der Bibliothek verfügbar.

Literatur

Brahma Dathan,
Sarnath Ramnath.
Object-Oriented Analysis, Design and Implementation – An Integrated Approach.
Springer, 2015



Literatur

David Harel.
Statecharts: A visual formalism for complex systems.
Science of Computer Programming, 8, pages 231–274, 1987

Science of Computer Programming 8 (1987) 231-274
North-Holland

STATECHARTS: A VISUAL FORMALISM FOR COMPLEX SYSTEMS*

David HAREL
Department of Applied Mathematics, The Weizmann Institute of Science, Rehovot, Israel

Communicated by A. Pnueli
Received December 1984
Revised July 1986

Abstract. We present a broad extension of the conventional formalism of state machines and state diagrams, that is relevant to the specification and design of complex discrete-event systems, such as multi-computer real-time systems, communication protocols and digital control units. Our diagrams, which we call *statecharts*, extend conventional state-transition diagrams with essentially three elements, dealing, respectively, with the notions of hierarchy, concurrency and communication. These transform the language of state diagrams into a highly structured and economical description language. Statecharts are thus compact and expressive—small diagrams can express complex behavior—as well as compositional and modular. When coupled with the capabilities of computerized graphics, statecharts enable viewing the description at different levels of detail, and make even very large specifications manageable and comprehensible. In fact, we intend to demonstrate here that statecharts counter many of the objections raised against conventional state diagrams, and thus appear to render specification by diagrams an attractive and plausible approach. Statecharts can be used either as a stand-alone behavioral description or as part of a more general design methodology that deals also with the system's other aspects, such as functional decomposition and data-flow specification. We also discuss some practical experience that was gained over the last three years in applying the statechart formalism to the specification of a particularly complex system.

1. Introduction

The literature on software and systems engineering is almost unanimous in recognizing the existence of a major problem in the specification and design of large and complex *reactive systems*. A reactive system (see [14]), in contrast with a *transformational system*, is characterized by being, to a large extent, event-driven, continuously having to react to external and internal stimuli. Examples include telephones, automobiles, communication networks, computer operating systems, missile and avionics systems, and the man-machine interface of many kinds of ordinary software. The problem is rooted in the difficulty of describing reactive behavior in ways that are clear and realistic, and at the same time formal and

* The initial part of this research was carried out while the author was consulting for the Research and Development Division of the Israel Aircraft Industries (IAI), Lod, Israel. Later stages were supported in part by grants from IAI and AD CAD, Ltd.

0167-6423/87/$3.50 © 1987, Elsevier Science Publishers B.V. (North-Holland)

https://dx.doi.org/10.1016/0167-6423(87)90035-9

Literatur

Hinweise:

- Die Literatur ist als Ergänzung gedacht, sie präsentiert den Stoff oft aus einem anderen Blickwinkel.
- Sehen Sie sich die Bücher erst an, bevor Sie etwas kaufen. Nicht jede/r kommt mit jedem Buch zurecht.
- Von einigen der Bücher können Sie über Ihren Uni-Account eine elektronische Version kostenlos erhalten.
- Weitere relevante Bücher sind in der Bibliothek (Gebäude LK) verfügbar.
- Wagen Sie sich ruhig auch an englische Literatur, denn englische Fachsprache ist zumeist gut verständlich.

Einführung in die Modellierung

Was ist Modellierung?

Modell

Ein Modell ist eine Repräsentation eines Systems von Objekten, Beziehungen und/oder Abläufen. Ein Modell vereinfacht und abstrahiert dabei im Allgemeinen das repräsentierte System.

Was ist Modellierung?

Modell

Ein Modell ist eine Repräsentation eines Systems von Objekten, Beziehungen und/oder Abläufen. Ein Modell vereinfacht und abstrahiert dabei im Allgemeinen das repräsentierte System.

System

Der Begriff System wird hier sehr allgemein verwendet. Er kann

- einen Teil der Realität oder ein (noch) nicht bestehendes Gebilde;

Was ist Modellierung?

Modell

Ein Modell ist eine Repräsentation eines Systems von Objekten, Beziehungen und/oder Abläufen. Ein Modell vereinfacht und abstrahiert dabei im Allgemeinen das repräsentierte System.

System

Der Begriff System wird hier sehr allgemein verwendet. Er kann

- einen Teil der Realität oder ein (noch) nicht bestehendes Gebilde;
- etwas Gegenständliches oder Virtuelles

bezeichnen.

Was ist Modellierung?

Modellierung

Modellierung ist der Prozess, bei dem ein Modell eines Systems erstellt wird.

Was ist Modellierung?

Modellierung

Modellierung ist der Prozess, bei dem ein Modell eines Systems erstellt wird.

Warum sollte man modellieren?

Was ist Modellierung?

Modellierung

Modellierung ist der Prozess, bei dem ein Modell eines Systems erstellt wird.

Warum sollte man modellieren?

$\rightsquigarrow$ Um ein System zu entwerfen, besser zu verstehen, zu visualisieren, zu simulieren, ...

Was ist Modellierung?

Modellierung

Modellierung ist der Prozess, bei dem ein Modell eines Systems erstellt wird.

Warum sollte man modellieren?

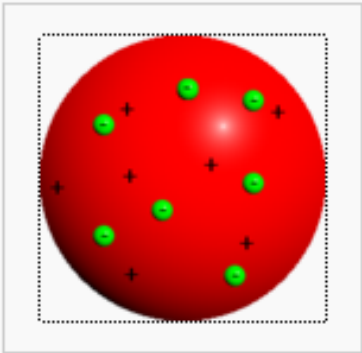
$\rightsquigarrow$ Um ein System zu entwerfen, besser zu verstehen, zu visualisieren, zu simulieren, ...

Um etwas konkreter zu werden, betrachten wir Beispiele in verschiedenen Disziplinen (Physik, Biologie, Klimaforschung).

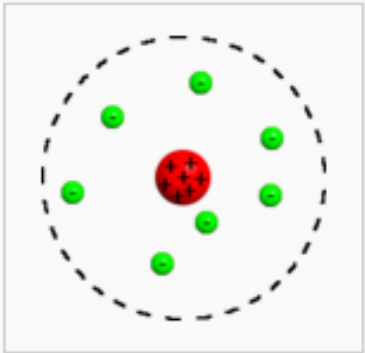
Modellierung in der Physik

Atommodelle

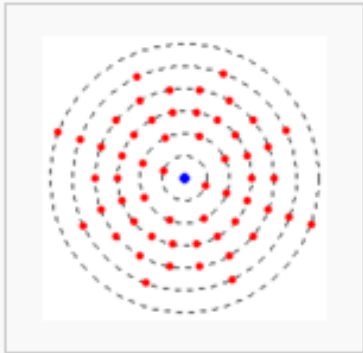
Atome bestehen aus Protonen, Neutronen und Elektronen. Wie diese Teilchen zusammenwirken, wird in Atommodellen beschrieben, die im Laufe der Zeit immer wieder verändert wurden.



Thomsonsches Atommodell



Rutherfordsches Atommodell



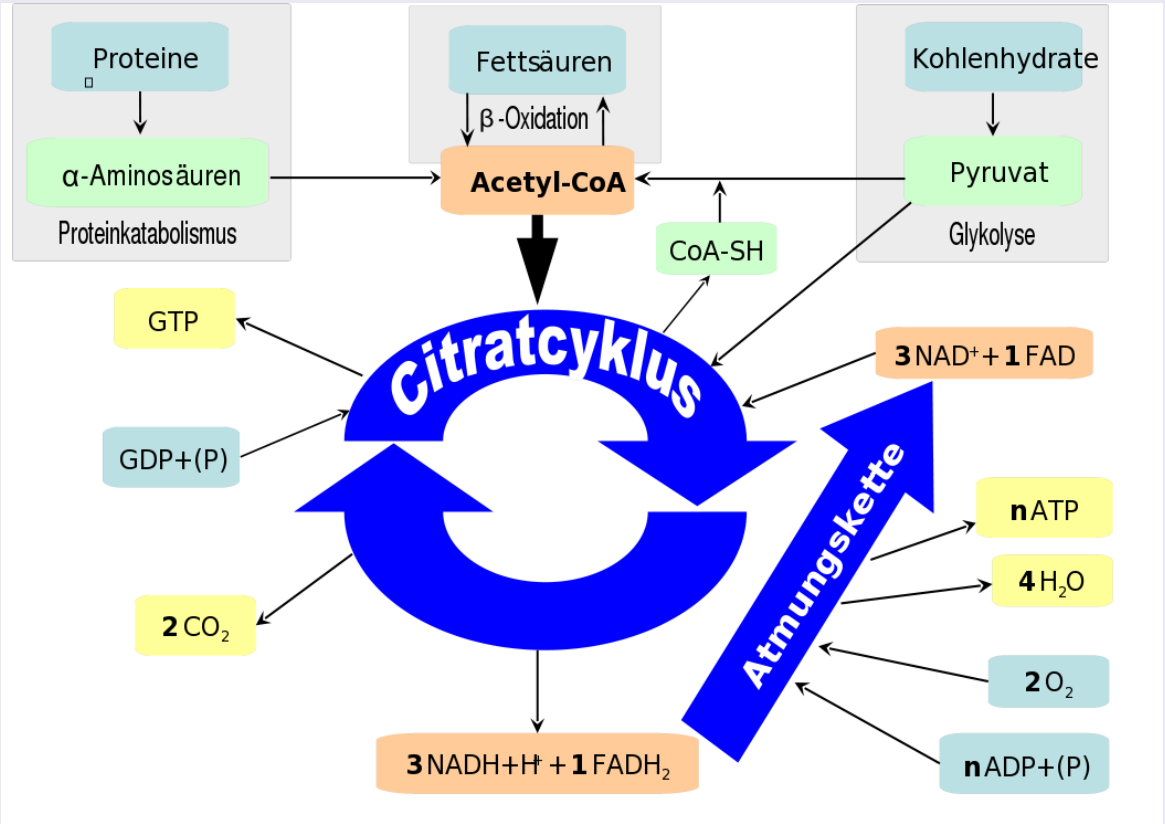
Bohrsches Atommodell

Orbitalmodell

Modellierung in der Biologie

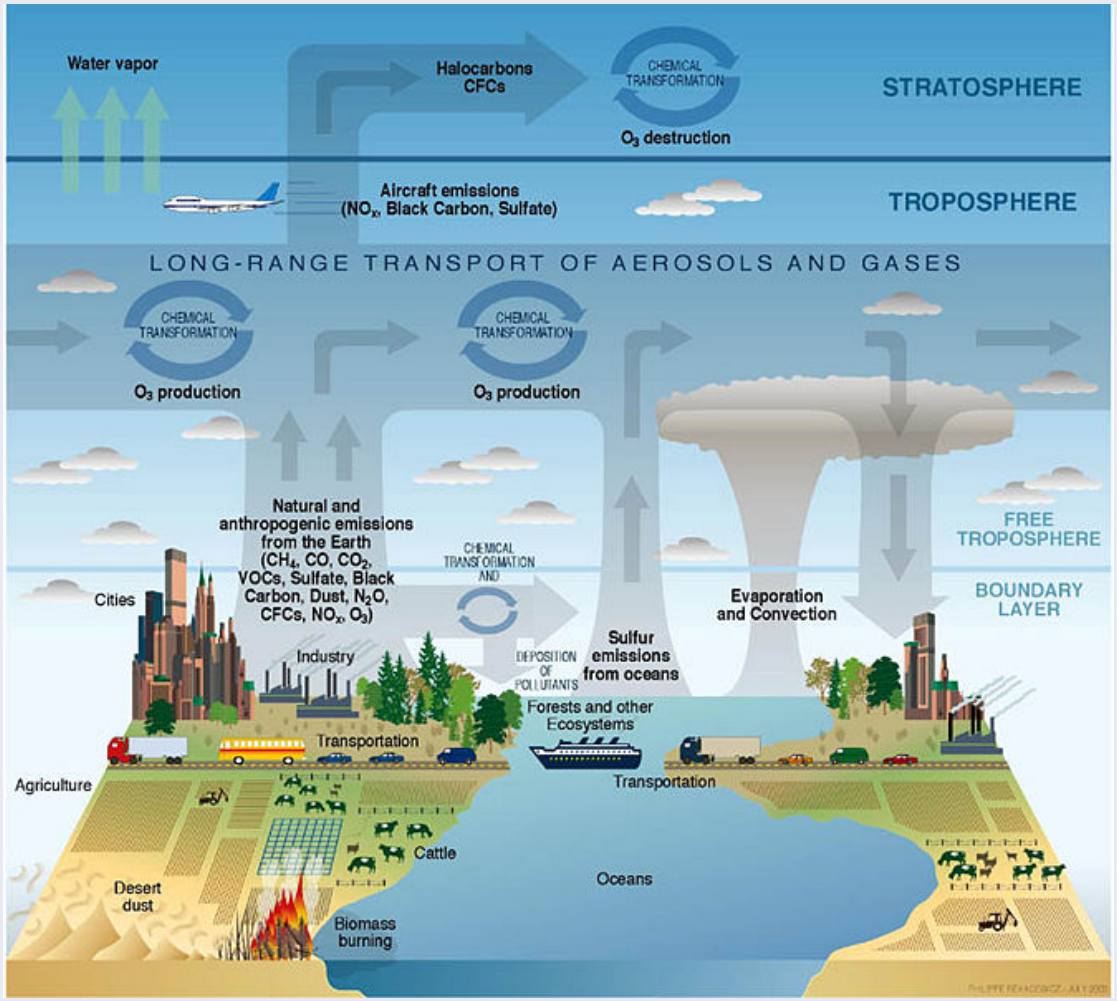
Zitronensäurezyklus

Der Zitronensäurezyklus oder Citratzyklus modelliert den Abbau organischer Stoffe im Körper.



Modellierung in der Klimaforschung

Modell des Transports von Gasen in der Atmosphäre



Arten von Modellen

visuell vs. textuell

Nicht alle Modelle sind visuell bzw. grafisch. Auch mit textuellen Beschreibungen und Formeln kann man modellieren (siehe beispielsweise mathematische Modelle).

Dennoch werden häufig grafische Darstellungen benutzt, auch aus didaktischen Gründen und um sich besser über die Modelle verständigen zu können.

Arten von Modellen

qualitativ vs. quantitativ

- qualitative Modelle: Welche Objekte gibt es? Was passiert? Warum passiert es? In welcher Reihenfolge geschehen die Ereignisse? Was sind die kausalen Zusammenhänge? Welche Phänomene treten auf?
- quantitative Modelle: Wieviele Objekte gibt es? Wie lange dauert ein Vorgang? Wie wahrscheinlich ist ein bestimmtes Ereignis?

Arten von Modellen

black box vs. white box

- black box: Nur das von außen beobachtbare Verhalten wird beschrieben.
- white box: Es wird auch beschrieben, wie das von außen beobachtbare Verhalten im „Inneren“ des Systems erzeugt wird.

Arten von Modellen

statisch vs. dynamisch

- Ein statisches Modell beschreibt einen Zustand des Systems zu einem bestimmten Zeitpunkt.
- Ein dynamisches Modell beschreibt hingegen auch, wie das System sich entwickelt (ein oder mehrere mögliche Abläufe oder sogar das gesamte Systemverhalten).

Arten von Modellen

formal vs. semi-formal vs. nicht-formal

Je nach Exaktheit der Modelle erhält man:

- formale Modelle, die vollkommen exakt in ihren Aussagen sind (vor allem mathematische Modelle)
- semi-formale Modelle, die teilweise exakt sind, jedoch nicht alles vollständig spezifieren
- nicht-formale Modelle, die als grobe Richtlinie dienen können, jedoch eher vage Aussagen machen

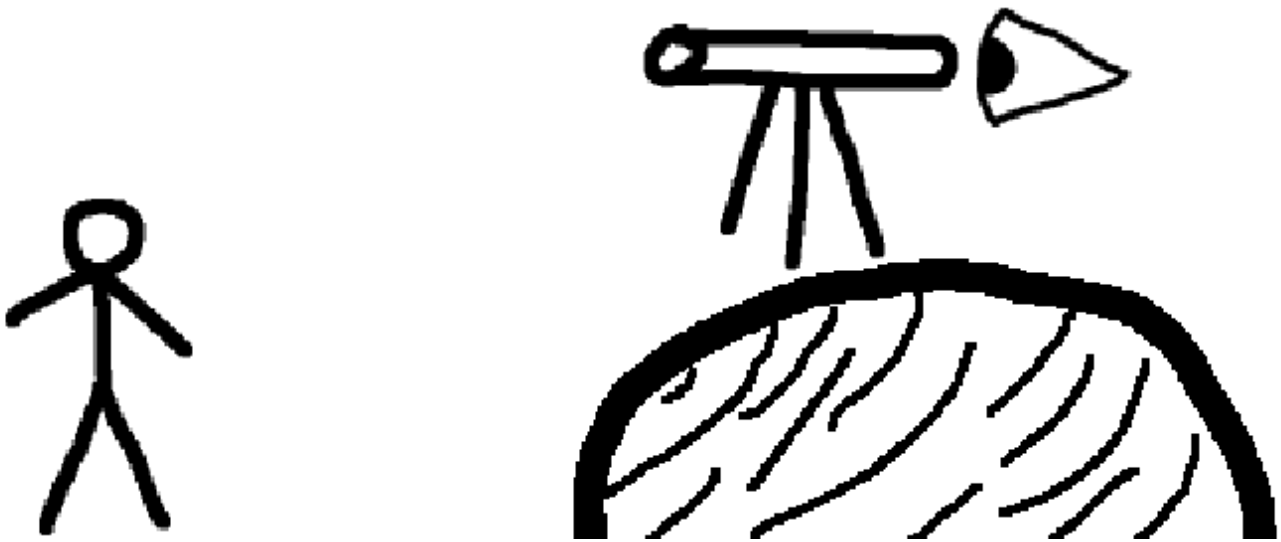
Probleme mit nicht-formalen Modellen

Natürliche Sprache ist nicht immer eindeutig.

Beispiel:

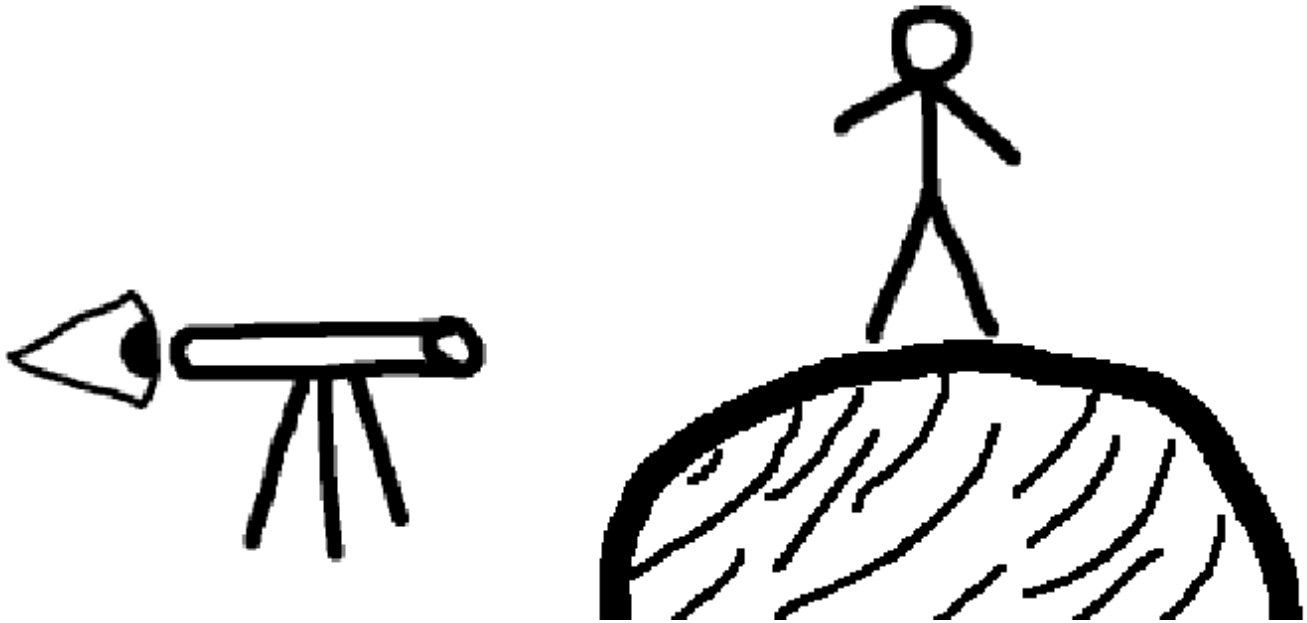
> Ich sah den Mann auf dem Berg mit dem Fernrohr.

Probleme mit nicht-formalen Modellen



(((Ich sah den Mann) auf dem Berg) mit dem Fernrohr)

Probleme mit nicht-formalen Modellen

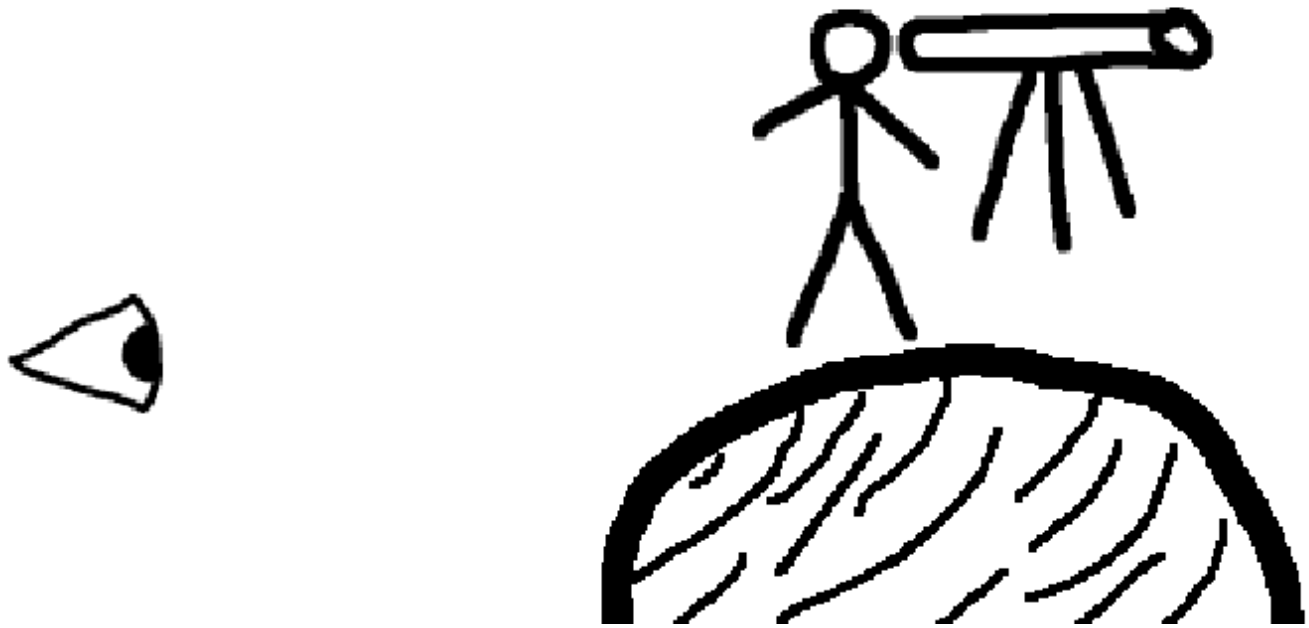


((Ich sah (den Mann auf dem Berg)) mit dem Fernrohr)

Probleme mit nicht-formalen Modellen

((Ich sah den Mann) (auf dem Berg mit dem Fernrohr))

Probleme mit nicht-formalen Modellen



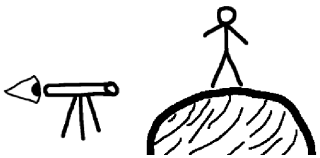
(Ich sah ((den Mann auf dem Berg) mit dem Fernrohr))

Probleme mit nicht-formalen Modellen

(Ich sah (den Mann (auf dem Berg mit dem Fernrohr)))

Probleme mit nicht-formalen Modellen

(((Ich sah den Mann) auf dem Berg) mit dem Fernrohr)



((Ich sah (den Mann auf dem Berg)) mit dem Fernrohr)

((Ich sah den Mann) (auf dem Berg mit dem Fernrohr))

(Ich sah ((den Mann auf dem Berg) mit dem Fernrohr))

(Ich sah (den Mann (auf dem Berg mit dem Fernrohr)))

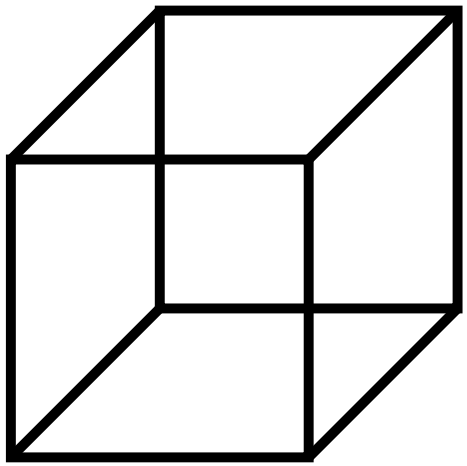
5 mögliche Interpretationen!

Probleme mit nicht-formalen Modellen

Auch grafische Darstellungen können uneindeutig sein:

Probleme mit nicht-formalen Modellen

Auch grafische Darstellungen können uneindeutig sein:



Modellierung in der Informatik

In dieser Lehrveranstaltung geht es um Modellierungsmethoden in der Informatik.

Diese werden zum Entwurf verschiedener Systeme eingesetzt:

- (Objekt-orientierte) Programme
- (Große) Software-Systeme
- Benutzungsoberflächen
- Datenbanken
- Virtual Reality, Computer-Spiele
- ...

Wozu ist Modellierung gut?

Wozu benötigt man in der Informatik Modelle?

Wozu ist Modellierung gut?

Wozu benötigt man in der Informatik Modelle?

Je komplexer ein Informatik-System sein wird, desto wichtiger ist es, einen Plan zu haben, bevor man beginnt, es zu konstruieren.

Wozu ist Modellierung gut?

Wozu benötigt man in der Informatik Modelle?

Je komplexer ein Informatik-System sein wird, desto wichtiger ist es, einen Plan zu haben, bevor man beginnt, es zu konstruieren.

Dies führt idealerweise zu:

- Vermeidung von Fehlern
- besserer Qualität
- niedrigeren Kosten
- besserer Dokumentation und Wiederverwendbarkeit

Wozu ist Modellierung gut?

Analogie: Bau eines Hauses

Wozu ist Modellierung gut?

Analogie: Bau eines Hauses

Beim Bau einer Hundehütte kann man zumeist ohne große Planung vorgehen. Die Hütte kann von einer einzelnen Person erstellt werden und ein Hund hat zumeist keine großen Anforderungen.

Wozu ist Modellierung gut?

Analogie: Bau eines Hauses

Beim Bau eines Einfamilienhauses ist Planung viel wichtiger. Die Familie ist anspruchsvoller als ein Hund, Bauvorschriften müssen eingehalten werden und vermutlich werden nicht alle Arbeiten von derselben Person durchgeführt.

Wozu ist Modellierung gut?

Analogie: Bau eines Hauses

Der Bau eines Hochhauses ist ohne vorherige Erstellung eines detaillierten Plans bzw. Modells nicht möglich. Das Risiko, Fehler zu machen, ist sehr groß und Fehler können extrem kostspielig werden.

Wozu ist Modellierung gut?

Schwierigkeiten beim Entwurf komplexer Systeme

- Menschen können sich komplexe Systeme normalerweise nicht in vollem Umfang vorstellen.
- Bei mehreren Beteiligten gibt es oft unterschiedliche Meinungen darüber, wie das System aussehen muss.
  $\rightsquigarrow$ Modelle dienen zur Kommunikation!

Wozu ist Modellierung gut?

Schwierigkeiten beim Entwurf komplexer Systeme

- Menschen können sich komplexe Systeme normalerweise nicht in vollem Umfang vorstellen.
- Bei mehreren Beteiligten gibt es oft unterschiedliche Meinungen darüber, wie das System aussehen muss.
  $\rightsquigarrow$ Modelle dienen zur Kommunikation!

Modellierung ist in der Informatik weniger verbreitet als in den Ingenieurwissenschaften, aber ebenso relevant.

Wozu ist Modellierung gut?

Schwierigkeiten beim Entwurf komplexer Systeme

- Menschen können sich komplexe Systeme normalerweise nicht in vollem Umfang vorstellen.
- Bei mehreren Beteiligten gibt es oft unterschiedliche Meinungen darüber, wie das System aussehen muss.
  $\rightsquigarrow$ Modelle dienen zur Kommunikation!

Modellierung ist in der Informatik weniger verbreitet als in den Ingenieurwissenschaften, aber ebenso relevant.

Besonderheiten von Software beeinflussen Rolle von Modellen:

- Immaterialität, daher gar nicht so leicht, etwa festzustellen, „wie viel“ eines Modells schon umgesetzt wurde.

Wozu ist Modellierung gut?

Schwierigkeiten beim Entwurf komplexer Systeme

- Menschen können sich komplexe Systeme normalerweise nicht in vollem Umfang vorstellen.
- Bei mehreren Beteiligten gibt es oft unterschiedliche Meinungen darüber, wie das System aussehen muss.
  $\rightsquigarrow$ Modelle dienen zur Kommunikation!

Modellierung ist in der Informatik weniger verbreitet als in den Ingenieurwissenschaften, aber ebenso relevant.

Besonderheiten von Software beeinflussen Rolle von Modellen:

- Immaterialität, daher gar nicht so leicht, etwa festzustellen, „wie viel“ eines Modells schon umgesetzt wurde.
- Einzigartigkeit jedes Softwareprojekts, daher Modelle nicht in der Rolle von Vorlagen zum Zweck mehrfacher Realisierung.

Probleme bei der Entwicklung großer Systeme

Feststellung (nach Glinz)

Die Entwicklung von Klein-Software unterscheidet sich fundamental von der Entwicklung größerer Software.

Klein-Groß-Gegensätze in der Software-Entwicklung:

klein	**groß**
Programme bis ungefähr 300 Zeilen	Längere Programme
Für den Eigengebrauch	Für den Gebrauch durch Dritte
Vage Zielsetzung genügt, das Produkt ist seine eigene Spezifikation	Genaue Zielbestimmung, das heißt explizite Anforderungsanalyse, erforderlich

Probleme bei der Entwicklung großer Systeme

klein	**groß**
Ein Schritt vom Problem zur Lösung genügt: Lösung wird direkt programmiert	Mehrere Schritte vom Problem zur Lösung erforderlich: Anforderungsanalyse, Spezifikation, Konzept, Entwurf und Programmieren der Teile, Zusammensetzen, Inbetriebnahme
Validierung (Überprüfung/Testen) und nötige Korrekturen finden am Endprodukt statt	Auf jeden Entwicklungsschritt muss ein Prüfschritt folgen, sonst kann Endergebnis unbrauchbar werden

Probleme bei der Entwicklung großer Systeme

klein	**groß**
Eine Person entwickelt: keine Kooperation und Kommunikation erforderlich	Mehrere Personen entwickeln gemeinsam: Koordination und Kommunikation notwendig
Komplexität des Problems in der Regel klein, Strukturieren und Behalten der Übersicht nicht schwierig	Komplexität des Problems größer bis sehr groß, explizite Maßnahmen zur Strukturierung und Modularisierung erforderlich

Probleme bei der Entwicklung großer Systeme

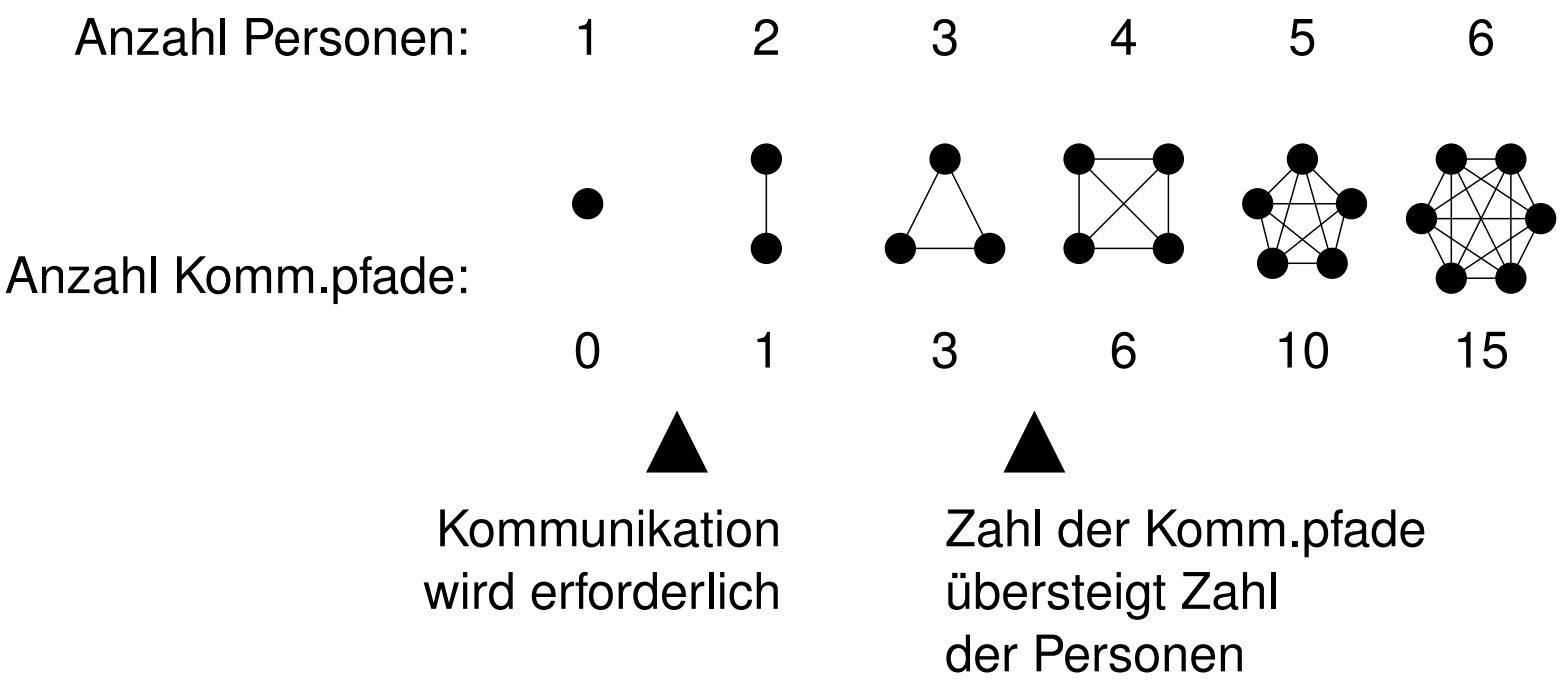
klein	**groß**
Software besteht aus wenigen Komponenten	Software besteht aus vielen Komponenten, die spezielle Maßnahmen zur Komponentenverwaltung erfordern
In der Regel wird keine Dokumentation erstellt	Dokumentation dringend erforderlich, damit Software wirtschaftlich betrieben und gepflegt werden kann

Probleme bei der Entwicklung großer Systeme

klein	**groß**
Keine Planung und Projektorganisation erforderlich	Planung und Projektorganisation zwingend erforderlich für eine zielgerichtete, wirtschaftliche Entwicklung

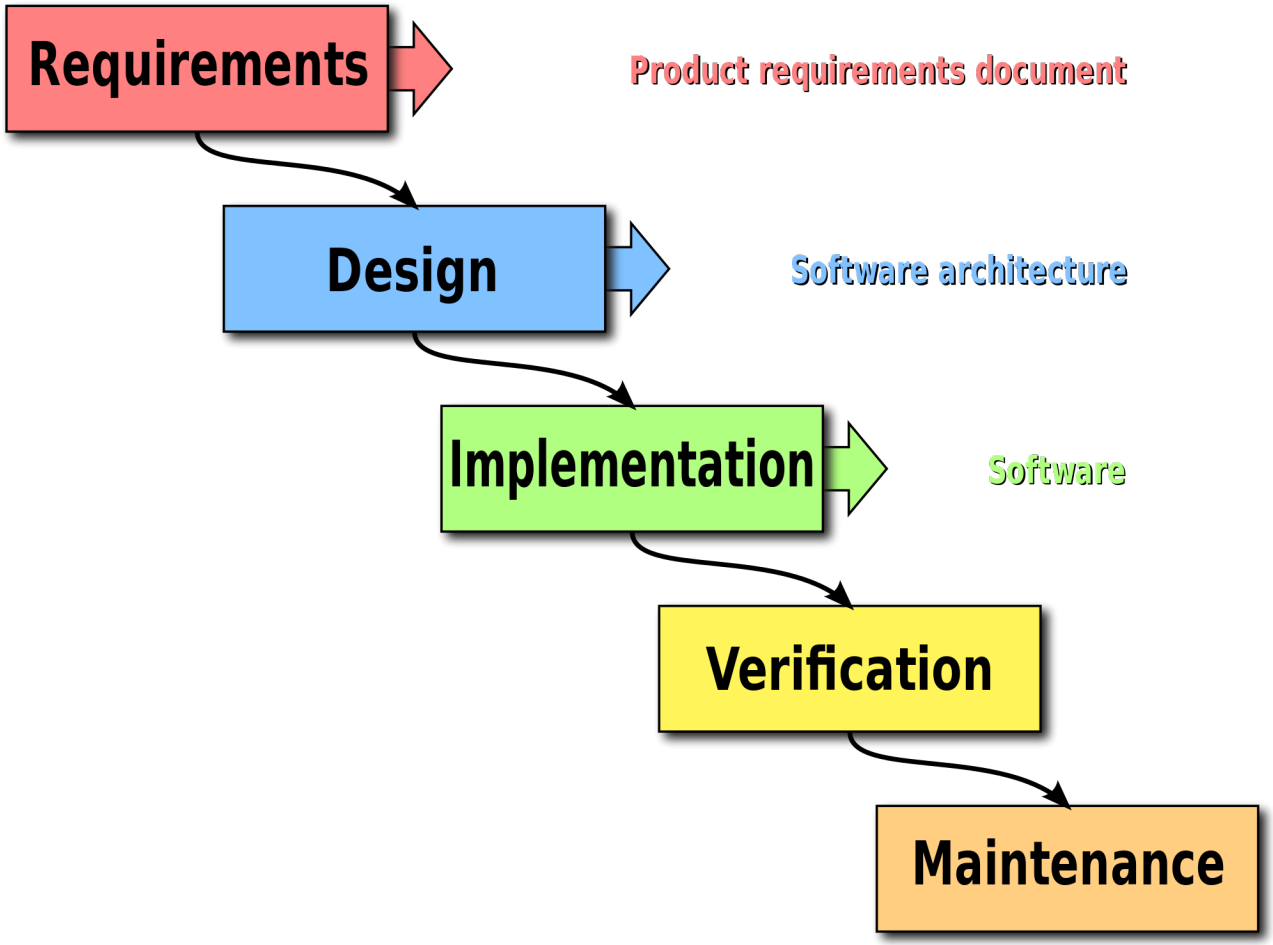
Probleme bei der Entwicklung großer Systeme

Starker Anstieg der Anzahl der Kommunikationspfade bei Anstieg der Entwicklerzahl:



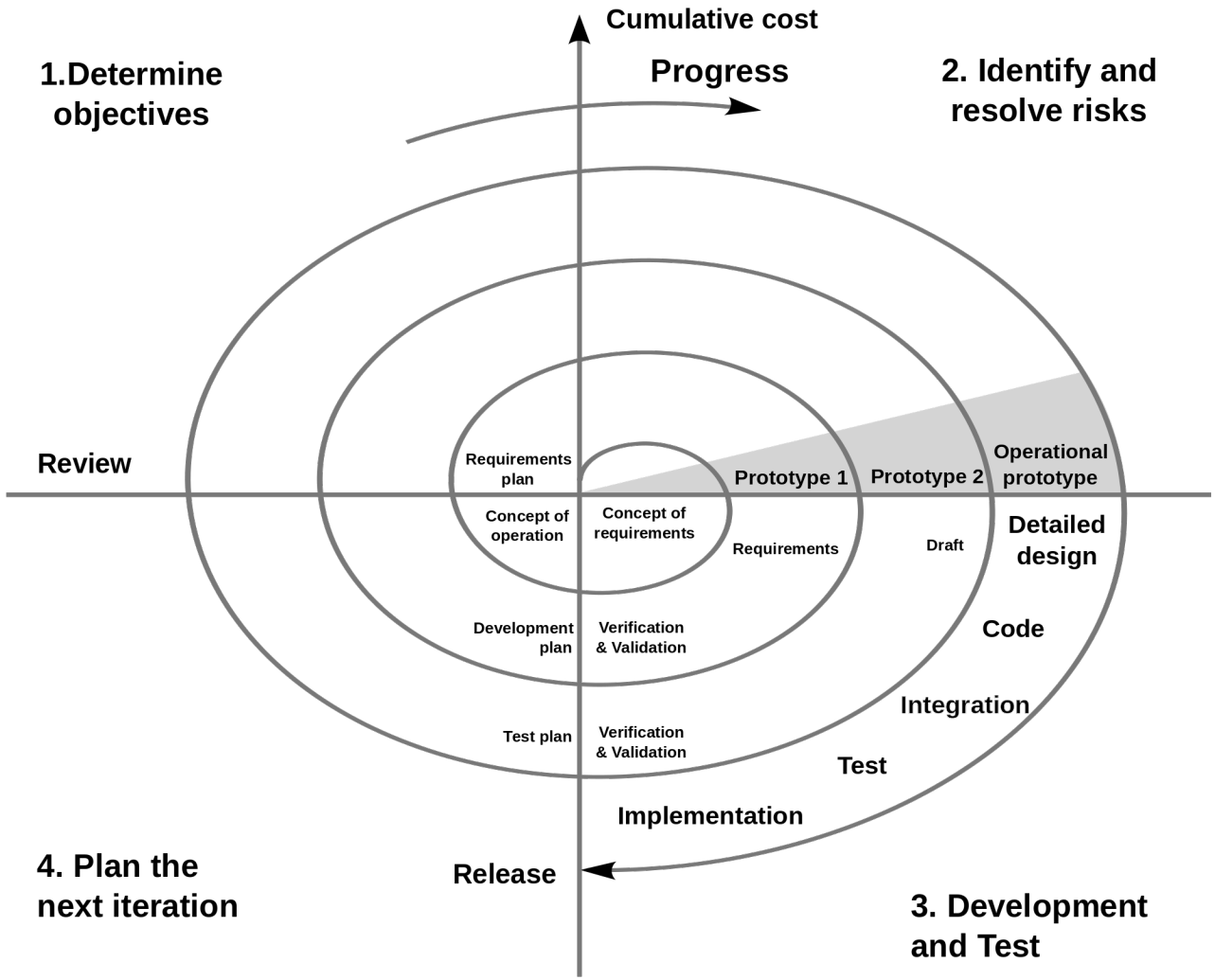
Wer modelliert, und wann?

Ein klassischer Softwareentwicklungs-Prozess, „Wasserfall“:



Wer modelliert, und wann?

Alternativer iterativer Ansatz, „Spiral“:



Wer modelliert, und wann?

„Sonderfälle“:

- Agile Software Development
- Open Source Development
- Model Driven Engineering
- . . .

Weitere Aspekte

Weitere wichtige Gesichtspunkte sind:

- Analyse:
  - Ist ein Modell korrekt? Ist es in sich konsistent?

Weitere Aspekte

Weitere wichtige Gesichtspunkte sind:

- Analyse:
  - Ist ein Modell korrekt? Ist es in sich konsistent?
  - Stimmt das Modell mit der späteren Implementierung überein?
    (Hier werden Verfahren zum Testen und zur Verifikation benötigt.)

Weitere Aspekte

Weitere wichtige Gesichtspunkte sind:

- Analyse:
  - Ist ein Modell korrekt? Ist es in sich konsistent?
  - Stimmt das Modell mit der späteren Implementierung überein?
    (Hier werden Verfahren zum Testen und zur Verifikation benötigt.)
- Werkzeuge, Software-Tools:
  ... werden benötigt zum Zeichnen, zum Darstellen (und Wechseln zwischen verschiedenen Darstellungen), zum Archivieren, zur Code-Generierung, zur Analyse, ...

Inhalt der Lehrveranstaltung

Inhalt

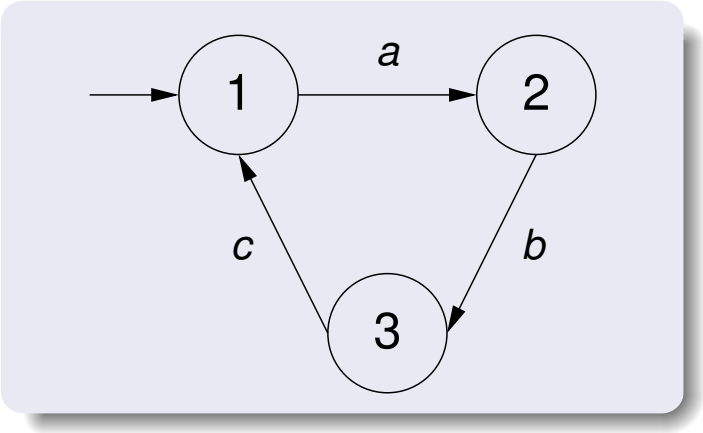
- Mathematische Grundlagen
- Graphen für statische und dynamische Systembeschreibungen
- Petrinetze
- UML (Unified Modeling Language)

Aus Gründen der Übersichtlichkeit und Didaktik werden wir uns hauptsächlich mit kleinen Modellen befassen.

Die Verwendung von Modellen zur Verifikation von Eigenschaften wird nicht im Zentrum stehen.

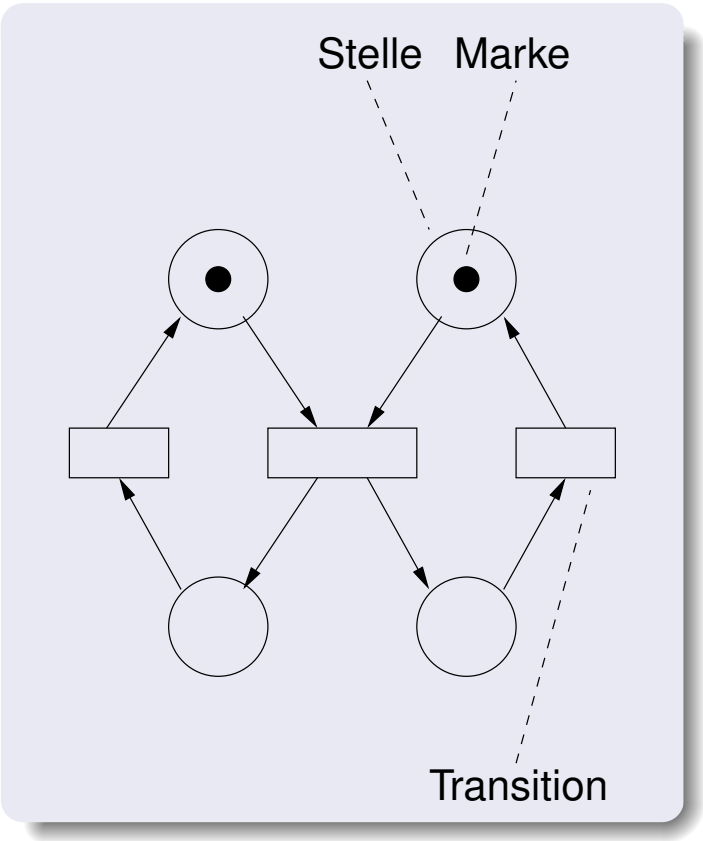
Graphen

- Graphen sind netzartige Strukturen, bestehend aus Knoten und Kanten.
- Sie können eingesetzt werden für
  - Statische Modellierung: Komponenten und Beziehungen zwischen den Komponenten
  - Dynamische Modellierung: Zustände und Übergänge zwischen den Zuständen



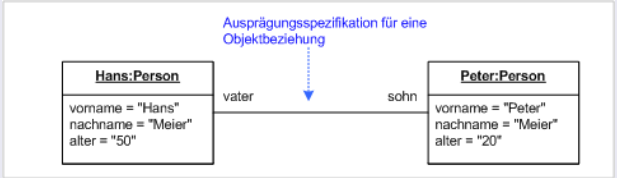
Petrinetze

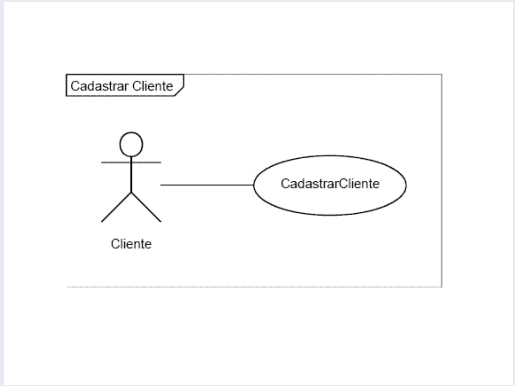
- Modellierung nebenläufiger und verteilter Systeme, zur Beschreibung der gemeinsamen Nutzung von Ressourcen.
- Schwerpunkt liegt auf der Modellierung des dynamischen Verhaltens.
- Etablierter Ansatz, der vielfältig eingesetzt wird.
- Besitzt formale Semantik.
- Entwickelt von Carl Adam Petri (1962).

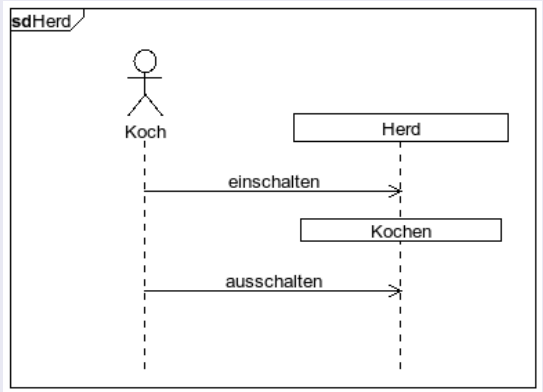


UML: Unified Modeling Language

- Standard-Modellierungssprache für Software Engineering.
- Basiert auf objekt-orientierten Konzepten.
- Sehr umfangreich, enthält viele verschiedene Typen von Modellen.
- Entwickelt von Grady Booch, James Rumbaugh, Ivar Jacobson (1997).







Inhalt der Lehrveranstaltung

Die vorgestellten Modellierungsmethoden sind nicht die einzigen Modellierungsmethoden in der Informatik.

Hier: Fokus auf visuelle Modellierung mit Hilfe von Diagrammen

Mögliche Alternative: algebraische Modellierungsmethoden, die sich stärker an der Mathematik orientieren

Beispiel: Wolf, Ziege, Kohlkopf

Wir modellieren folgendes System, um eine mögliche Lösung zu finden.

Wolf-Ziege-Kohlkopf-Problem

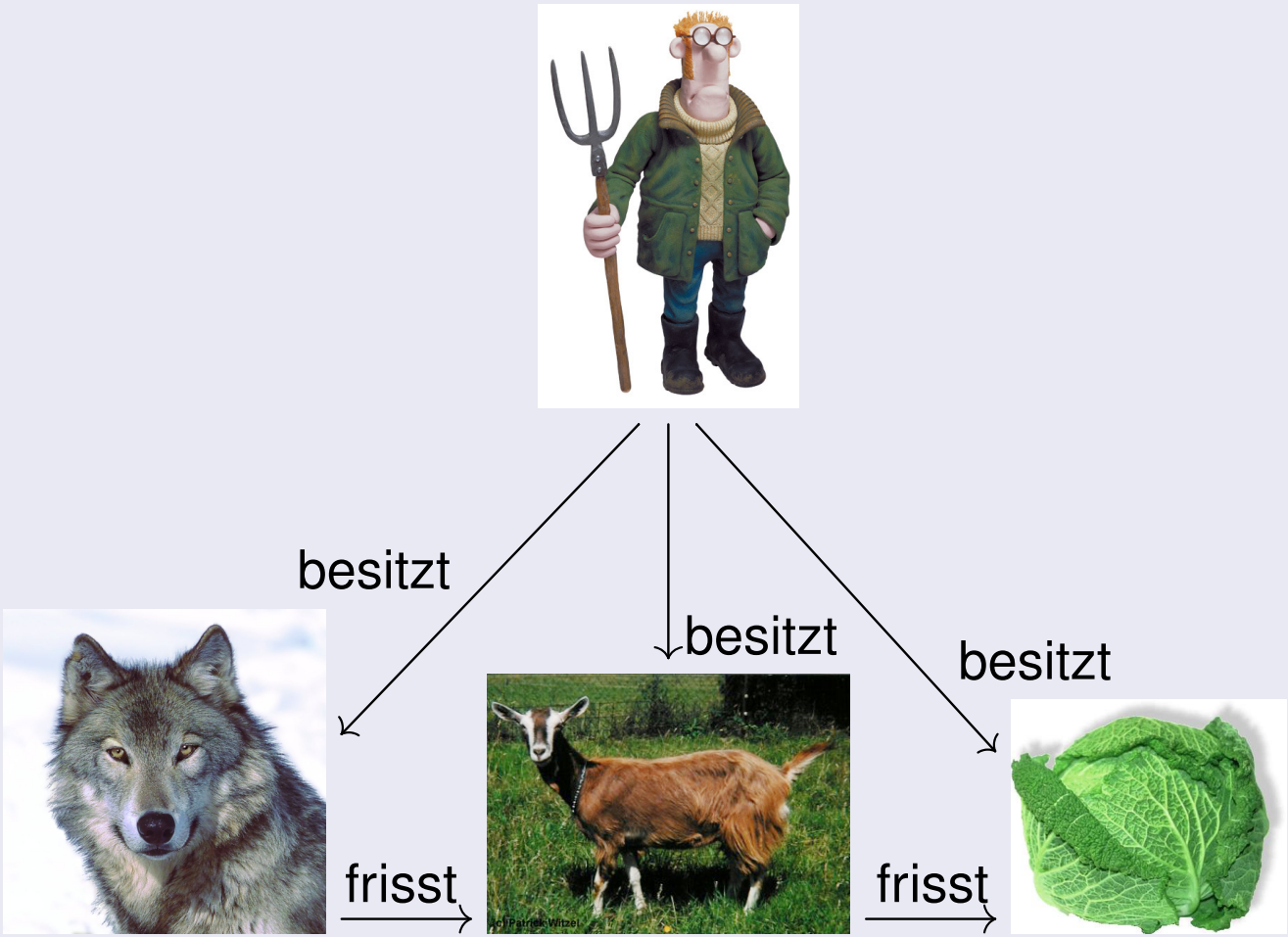
Ein Farmer will einen Fluss überqueren. Er hat einen Wolf, eine Ziege und einen Kohlkopf bei sich. Wenn sie allein gelassen werden, so frisst die Ziege den Kohlkopf und der Wolf die Ziege. Zur Überquerung des Flusses steht ein Boot mit zwei Plätzen zur Verfügung. Nur der Farmer kann rudern und er kann das Boot entweder allein benutzen oder ein Tier oder den Kohlkopf mitnehmen.

Beispiel: Wolf, Ziege, Kohlkopf

Statisches Modell I: Beteiligte Akteure/Objekte

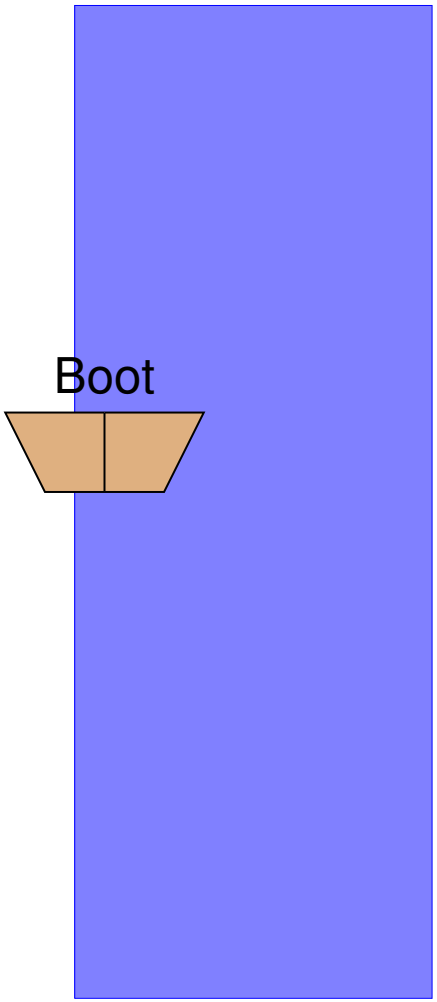
Beispiel: Wolf, Ziege, Kohlkopf

Statisches Modell II: Fress- und Eigentumsbeziehungen zwischen den Akteuren



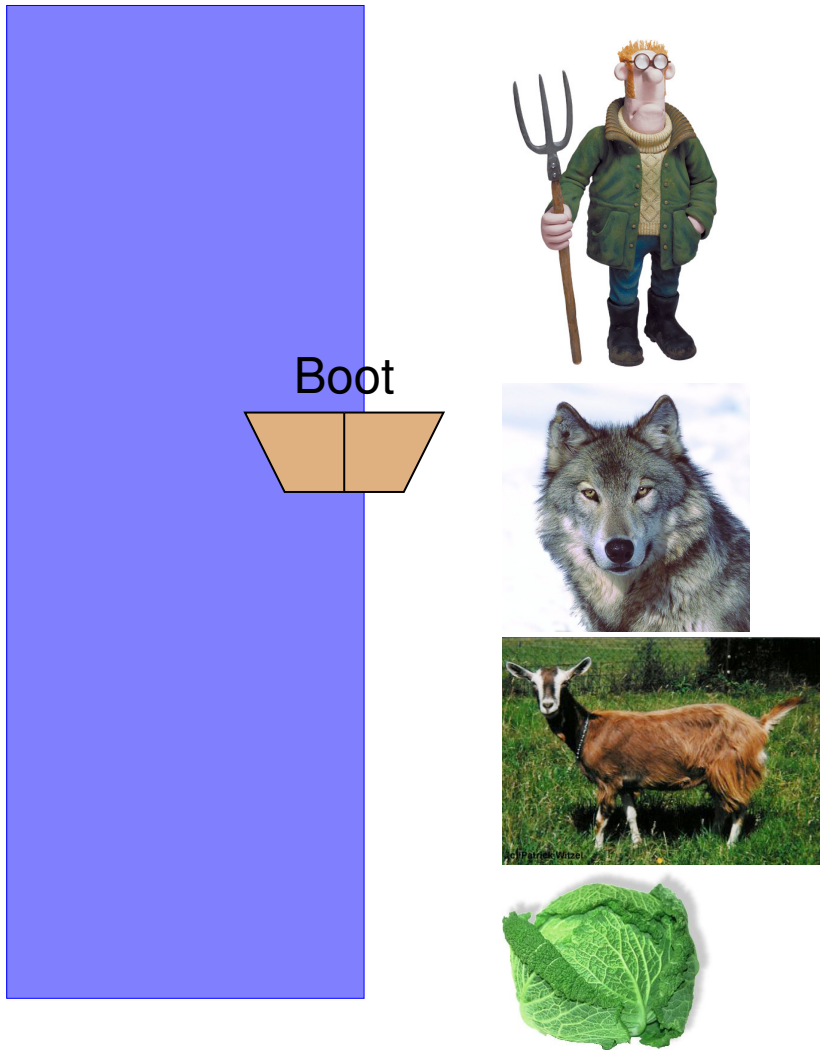
Beispiel: Wolf, Ziege, Kohlkopf

Ausgangssituation: vor Überquerung des Flusses



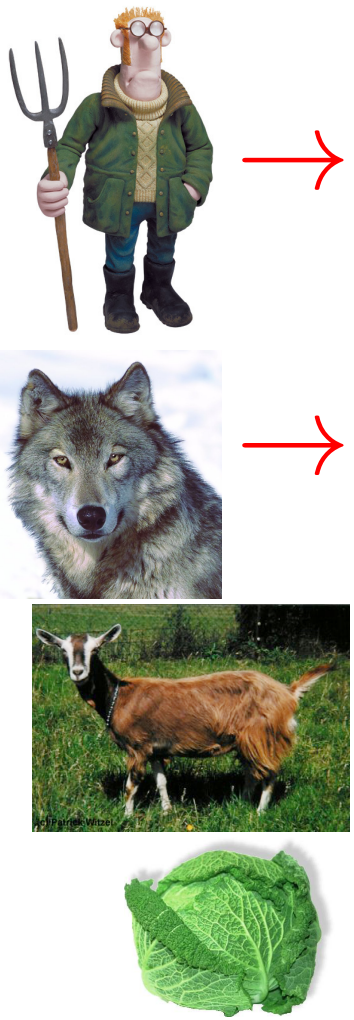
Beispiel: Wolf, Ziege, Kohlkopf

Zielsituation: nach Überquerung des Flusses



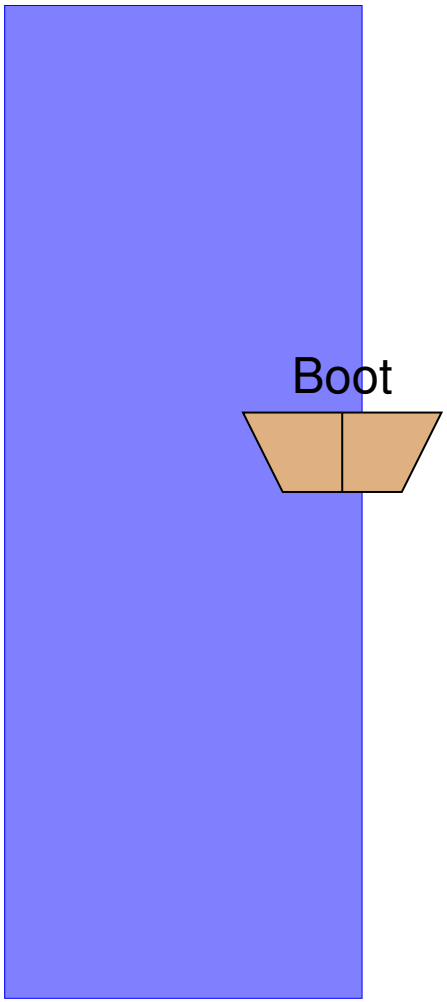
Beispiel: Wolf, Ziege, Kohlkopf

Dynamisches Modell: Beispielablauf, erster Schritt;
Farmer und Wolf setzen gemeinsam über



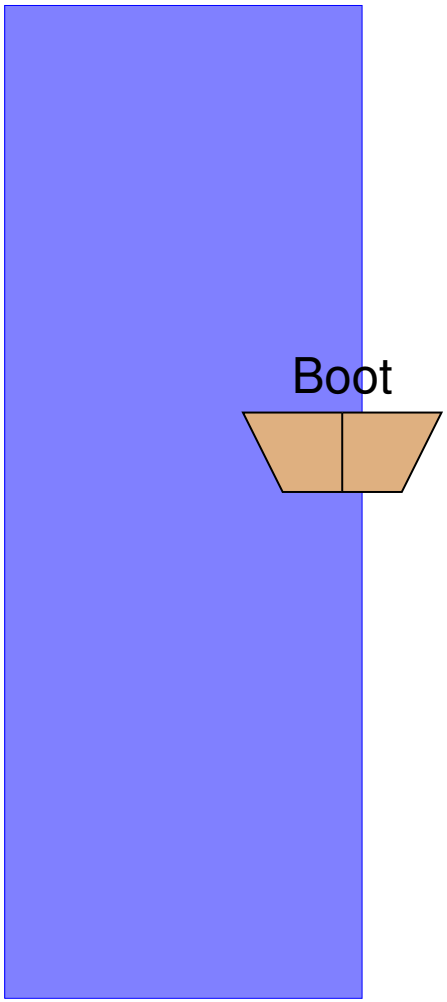
Beispiel: Wolf, Ziege, Kohlkopf

Dynamisches Modell: Beispielablauf, zweiter Schritt



Beispiel: Wolf, Ziege, Kohlkopf

Dynamisches Modell: Beispielablauf, zweiter Schritt;
Ziege frisst Kohlkopf



Syntax und Semantik

Man unterscheidet bei der Modellierung zwischen:

- Syntax: Symbole und Formen/Diagramme, die für die Darstellung des Modells genutzt werden dürfen

  Im Beispiel: Bild der Ziege, blaue Fläche, etc.
- Semantik: Bedeutung, die sich jeweils dahinter verbirgt

  Im Beispiel: Die blaue Fläche symbolisiert den Fluss.
  Die Pfeile bedeuten: „Fluss wird überquert“. Etc.

Syntax und Semantik

Man unterscheidet bei der Modellierung zwischen:

- Syntax: Symbole und Formen/Diagramme, die für die Darstellung des Modells genutzt werden dürfen

  Im Beispiel: Bild der Ziege, blaue Fläche, etc.
- Semantik: Bedeutung, die sich jeweils dahinter verbirgt

  Im Beispiel: Die blaue Fläche symbolisiert den Fluss.
  Die Pfeile bedeuten: „Fluss wird überquert“. Etc.

Zu einer Syntax gibt es nicht immer eine dazugehörige Semantik.
(Im Beispiel ist die Semantik sehr vage.)

Syntax und Semantik

Man unterscheidet bei der Modellierung zwischen:

- Syntax: Symbole und Formen/Diagramme, die für die Darstellung des Modells genutzt werden dürfen

  Im Beispiel: Bild der Ziege, blaue Fläche, etc.
- Semantik: Bedeutung, die sich jeweils dahinter verbirgt

  Im Beispiel: Die blaue Fläche symbolisiert den Fluss.
  Die Pfeile bedeuten: „Fluss wird überquert“. Etc.

Zu einer Syntax gibt es nicht immer eine dazugehörige Semantik.
(Im Beispiel ist die Semantik sehr vage.)

Wünschenswert ist jedoch im Allgemeinen, dass die Bedeutung aller Symbole und Formen möglichst präzise festgelegt wird.

$\rightsquigarrow$ Einigung auf eine gemeinsame Sprache/Notation, oder auf gemeinsame visuelle Beschreibungen; zur Vermeidung von Missverständnissen

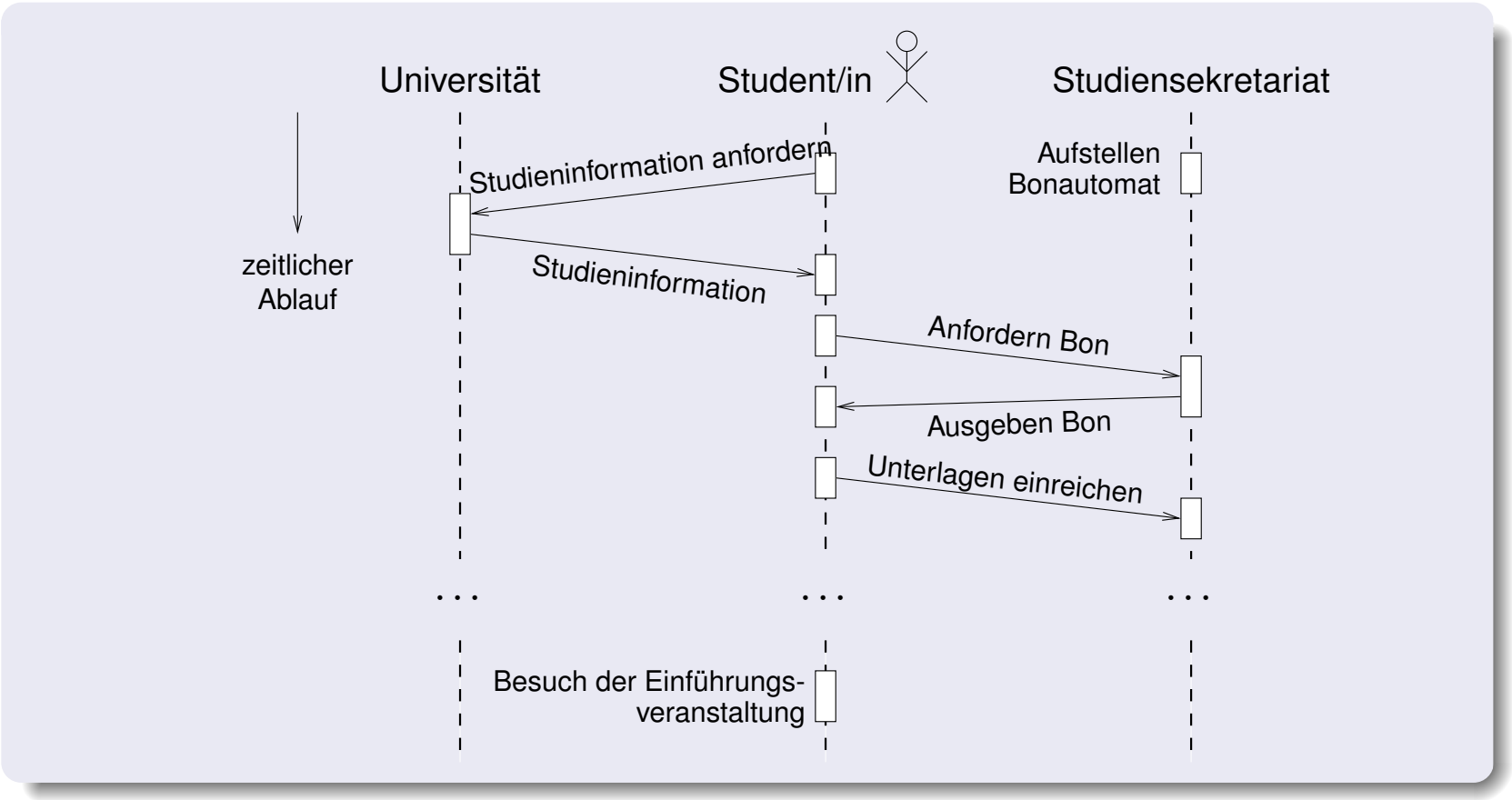
Ein weiteres Beispiel: Einschreiben an der Universität

Szenario: Eine Reorganisation der Universität, und insbesondere des Studiensekretariats, das für Einschreibungen zuständig ist, steht an.

Hierzu soll der Ablauf des Einschreibens neuer Studierender modelliert werden . . .

Ein weiteres Beispiel: Einschreiben an der Universität

Darstellung des zeitlichen Ablaufs durch Symbolisieren der beteiligten Partner als Linien und der Kommunikation durch Pfeile (Ausschnitt)



Ein weiteres Beispiel: Einschreiben an der Universität

Solche Diagramme sind Bestandteil von UML.
Sie werden Sequenzdiagramme (engl. sequence diagrams, auch message sequence charts) genannt.

Einige mathematische Hilfsmittel

Mengen

Menge

Menge M von Elementen, oft beschrieben als Aufzählung

$$M = \{0, 2, 4, 6, 8, \ldots\}$$

Mengen

Menge

Menge M von Elementen, oft beschrieben als Aufzählung

$$M = \{0, 2, 4, 6, 8, \dots\}$$

oder als Menge von Elementen mit einer bestimmten Eigenschaft

$$M = \{n \mid n \in \mathbb{N}_0 \text{ und } n \text{ gerade}\}.$$

Mengen

Menge

Menge M von Elementen, oft beschrieben als Aufzählung

$$M = \{0, 2, 4, 6, 8, \dots\}$$

oder als Menge von Elementen mit einer bestimmten Eigenschaft

$$M = \{n \mid n \in \mathbb{N}_0 \text{ und } n \text{ gerade}\}.$$

Allgemeines Format:

$$M = \{x \mid E(x)\}$$

M ist Menge aller Elemente, die die Eigenschaft E erfüllen.

Mengen

Menge

Menge M von Elementen, oft beschrieben als Aufzählung

$$M = \{0, 2, 4, 6, 8, \dots\}$$

oder als Menge von Elementen mit einer bestimmten Eigenschaft

$$M = \{n \mid n \in \mathbb{N}_0 \text{ und } n \text{ gerade}\}.$$

Allgemeines Format:

$$M = \{x \mid E(x)\}$$

M ist Menge aller Elemente, die die Eigenschaft E erfüllen.

$$M = \{x \in X \mid E(x)\}$$

M ist Menge aller entsprechenden Elemente aus Grundmenge X.

Mengen

Menge

Menge M von Elementen, oft beschrieben als Aufzählung

$$M = \{0, 2, 4, 6, 8, \dots\}$$

oder als Menge von Elementen mit einer bestimmten Eigenschaft

$$M = \{n \mid n \in \mathbb{N}_0 \text{ und } n \text{ gerade}\} = \{n \in \mathbb{N}_0 \mid n \text{ gerade}\}.$$

Allgemeines Format:

$$M = \{x \mid E(x)\}$$

M ist Menge aller Elemente, die die Eigenschaft E erfüllen.

$$M = \{x \in X \mid E(x)\}$$

M ist Menge aller entsprechenden Elemente aus Grundmenge X.

Mengen

Bemerkungen:

- Die Elemente einer Menge sind ungeordnet, das heißt, ihre Ordnung spielt keine Rolle. Beispielsweise gilt:

$$\{1,2,3\} = \{1,3,2\} = \{2,1,3\} = \{2,3,1\} = \{3,1,2\} = \{3,2,1\}$$

Mengen

Bemerkungen:

- Die Elemente einer Menge sind ungeordnet, das heißt, ihre Ordnung spielt keine Rolle. Beispielsweise gilt:

$$\{1,2,3\} = \{1,3,2\} = \{2,1,3\} = \{2,3,1\} = \{3,1,2\} = \{3,2,1\}$$

- Ein Element kann nicht mehrfach in einer Menge auftreten. Es ist entweder in der Menge, oder es ist nicht in der Menge. Beispielsweise gilt:

$$\{1,2,3,4,4\} = \{1,2,3,4\}$$

Mengen

Bemerkungen:

- Die Elemente einer Menge sind ungeordnet, das heißt, ihre Ordnung spielt keine Rolle. Beispielsweise gilt:

$$\{1,2,3\} = \{1,3,2\} = \{2,1,3\} = \{2,3,1\} = \{3,1,2\} = \{3,2,1\}$$

- Ein Element kann nicht mehrfach in einer Menge auftreten. Es ist entweder in der Menge, oder es ist nicht in der Menge. Beispielsweise gilt:

$$\{1,2,3,4,4\} = \{1,2,3,4\} \neq \{1,2,3\}$$

Mengen

Element einer Menge

Wir schreiben $a \in M$, falls ein Element a in der Menge M enthalten ist.

Mengen

Element einer Menge

Wir schreiben $a \in M$, falls ein Element a in der Menge M enthalten ist.

Anzahl der Elemente einer Menge

Für eine endliche Menge M gibt $|M|$ die Anzahl ihrer Elemente an.

Mengen

Element einer Menge

Wir schreiben $a \in M$, falls ein Element a in der Menge M enthalten ist.

Anzahl der Elemente einer Menge

Für eine endliche Menge M gibt $|M|$ die Anzahl ihrer Elemente an.

Teilmengenbeziehung

Wir schreiben $A \subseteq B$, falls jedes Element von A auch in B enthalten ist.

Mengen

Element einer Menge

Wir schreiben $a \in M$, falls ein Element a in der Menge M enthalten ist.

Anzahl der Elemente einer Menge

Für eine endliche Menge M gibt $|M|$ die Anzahl ihrer Elemente an.

Teilmengenbeziehung

Wir schreiben $A \subseteq B$, falls jedes Element von A auch in B enthalten ist.

Leere Menge

Mit $\emptyset$ oder $\{\}$ bezeichnen wir die leere Menge. Sie enthält keine Elemente und ist (echte) Teilmenge jeder anderen Menge.

Mengen

Mengenvereinigung

Die Vereinigung zweier Mengen A und B ist diejenige Menge, welche die Elemente enthält, die in A oder B (oder in beiden) vorkommen. Man schreibt dafür $A \cup B$.

$$A \cup B = \{x \mid x \in A \text{ oder } x \in B\}$$

Mengen

Mengenvereinigung

Die Vereinigung zweier Mengen A und B ist diejenige Menge, welche die Elemente enthält, die in A oder B (oder in beiden) vorkommen. Man schreibt dafür $A \cup B$.

$$A \cup B = \{x \mid x \in A \text{ oder } x \in B\}$$

Mengenschnitt

Der Schnitt zweier Mengen A und B ist diejenige Menge, welche die Element enthält, die sowohl in A als auch in B vorkommen. Man schreibt dafür $A \cap B$.

$$A \cap B = \{x \mid x \in A \text{ und } x \in B\}$$

Mengen

Mengendifferenz

Die Differenz zweier Mengen A und B ist diejenige Menge, welche die Elemente enthält, die in A vorkommen und in B nicht vorkommen. Man schreibt dafür $A \setminus B$.

$$A \setminus B = \{x \mid x \in A \text{ und } x \notin B\}$$

Mengen

Mengendifferenz

Die Differenz zweier Mengen A und B ist diejenige Menge, welche die Elemente enthält, die in A vorkommen und in B nicht vorkommen. Man schreibt dafür $A \setminus B$.

$$A \setminus B = \{x \mid x \in A \text{ und } x \notin B\} = \{x \in A \mid x \notin B\}$$

Mengen

Mengendifferenz

Die Differenz zweier Mengen A und B ist diejenige Menge, welche die Elemente enthält, die in A vorkommen und in B nicht vorkommen. Man schreibt dafür $A \setminus B$.

$$A \setminus B = \{x \mid x \in A \text{ und } x \notin B\} = \{x \in A \mid x \notin B\}$$

Beispiele:

- $\{0,1,2,3,4,5\} \setminus \{0\} = \{1,2,3,4,5\}$
- $\{a,b,c\} \setminus \{c,d\} = \{a,b\}$
- $\{1,2,3\} \setminus \emptyset = \{1,2,3\}$
- $\{1,2,3\} \setminus \mathbb{N}_0 = \emptyset$
- $(\{1,2,3\} \setminus \{1,2\}) \setminus \{1\} \neq \{1,2,3\} \setminus (\{1,2\} \setminus \{1\})$

Mengen

Kreuzprodukt (Kartesisches Produkt)

Das Kreuzprodukt zweier Mengen A und B ist diejenige Menge, welche alle Paare (a, b) enthält, wobei die erste Komponente des Paars aus A, die zweite aus B kommt. Man schreibt dafür $A \times B$.

$$A \times B = \{(a, b) \mid a \in A \text{ und } b \in B\}$$

Mengen

Kreuzprodukt (Kartesisches Produkt)

Das Kreuzprodukt zweier Mengen A und B ist diejenige Menge, welche alle Paare (a, b) enthält, wobei die erste Komponente des Paars aus A, die zweite aus B kommt. Man schreibt dafür $A \times B$.

$$A \times B = \{(a, b) \mid a \in A \text{ und } b \in B\}$$

Beispiele:

- $\{1,2\} \times \{3,4,5\} = \{(1,3),(1,4),(1,5),(2,3),(2,4),(2,5)\}$

Mengen

Kreuzprodukt (Kartesisches Produkt)

Das Kreuzprodukt zweier Mengen A und B ist diejenige Menge, welche alle Paare (a, b) enthält, wobei die erste Komponente des Paars aus A, die zweite aus B kommt. Man schreibt dafür $A \times B$.

$$A \times B = \{(a, b) \mid a \in A \text{ und } b \in B\}$$

Beispiele:

- $\{1,2\} \times \{3,4,5\} = \{(1,3),(1,4),(1,5),(2,3),(2,4),(2,5)\}$
- $\{1,2\} \times \emptyset = \emptyset$

Mengen

Kreuzprodukt (Kartesisches Produkt)

Das Kreuzprodukt zweier Mengen A und B ist diejenige Menge, welche alle Paare (a,b) enthält, wobei die erste Komponente des Paars aus A, die zweite aus B kommt. Man schreibt dafür $A \times B$.

$$A \times B = \{(a,b) \mid a \in A \text{ und } b \in B\}$$

Beispiele:

- $\{1,2\} \times \{3,4,5\} = \{(1,3),(1,4),(1,5),(2,3),(2,4),(2,5)\}$
- $\{1,2\} \times \emptyset = \emptyset$

Anmerkungen:

- Für endliche Mengen A und B gilt $|A \times B| = |A| \cdot |B|$.

Mengen

Kreuzprodukt (Kartesisches Produkt)

Das Kreuzprodukt zweier Mengen A und B ist diejenige Menge, welche alle Paare (a,b) enthält, wobei die erste Komponente des Paars aus A, die zweite aus B kommt. Man schreibt dafür $A \times B$.

$$A \times B = \{(a,b) \mid a \in A \text{ und } b \in B\}$$

Beispiele:

- $\{1,2\} \times \{3,4,5\} = \{(1,3),(1,4),(1,5),(2,3),(2,4),(2,5)\}$
- $\{1,2\} \times \emptyset = \emptyset$

Anmerkungen:

- Für endliche Mengen A und B gilt $|A \times B| = |A| \cdot |B|$.
- Wenn $A \subseteq B$, dann $A \times C \subseteq B \times C$.

Mengen

Kreuzprodukt (Kartesisches Produkt)

Das Kreuzprodukt zweier Mengen A und B ist diejenige Menge, welche alle Paare (a,b) enthält, wobei die erste Komponente des Paars aus A, die zweite aus B kommt. Man schreibt dafür $A \times B$.

$$A \times B = \{(a,b) \mid a \in A \text{ und } b \in B\}$$

Beispiele:

- $\{1,2\} \times \{3,4,5\} = \{(1,3),(1,4),(1,5),(2,3),(2,4),(2,5)\}$
- $\{1,2\} \times \emptyset = \emptyset$

Anmerkungen:

- Für endliche Mengen A und B gilt $|A \times B| = |A| \cdot |B|$.
- Wenn $A \subseteq B$, dann $A \times C \subseteq B \times C$ und $C \times A \subseteq C \times B$.

Mengen

Weitere Bemerkungen:

- Wir betrachten nicht nur Paare, sondern auch Tupel aus mehr als zwei Komponenten (Tripel, Quadrupel, Quintupel, ...).
  Ein Tupel $(a_1, \ldots, a_n)$ bestehend aus n Komponenten heißt auch n-Tupel.

Mengen

Weitere Bemerkungen:

- Wir betrachten nicht nur Paare, sondern auch Tupel aus mehr als zwei Komponenten (Tripel, Quadrupel, Quintupel, ...). Ein Tupel $(a_1, \ldots, a_n)$ bestehend aus n Komponenten heißt auch n-Tupel.
- In einem Tupel sind die Komponenten geordnet! Es gilt z.B.:

$$(1,2,3) \neq (1,3,2) \in \mathbb{N}_0 \times \mathbb{N}_0 \times \mathbb{N}_0$$

Mengen

Weitere Bemerkungen:

- Wir betrachten nicht nur Paare, sondern auch Tupel aus mehr als zwei Komponenten (Tripel, Quadrupel, Quintupel, ...). Ein Tupel $(a_1, \ldots, a_n)$ bestehend aus n Komponenten heißt auch *n*-Tupel.
- In einem Tupel sind die Komponenten geordnet! Es gilt z.B.:

$$(1,2,3) \neq (1,3,2) \in \mathbb{N}_0 \times \mathbb{N}_0 \times \mathbb{N}_0$$

- Ein Element kann mehrfach in einem Tupel auftreten. Tupel unterschiedlicher Länge sind immer verschieden. Beispielsweise:

$$(1,2,3,4) \neq (1,2,3,4,4)$$

Mengen

Potenzmenge

Die Potenzmenge einer Menge M ist diejenige Menge, welche alle Teilmengen von M enthält. Man schreibt dafür $\mathscr{P}(M)$.

$$\mathscr{P}(M) = \{A \mid A \subseteq M\}$$

Mengen

Potenzmenge

Die Potenzmenge einer Menge M ist diejenige Menge, welche alle Teilmengen von M enthält. Man schreibt dafür $\mathscr{P}(M)$.

$$\mathscr{P}(M) = \{A \mid A \subseteq M\}$$

Beispiele:

- $\mathscr{P}(\{1,2,3\}) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$

Mengen

Potenzmenge

Die Potenzmenge einer Menge M ist diejenige Menge, welche alle Teilmengen von M enthält. Man schreibt dafür $\mathscr{P}(M)$.

$$\mathscr{P}(M) = \{A \mid A \subseteq M\}$$

Beispiele:

- $\mathscr{P}(\{1,2,3\}) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$
- $\mathscr{P}(\emptyset) = \{\emptyset\}$

Mengen

Potenzmenge

Die Potenzmenge einer Menge M ist diejenige Menge, welche alle Teilmengen von M enthält. Man schreibt dafür $\mathscr{P}(M)$.

$$\mathscr{P}(M) = \{A \mid A \subseteq M\}$$

Beispiele:

- $\mathscr{P}(\{1,2,3\}) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$
- $\mathscr{P}(\emptyset) = \{\emptyset\}$
- $\mathscr{P}(\mathscr{P}(\emptyset)) = \{\emptyset, \{\emptyset\}\}$

Mengen

Potenzmenge

Die Potenzmenge einer Menge M ist diejenige Menge, welche alle Teilmengen von M enthält. Man schreibt dafür $\mathscr{P}(M)$.

$$\mathscr{P}(M) = \{A \mid A \subseteq M\}$$

Beispiele:

- $\mathscr{P}(\{1,2,3\}) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$
- $\mathscr{P}(\emptyset) = \{\emptyset\}$
- $\mathscr{P}(\mathscr{P}(\emptyset)) = \{\emptyset, \{\emptyset\}\}$

Anmerkung: Für endliche Mengen M gilt $|\mathscr{P}(M)| = 2^{|M|}$.

Mengen

Beispiel: Zustandsmodellierung

Angenommen, wir betrachten einen einfachen Snackautomaten für Riegel und Chips.

Mengen

Beispiel: Zustandsmodellierung

Angenommen, wir betrachten einen einfachen Snackautomaten für Riegel und Chips. Von jedem dieser beiden Snacks hat er maximal 30 Stück auf Vorrat.

Mengen

Beispiel: Zustandsmodellierung

Angenommen, wir betrachten einen einfachen Snackautomaten für Riegel und Chips. Von jedem dieser beiden Snacks hat er maximal 30 Stück auf Vorrat. Der Automat hat eine gelbe und eine rote Warnleuchte („kein Wechselgeld mehr“ bzw. „keine Scheine mehr akzeptiert“), die unabhängig voneinander leuchten können.

Mengen

Beispiel: Zustandsmodellierung

Angenommen, wir betrachten einen einfachen Snackautomaten für Riegel und Chips. Von jedem dieser beiden Snacks hat er maximal 30 Stück auf Vorrat. Der Automat hat eine gelbe und eine rote Warnleuchte („kein Wechselgeld mehr“ bzw. „keine Scheine mehr akzeptiert“), die unabhängig voneinander leuchten können. Die Menge der möglichen Zustände dieses Automaten können wir als

$$\mathscr{P}(\{\text{gelb}, \text{rot}\}) \times \{0, 1, \ldots, 30\} \times \{0, 1, \ldots, 30\}$$

beschreiben.

Mengen

Beispiel: Zustandsmodellierung

Angenommen, wir betrachten einen einfachen Snackautomaten für Riegel und Chips. Von jedem dieser beiden Snacks hat er maximal 30 Stück auf Vorrat. Der Automat hat eine gelbe und eine rote Warnleuchte („kein Wechselgeld mehr“ bzw. „keine Scheine mehr akzeptiert“), die unabhängig voneinander leuchten können. Die Menge der möglichen Zustände dieses Automaten können wir als

$$\mathscr{P}(\{\text{gelb}, \text{rot}\}) \times \{0, 1, \ldots, 30\} \times \{0, 1, \ldots, 30\}$$

beschreiben. Das Element $(\emptyset, 20, 10)$ dieser Menge zum Beispiel entspricht dem Zustand, in dem beide Warnleuchten ausgeschaltet sind und noch 20 Riegel und 10 Packungen Chips vorrätig.

Mengen

Beispiel: Zustandsmodellierung

Angenommen, wir betrachten einen einfachen Snackautomaten für Riegel und Chips. Von jedem dieser beiden Snacks hat er maximal 30 Stück auf Vorrat. Der Automat hat eine gelbe und eine rote Warnleuchte („kein Wechselgeld mehr“ bzw. „keine Scheine mehr akzeptiert“), die unabhängig voneinander leuchten können. Die Menge der möglichen Zustände dieses Automaten können wir als

$$\mathscr{P}(\{\text{gelb}, \text{rot}\}) \times \{0, 1, \ldots, 30\} \times \{0, 1, \ldots, 30\}$$

beschreiben. Das Element $(\emptyset, 20, 10)$ dieser Menge zum Beispiel entspricht dem Zustand, in dem beide Warnleuchten ausgeschaltet sind und noch 20 Riegel und 10 Packungen Chips vorrätig. Wären bei diesem Vorrat die Warnleuchten beide eingeschaltet, so befände sich der Automat stattdessen im Zustand $(\{\text{gelb}, \text{rot}\}, 20, 10)$.

Funktionen

Funktion (Abbildung)

Funktionen bilden Elemente eines Definitionsbereiches auf Elemente eines Wertebereiches ab. Man schreibt: „$f : A \rightarrow B$“.

Funktionen

Funktion (Abbildung)

Funktionen bilden Elemente eines Definitionsbereiches auf Elemente eines Wertebereiches ab. Man schreibt: „$f : A \rightarrow B$“. Paare aus einem Element a des Definitionsbereiches A und dem (eindeutig gegebenen) Element $b = f(a)$ des Wertebereiches B, auf welches die Funktion es abbildet, notiert man in der Form „$a \mapsto b$“.

Funktionen

Funktion (Abbildung)

Funktionen bilden Elemente eines Definitionsbereiches auf Elemente eines Wertebereiches ab. Man schreibt: „$f : A \to B$“.
Paare aus einem Element a des Definitionsbereiches A und dem (eindeutig gegebenen) Element $b = f(a)$ des Wertebereiches B, auf welches die Funktion es abbildet, notiert man in der Form „$a \mapsto b$“.
Die gleiche Notation verwendet man, um eine allgemeine Zuordnungsvorschrift anzugeben: „$a \mapsto f(a)$“.

Funktionen

Funktion (Abbildung)

Funktionen bilden Elemente eines Definitionsbereiches auf Elemente eines Wertebereiches ab. Man schreibt: „$f : A \to B$“.
Paare aus einem Element a des Definitionsbereiches A und dem (eindeutig gegebenen) Element $b = f(a)$ des Wertebereiches B, auf welches die Funktion es abbildet, notiert man in der Form „$a \mapsto b$“.
Die gleiche Notation verwendet man, um eine allgemeine Zuordnungsvorschrift anzugeben: „$a \mapsto f(a)$“.

Beispiel: Quadratfunktion auf der Menge der ganzen Zahlen

$$f : \mathbb{Z} \to \mathbb{N}_0$$

Funktionen

Funktion (Abbildung)

Funktionen bilden Elemente eines Definitionsbereiches auf Elemente eines Wertebereiches ab. Man schreibt: „$f : A \to B$“.
Paare aus einem Element a des Definitionsbereiches A und dem (eindeutig gegebenen) Element $b = f(a)$ des Wertebereiches B, auf welches die Funktion es abbildet, notiert man in der Form „$a \mapsto b$“.
Die gleiche Notation verwendet man, um eine allgemeine Zuordnungsvorschrift anzugeben: „$a \mapsto f(a)$“.

Beispiel: Quadratfunktion auf der Menge der ganzen Zahlen

$$f : \mathbb{Z} \to \mathbb{N}_0, \quad f(z) = z^2$$

Funktionen

Funktion (Abbildung)

Funktionen bilden Elemente eines Definitionsbereiches auf Elemente eines Wertebereiches ab. Man schreibt: „$f : A \to B$“.
Paare aus einem Element a des Definitionsbereiches A und dem (eindeutig gegebenen) Element $b = f(a)$ des Wertebereiches B, auf welches die Funktion es abbildet, notiert man in der Form „$a \mapsto b$“.
Die gleiche Notation verwendet man, um eine allgemeine Zuordnungsvorschrift anzugeben: „$a \mapsto f(a)$“.

Beispiel: Quadratfunktion auf der Menge der ganzen Zahlen

$$f : \mathbb{Z} \to \mathbb{N}_0, \quad f(z) = z^2, \quad \text{bzw. Angabe als: } z \mapsto z^2$$

Funktionen

Funktion (Abbildung)

Funktionen bilden Elemente eines Definitionsbereiches auf Elemente eines Wertebereiches ab. Man schreibt: „$f : A \to B$“. Paare aus einem Element a des Definitionsbereiches A und dem (eindeutig gegebenen) Element $b = f(a)$ des Wertebereiches B, auf welches die Funktion es abbildet, notiert man in der Form „$a \mapsto b$“.
Die gleiche Notation verwendet man, um eine allgemeine Zuordnungsvorschrift anzugeben: „$a \mapsto f(a)$“.

Beispiel: Quadratfunktion auf der Menge der ganzen Zahlen

$$f : \mathbb{Z} \to \mathbb{N}_0, \quad f(z) = z^2, \quad \text{bzw. Angabe als: } z \mapsto z^2$$

Angabe konkreter Paare:

$$\ldots, -3 \mapsto 9, -2 \mapsto 4, -1 \mapsto 1, 0 \mapsto 0, 1 \mapsto 1, 2 \mapsto 4, 3 \mapsto 9, \ldots$$

Graphen

Graphen sind netzartige Strukturen, bestehend aus Knoten und Kanten. Sie bilden die Grundlagen vieler diagrammatischer Modellierungstechniken.

Gerichteter und kantenbeschrifteter Graph

Sei L eine Menge von Beschriftungen (oder Labels).

Ein gerichteter und kantenbeschrifteter Graph $G = (V, E)$ besteht aus

- einer Knotenmenge V und
- einer Kantenmenge $E \subseteq V \times L \times V$.

Graphen

Graphen sind netzartige Strukturen, bestehend aus Knoten und Kanten. Sie bilden die Grundlagen vieler diagrammatischer Modellierungstechniken.

Gerichteter und kantenbeschrifteter Graph

Sei L eine Menge von Beschriftungen (oder Labels).

Ein gerichteter und kantenbeschrifteter Graph $G = (V, E)$ besteht aus

- einer Knotenmenge V und
- einer Kantenmenge $E \subseteq V \times L \times V$.

Bemerkung: V steht für vertices und E für edges.

Graphen

Beispiel: (Farmer, Wolf, Ziege, Kohlkopf)

$$V = \{F, W, Z, K\} \qquad L = \{\text{besitzt}, \text{frisst}\}$$

$$E = \{(F, \text{besitzt}, W), (F, \text{besitzt}, Z), (F, \text{besitzt}, K), (W, \text{frisst}, Z), (Z, \text{frisst}, K)\}$$

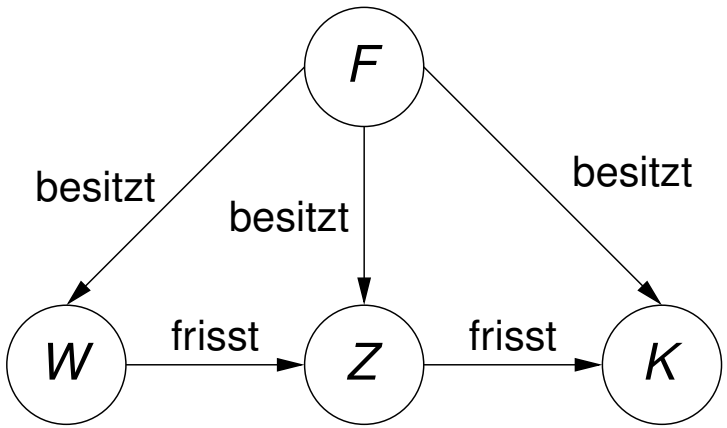
Graphen

Beispiel: (Farmer, Wolf, Ziege, Kohlkopf)

$$V = \{F, W, Z, K\} \qquad L = \{\text{besitzt}, \text{frisst}\}$$

$$E = \{(F, \text{besitzt}, W), (F, \text{besitzt}, Z), (F, \text{besitzt}, K), (W, \text{frisst}, Z), (Z, \text{frisst}, K)\}$$

Bildhaft:

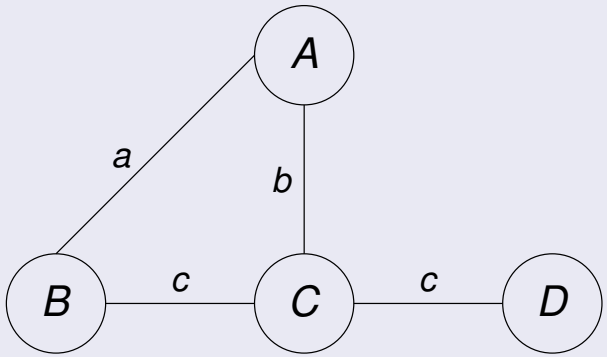


Graphen

Weitere Arten von Graphen:

Ungerichteter Graph

Bei ungerichteten Graphen spielt die Richtung der Kanten keine Rolle.

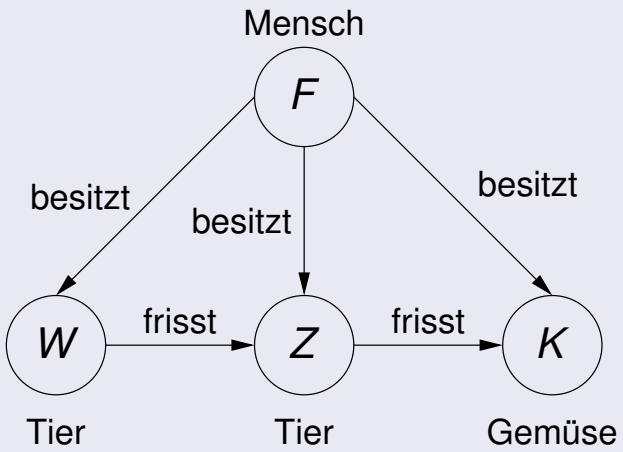


Graphen

Weitere Arten von Graphen:

Knotenbeschrifteter Graph

Auch Knoten können Beschriftungen tragen, wobei zwei verschiedene Knoten durchaus gleich beschriftet sein dürfen.

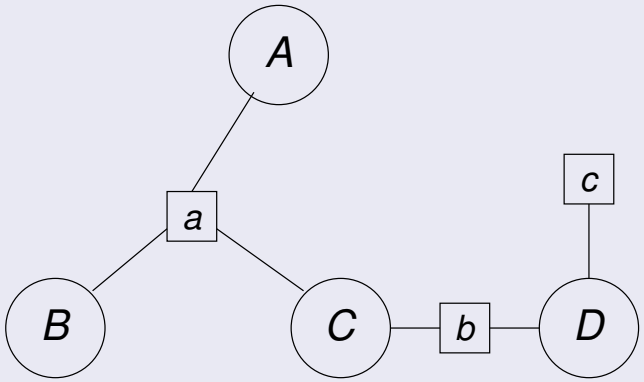


Graphen

Weitere Arten von Graphen:

Hypergraph

Bei Hypergraphen kann eine Kante (symbolisiert durch ein Quadrat oder Rechteck) mit einer beliebigen Anzahl von Knoten verbunden sein. Eventuell sind dabei die Knoten in Bezug auf die (Hyper-)Kante geordnet (wie bei gerichteten Graphen).



Graphen

Graphen können in vielfältiger Weise zur Modellierung eingesetzt werden. Wir betrachten zwei typische Fälle.

Graphen zur statischen Modellierung

Knoten sind Komponenten oder Objekte, die untereinander über Kanten verbunden sind bzw. in Beziehung stehen.

Beispiel: Beziehungen zwischen Farmer, Wolf, Ziege, Kohlkopf

Zustandsübergangsdiagramme

Graphen zur dynamischen Modellierung

Knoten sind Zustände und Kanten sind Zustandsübergänge.

Klassischer Vertreter: Zustandsübergangsdiagramme
(auch Transitionssysteme genannt)

Zustandsübergangsdiagramme

Graphen zur dynamischen Modellierung

Knoten sind Zustände und Kanten sind Zustandsübergänge.

Klassischer Vertreter: Zustandsübergangsdiagramme (auch Transitionssysteme genannt)

Zustandsübergangsdiagramm

Ein Zustandsübergangsdiagramm besteht aus einem gerichteten Graphen (Z, U), wobei Z (= Zustände) die Knotenmenge des Graphen und U (= Übergänge) die Kantenmenge des Graphen ist, und außerdem aus einem Startzustand $z_0 \in Z$.

Zustandsübergangsdiagramme

Graphen zur dynamischen Modellierung

Knoten sind Zustände und Kanten sind Zustandsübergänge.

Klassischer Vertreter: Zustandsübergangsdiagramme (auch Transitionssysteme genannt)

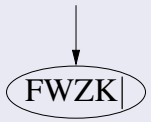
Zustandsübergangsdiagramm

Ein Zustandsübergangsdiagramm besteht aus einem gerichteten Graphen (Z, U), wobei Z (= Zustände) die Knotenmenge des Graphen und U (= Übergänge) die Kantenmenge des Graphen ist, und außerdem aus einem Startzustand $z_0 \in Z$.

Zustandsübergangsdiagramme werden grafisch wie normale gerichtete Graphen dargestellt. Der Startzustand (auch Anfangszustand genannt) wird dabei meist durch eine eingehende Pfeilspitze gekennzeichnet.

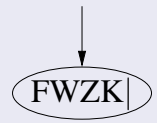
Zustandsübergangsdiagramme

Beispiel: Zustandsübergangsdiagramm für das Wolf-Ziege-Kohlkopf-Problem



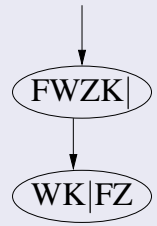
Zustandsübergangsdiagramme

Beispiel: Zustandsübergangsdiagramm für das Wolf-Ziege-Kohlkopf-Problem



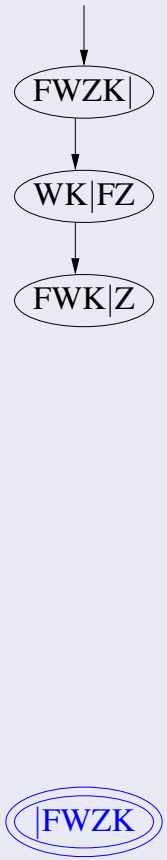
Zustandsübergangsdiagramme

Beispiel: Zustandsübergangsdiagramm für das Wolf-Ziege-Kohlkopf-Problem



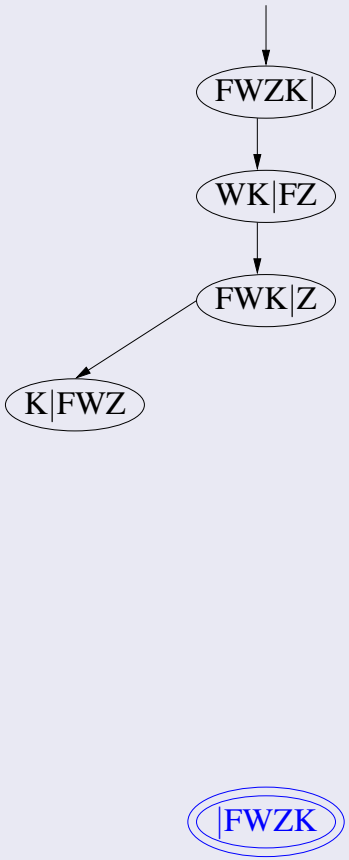
Zustandsübergangsdiagramme

Beispiel: Zustandsübergangsdiagramm für das Wolf-Ziege-Kohlkopf-Problem



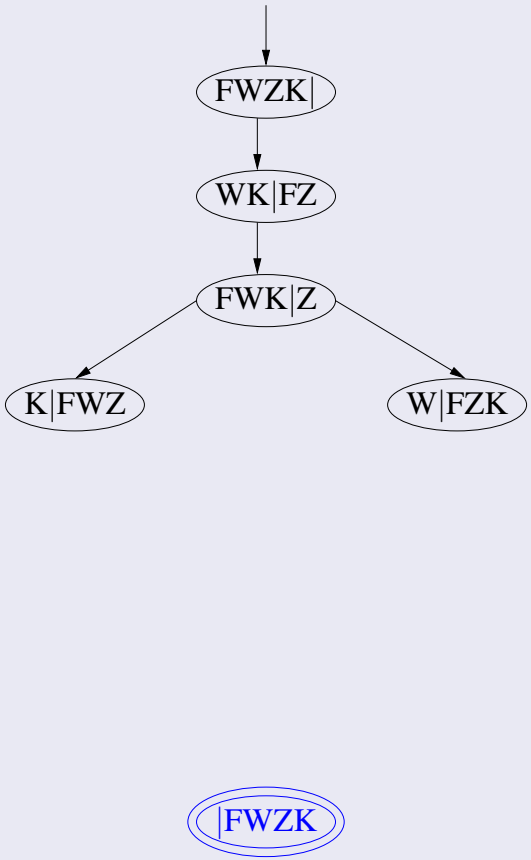
Zustandsübergangsdiagramme

Beispiel: Zustandsübergangsdiagramm für das Wolf-Ziege-Kohlkopf-Problem



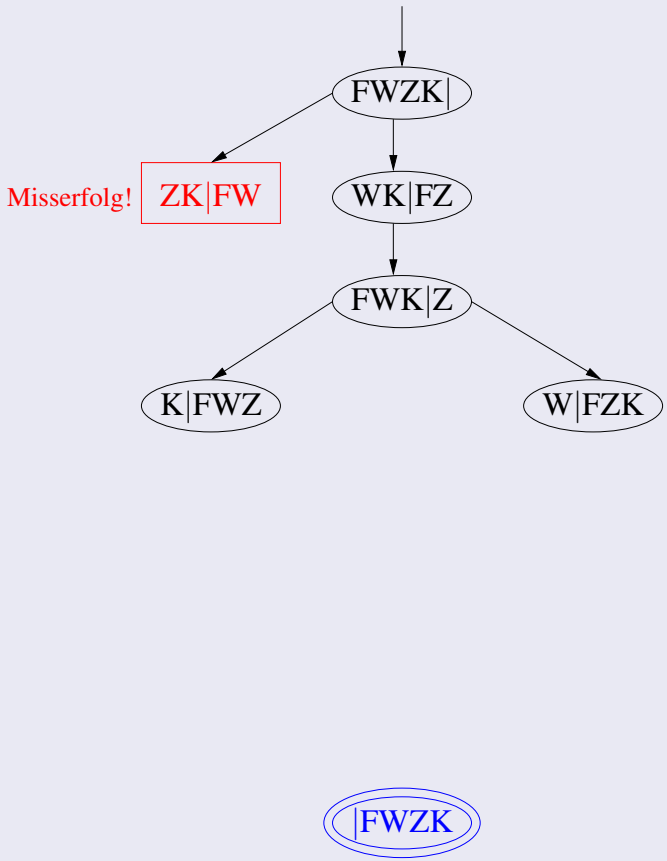
Zustandsübergangsdiagramme

Beispiel: Zustandsübergangsdiagramm für das Wolf-Ziege-Kohlkopf-Problem



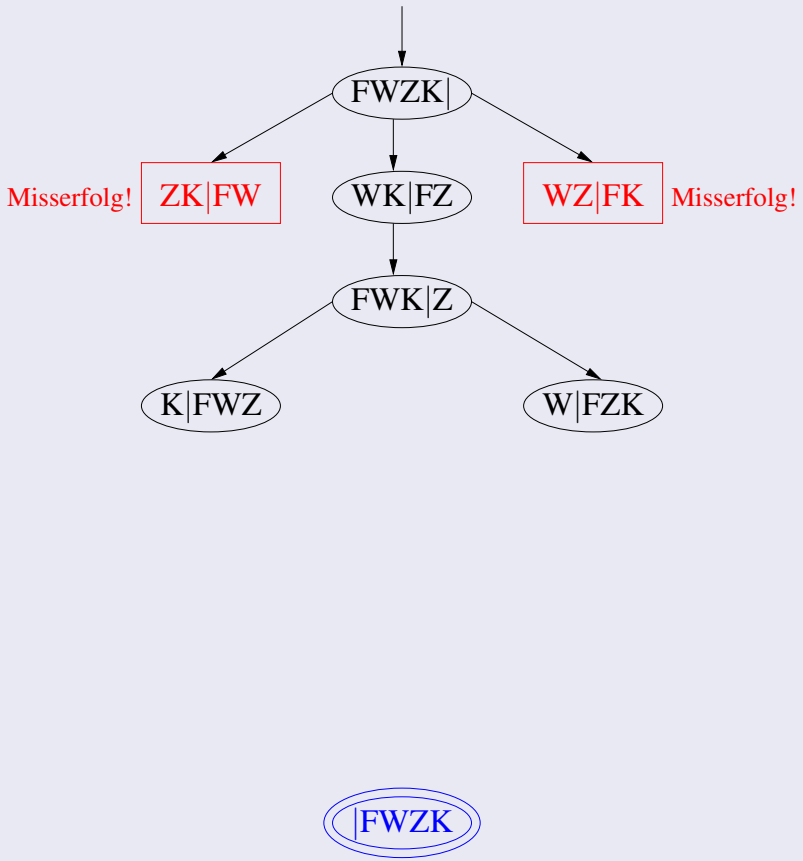
Zustandsübergangsdiagramme

Beispiel: Zustandsübergangsdiagramm für das Wolf-Ziege-Kohlkopf-Problem



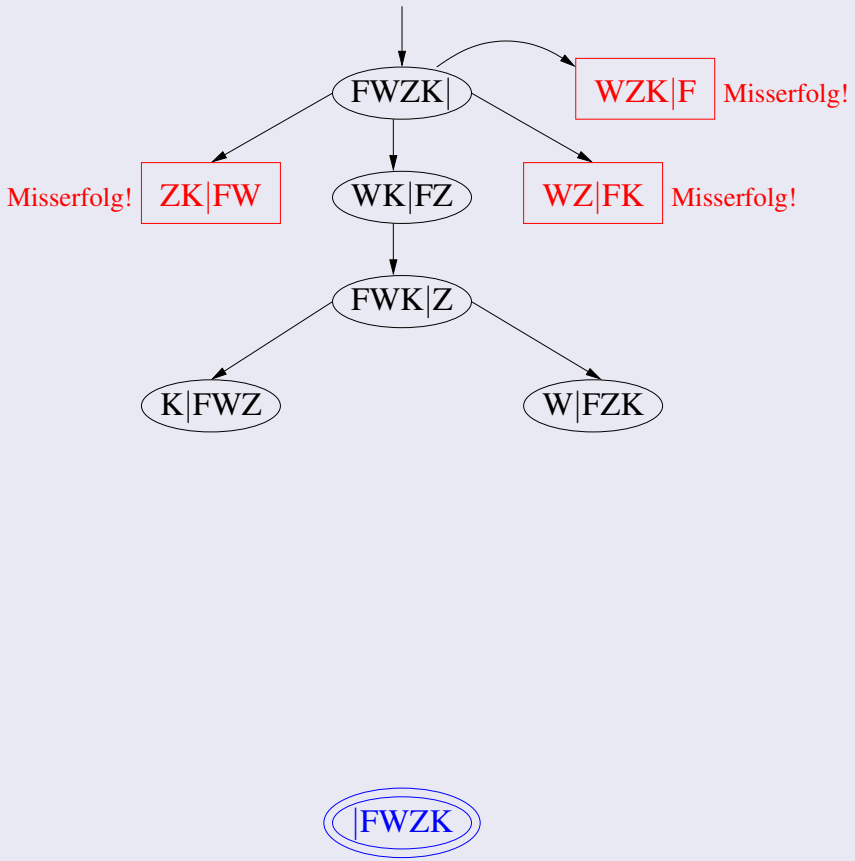
Zustandsübergangsdiagramme

Beispiel: Zustandsübergangsdiagramm für das Wolf-Ziege-Kohlkopf-Problem



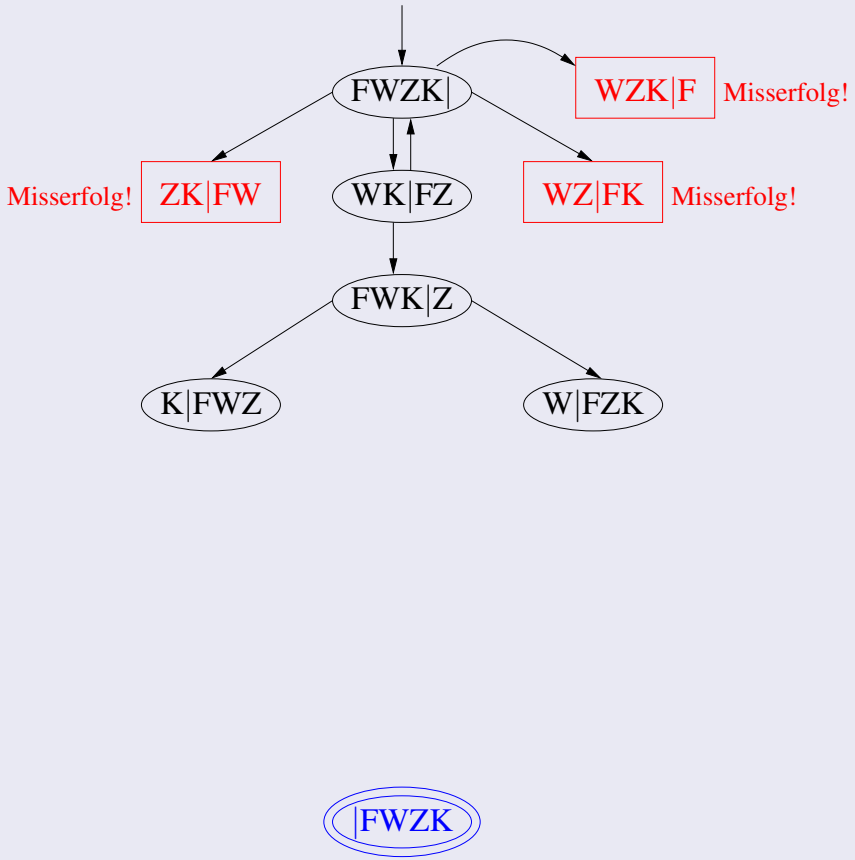
Zustandsübergangsdiagramme

Beispiel: Zustandsübergangsdiagramm für das Wolf-Ziege-Kohlkopf-Problem



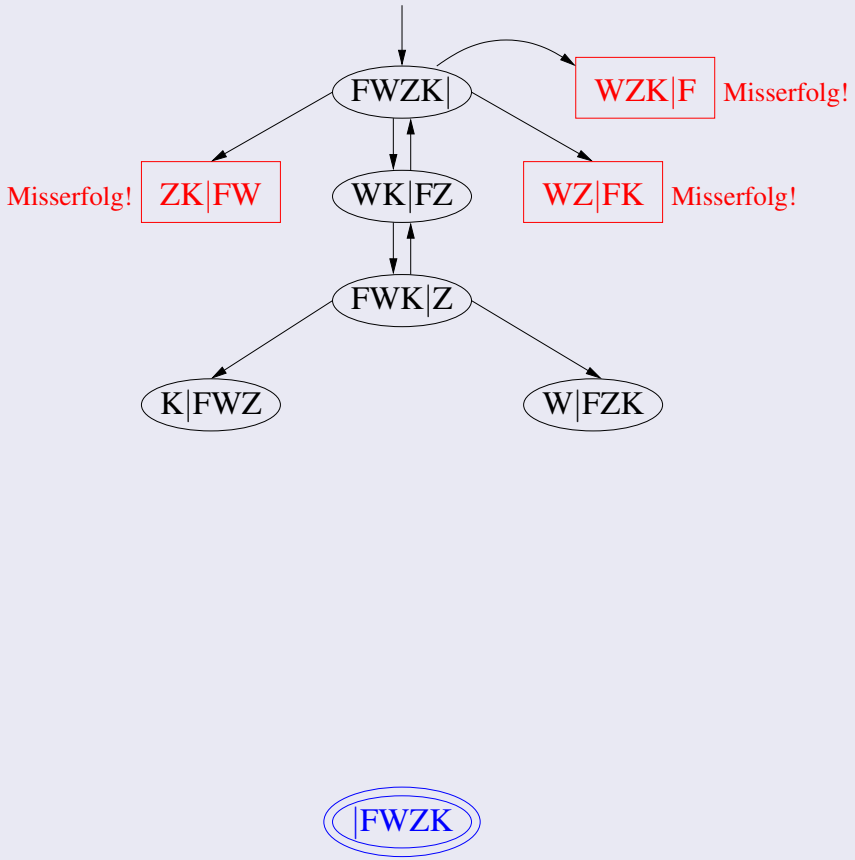
Zustandsübergangsdiagramme

Beispiel: Zustandsübergangsdiagramm für das Wolf-Ziege-Kohlkopf-Problem



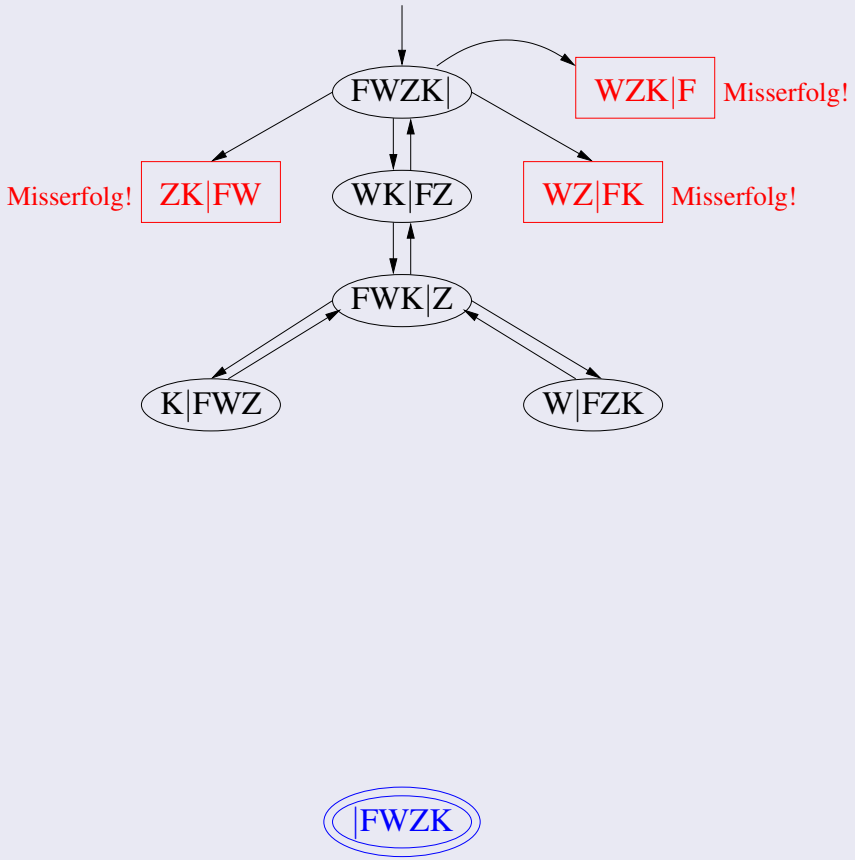
Zustandsübergangsdiagramme

Beispiel: Zustandsübergangsdiagramm für das Wolf-Ziege-Kohlkopf-Problem



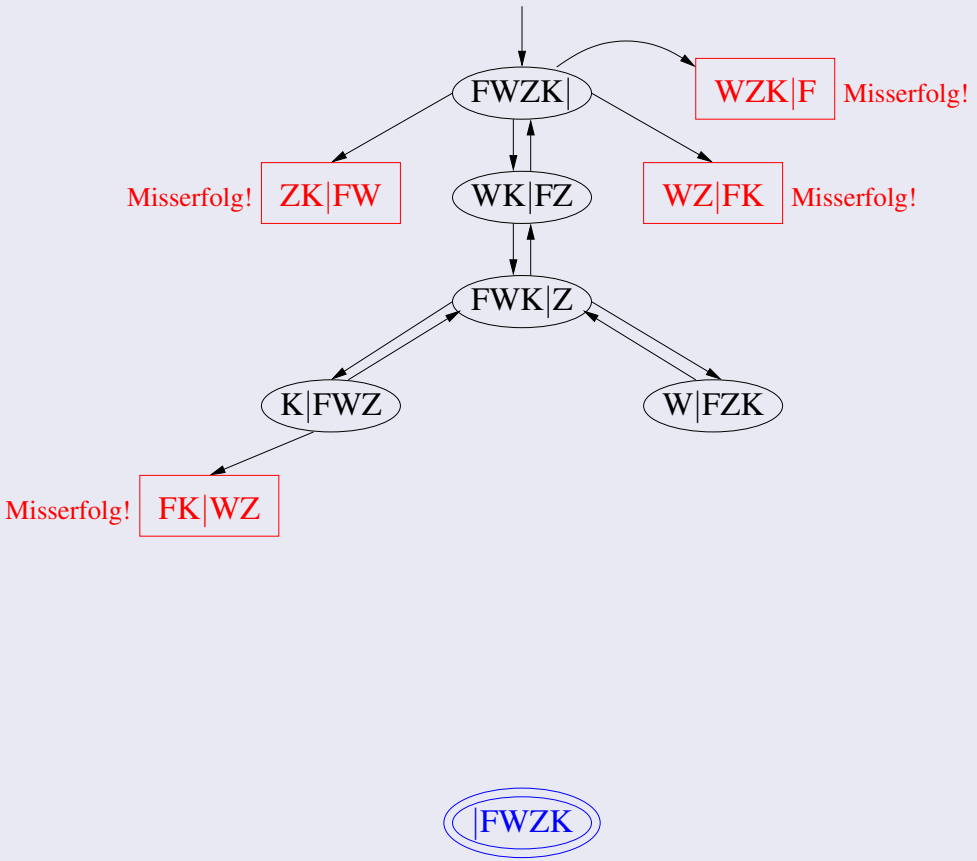
Zustandsübergangsdiagramme

Beispiel: Zustandsübergangsdiagramm für das Wolf-Ziege-Kohlkopf-Problem



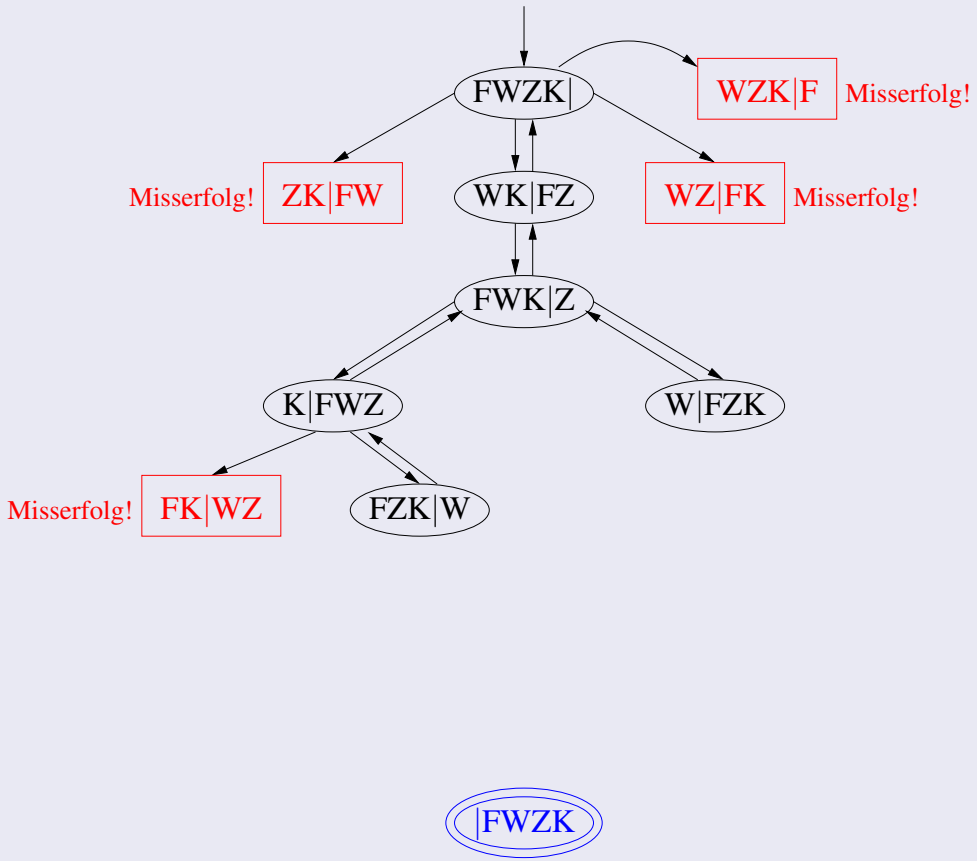
Zustandsübergangsdiagramme

Beispiel: Zustandsübergangsdiagramm für das Wolf-Ziege-Kohlkopf-Problem



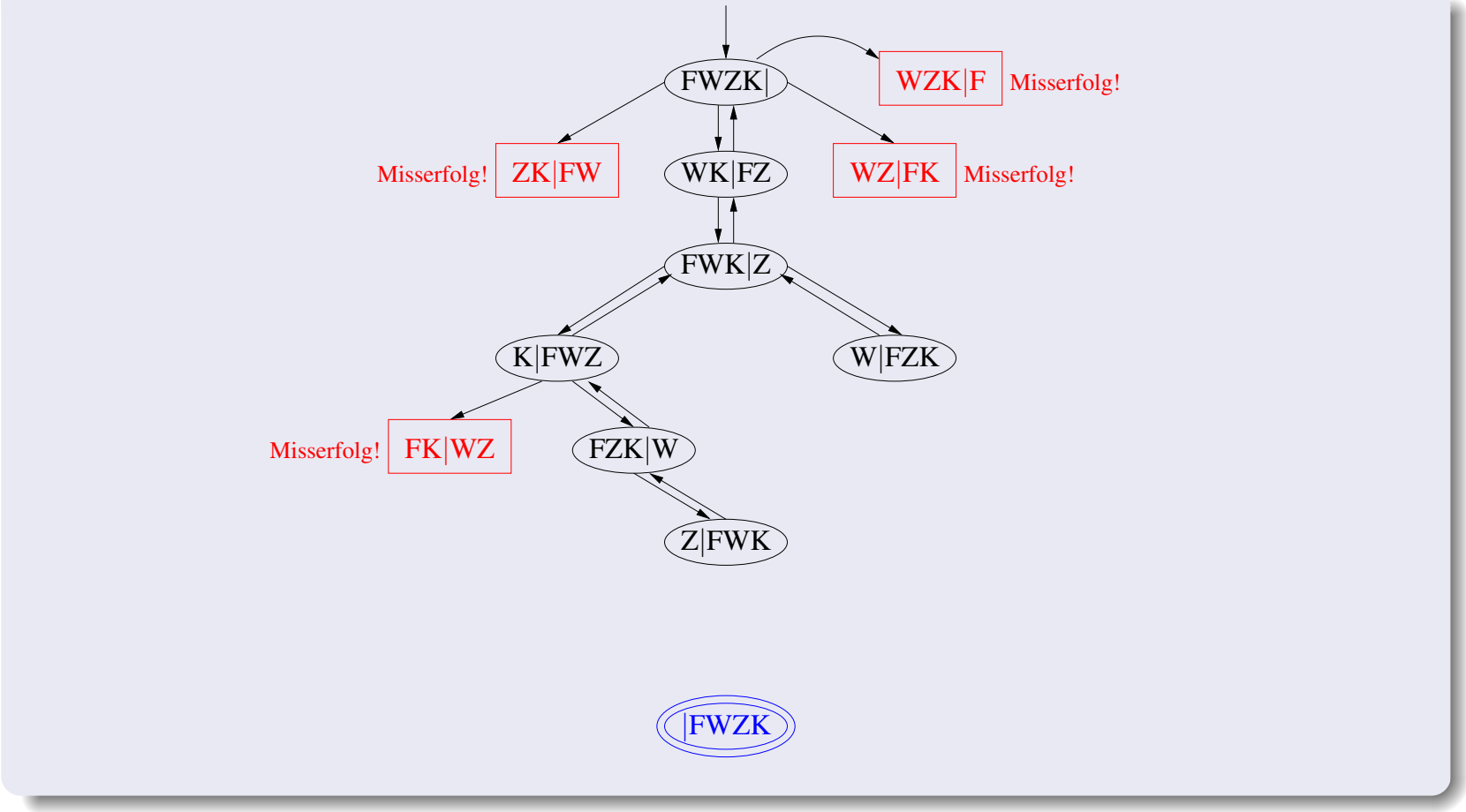
Zustandsübergangsdiagramme

Beispiel: Zustandsübergangsdiagramm für das Wolf-Ziege-Kohlkopf-Problem



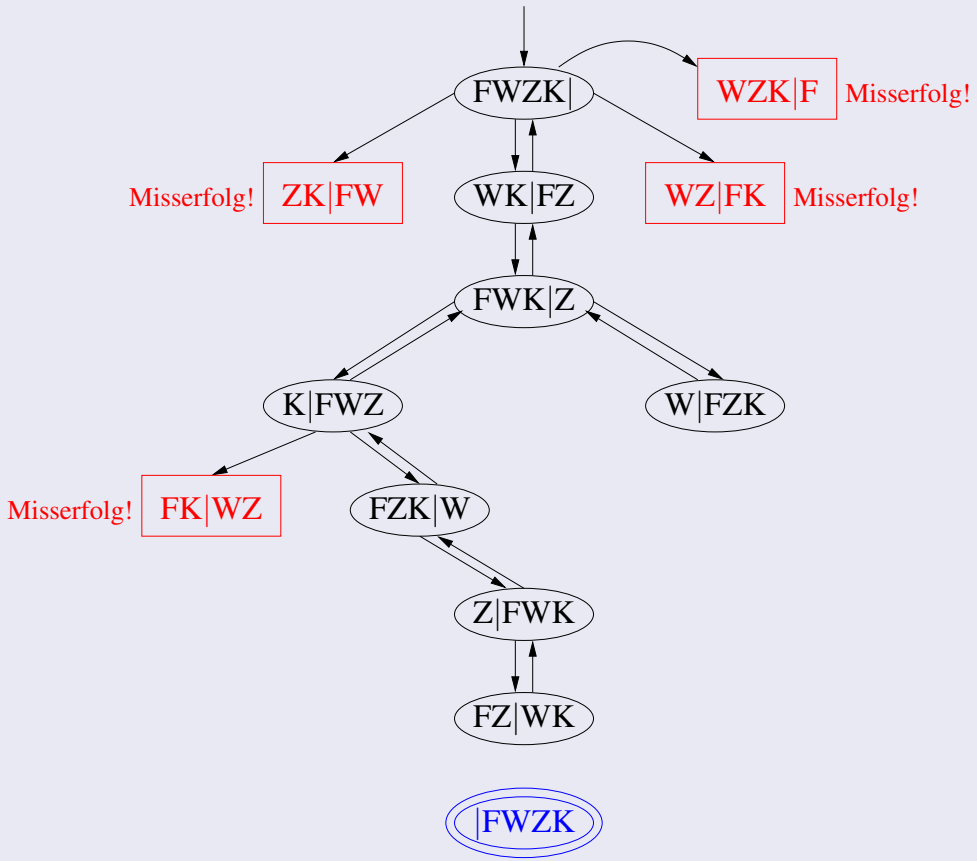
Zustandsübergangsdiagramme

Beispiel: Zustandsübergangsdiagramm für das Wolf-Ziege-Kohlkopf-Problem



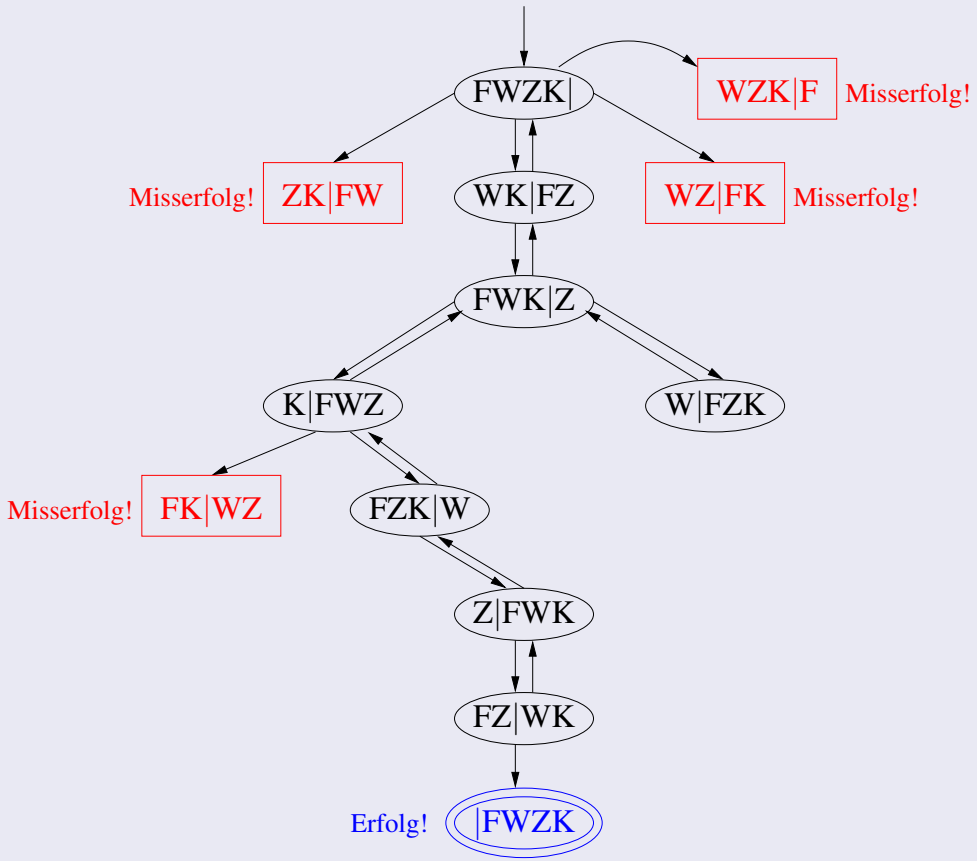
Zustandsübergangsdiagramme

Beispiel: Zustandsübergangsdiagramm für das Wolf-Ziege-Kohlkopf-Problem



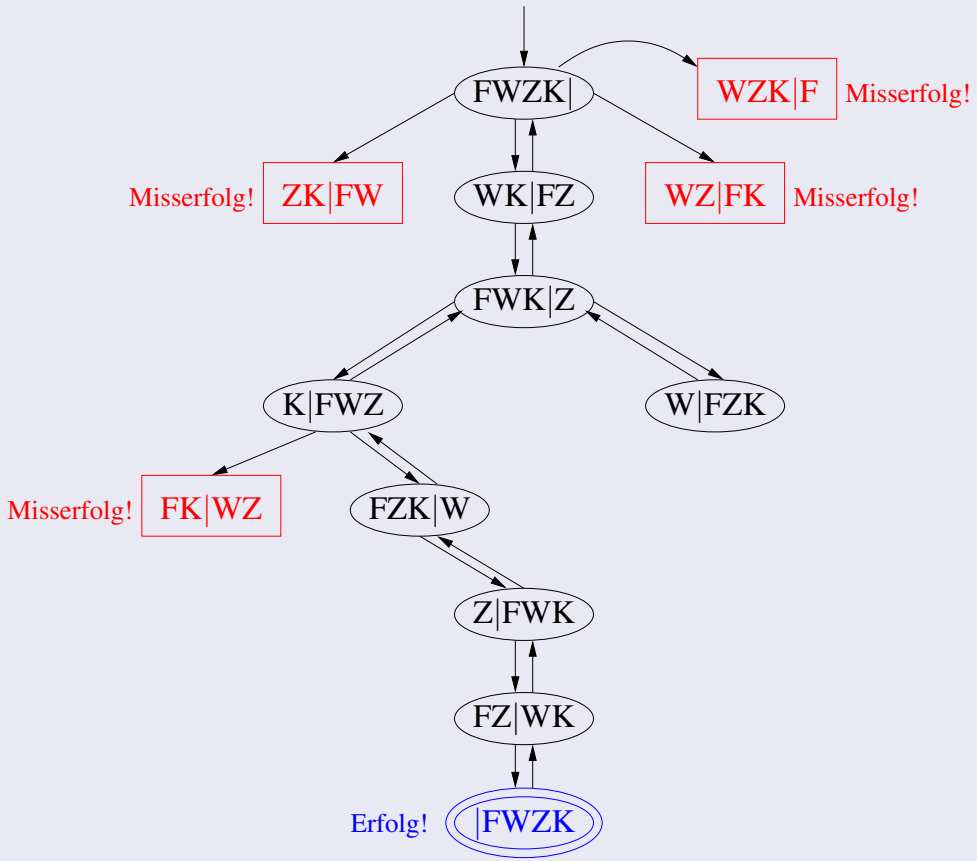
Zustandsübergangsdiagramme

Beispiel: Zustandsübergangsdiagramm für das Wolf-Ziege-Kohlkopf-Problem



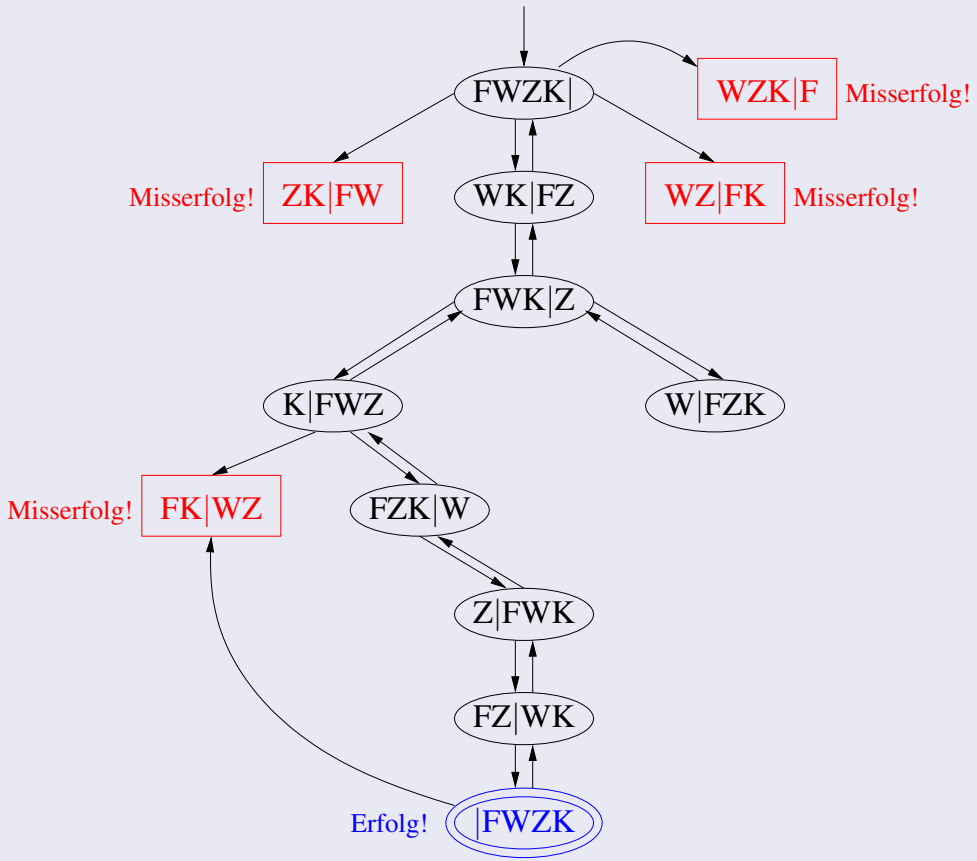
Zustandsübergangsdiagramme

Beispiel: Zustandsübergangsdiagramm für das Wolf-Ziege-Kohlkopf-Problem



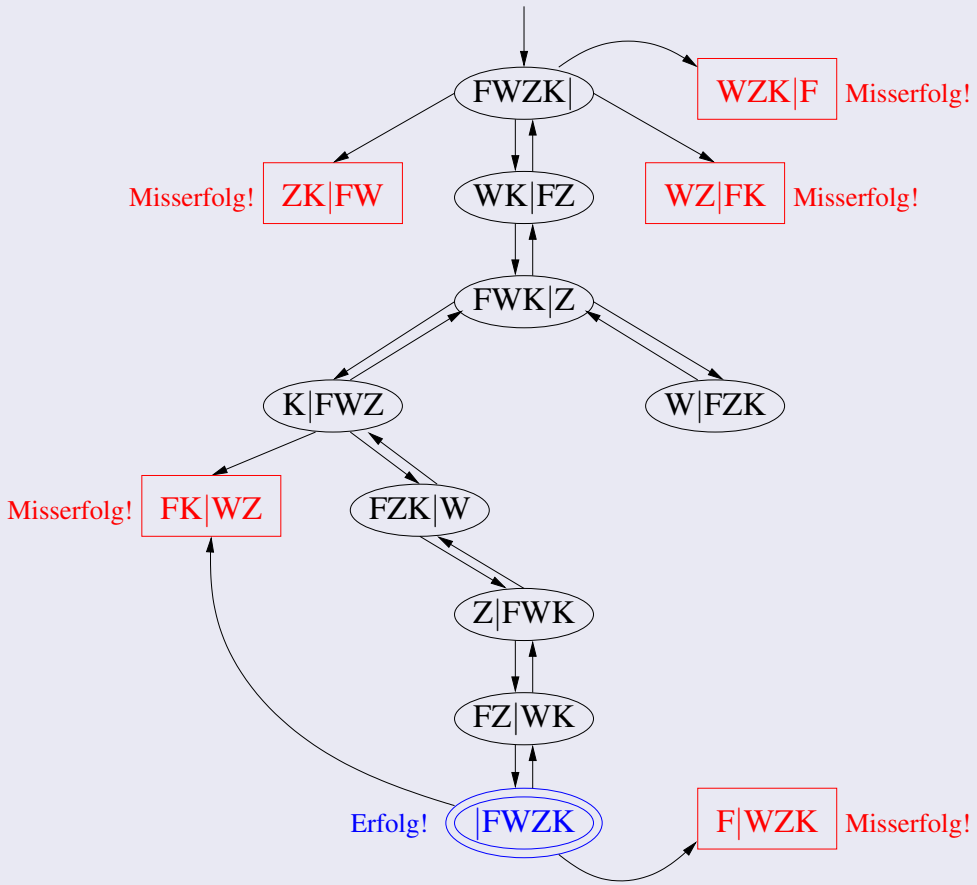
Zustandsübergangsdiagramme

Beispiel: Zustandsübergangsdiagramm für das Wolf-Ziege-Kohlkopf-Problem



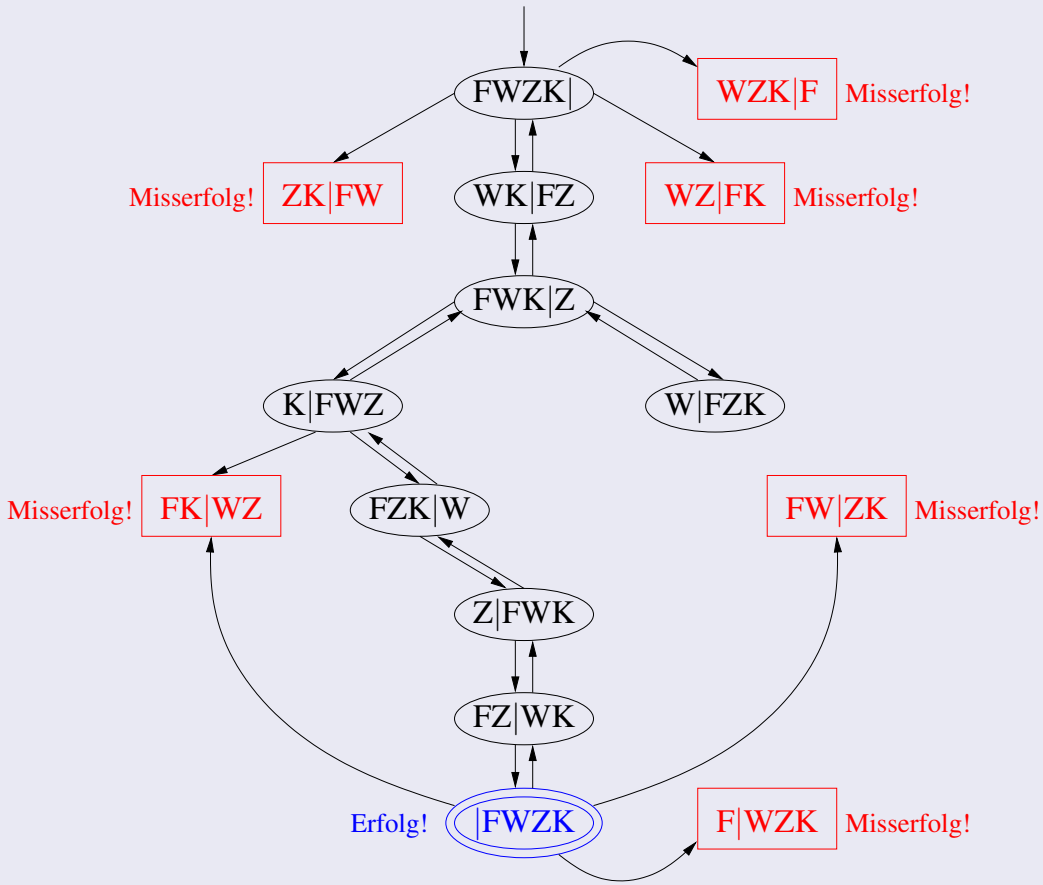
Zustandsübergangsdiagramme

Beispiel: Zustandsübergangsdiagramm für das Wolf-Ziege-Kohlkopf-Problem



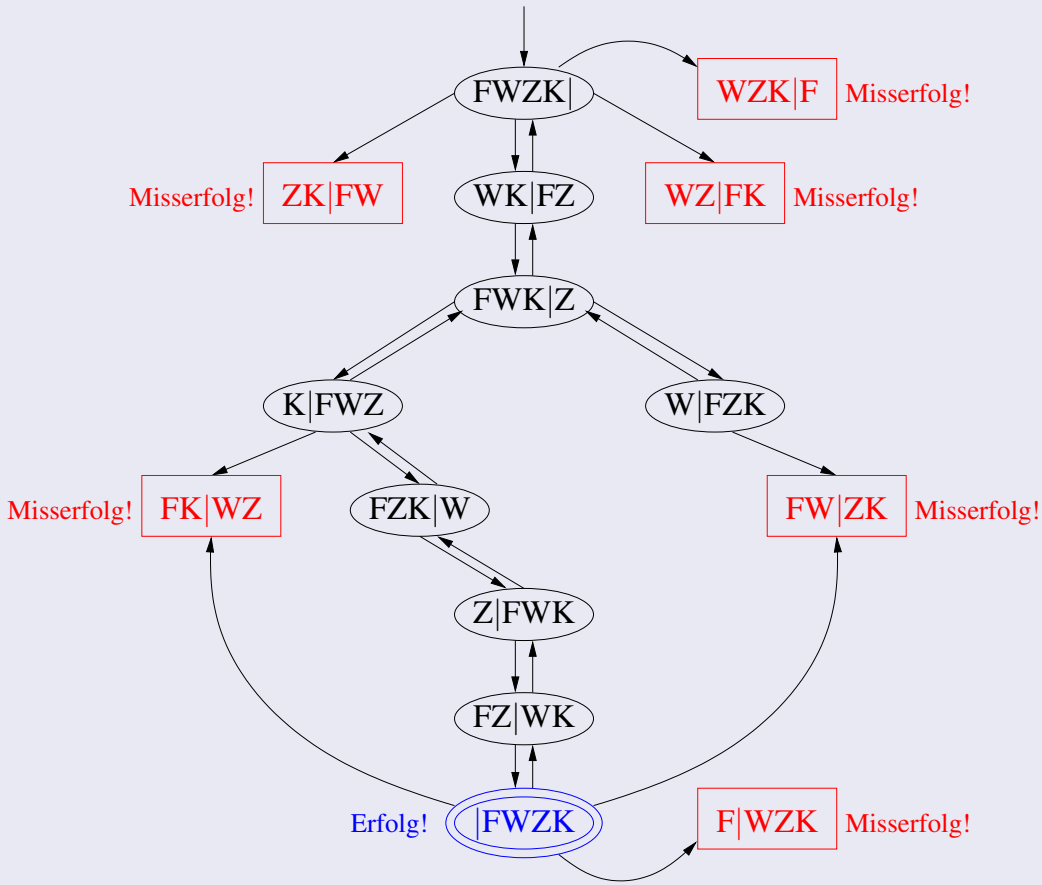
Zustandsübergangsdiagramme

Beispiel: Zustandsübergangsdiagramm für das Wolf-Ziege-Kohlkopf-Problem



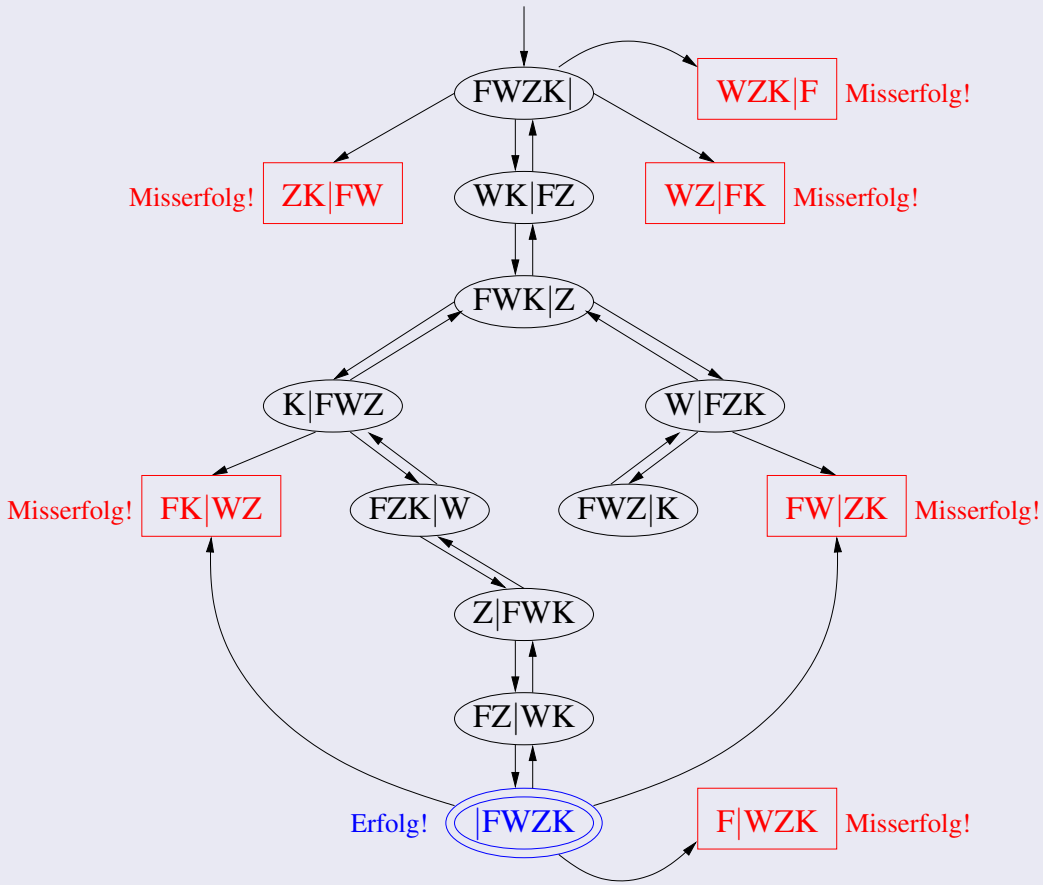
Zustandsübergangsdiagramme

Beispiel: Zustandsübergangsdiagramm für das Wolf-Ziege-Kohlkopf-Problem



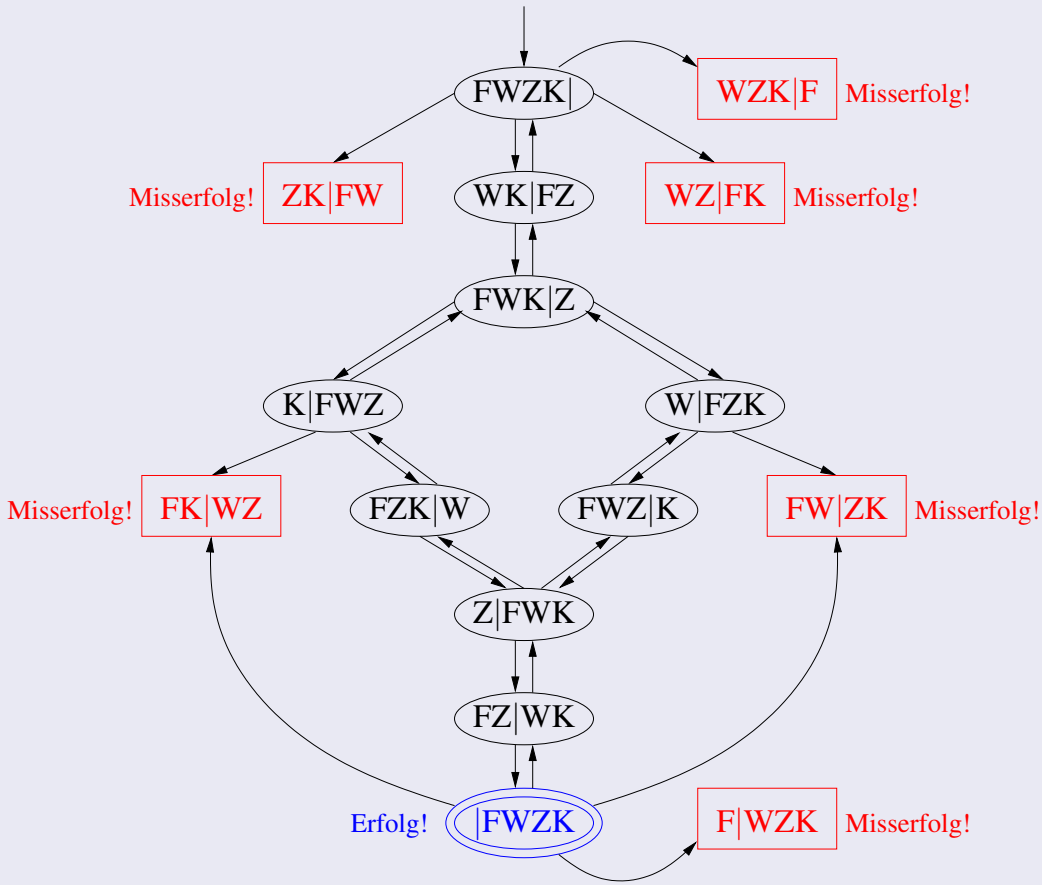
Zustandsübergangsdiagramme

Beispiel: Zustandsübergangsdiagramm für das Wolf-Ziege-Kohlkopf-Problem



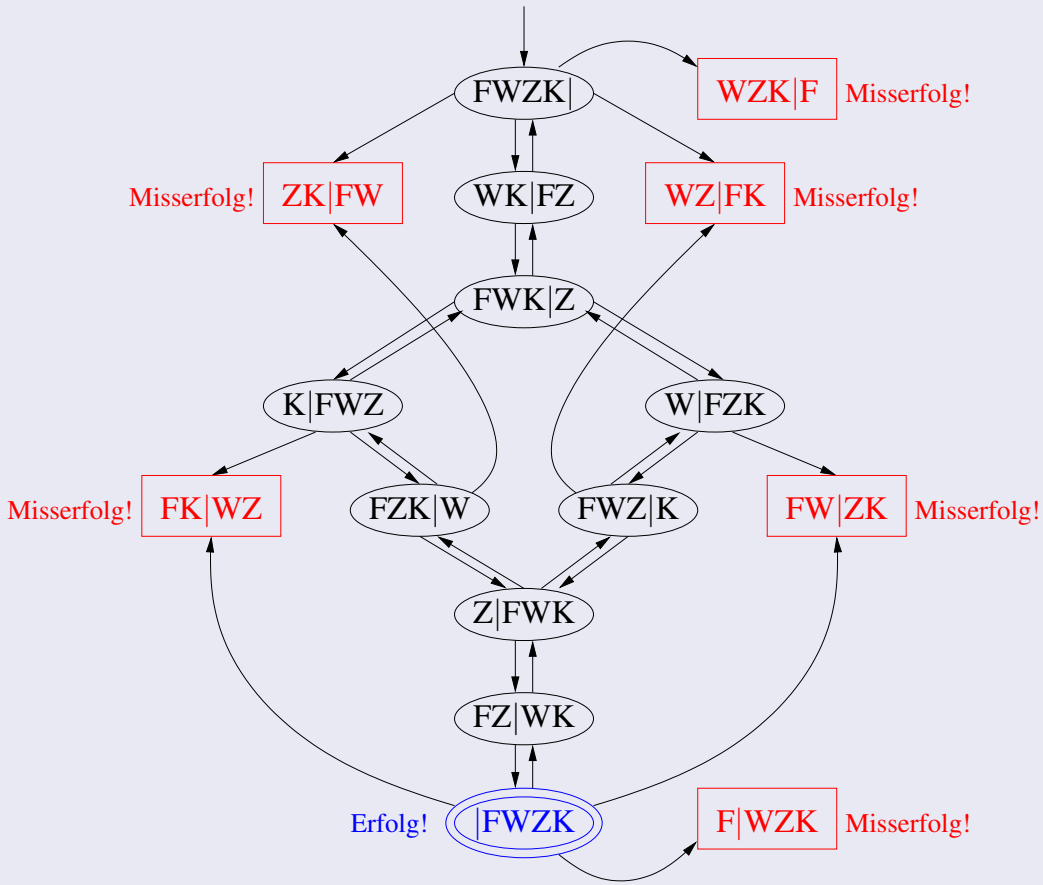
Zustandsübergangsdiagramme

Beispiel: Zustandsübergangsdiagramm für das Wolf-Ziege-Kohlkopf-Problem



Zustandsübergangsdiagramme

Beispiel: Zustandsübergangsdiagramm für das Wolf-Ziege-Kohlkopf-Problem



Zustandsübergangsdiagramme

Bemerkungen:

- Der senkrechte Strich | steht hier für den Fluss. Links und rechts davon befinden sich die Akteure/Objekte (F = Farmer, W = Wolf, Z = Ziege, K = Kohlkopf). Deren Reihenfolge ist egal, und das Boot braucht man nicht anzugeben, denn es ist immer beim Farmer.

Zustandsübergangsdiagramme

Bemerkungen:

- Der senkrechte Strich | steht hier für den Fluss. Links und rechts davon befinden sich die Akteure/Objekte (F = Farmer, W = Wolf, Z = Ziege, K = Kohlkopf). Deren Reihenfolge ist egal, und das Boot braucht man nicht anzugeben, denn es ist immer beim Farmer.
- Übergänge sind hier aus Gründen der Übersichtlichkeit nicht beschriftet. Sinnvolle Beschriftungen wären die ausgeführten Aktionen (z.B. „Farmer bringt Ziege über den Fluss“).

Zustandsübergangsdiagramme

Bemerkungen:

- Der senkrechte Strich | steht hier für den Fluss. Links und rechts davon befinden sich die Akteure/Objekte (F = Farmer, W = Wolf, Z = Ziege, K = Kohlkopf). Deren Reihenfolge ist egal, und das Boot braucht man nicht anzugeben, denn es ist immer beim Farmer.
- Übergänge sind hier aus Gründen der Übersichtlichkeit nicht beschriftet. Sinnvolle Beschriftungen wären die ausgeführten Aktionen (z.B. „Farmer bringt Ziege über den Fluss“).
- Eckige (rote) Zustände symbolisieren hier Misserfolg (z.B. „Ziege frisst Kohlkopf“; solche Aktionen haben Priorität vor Überfahrten). Kanten, die aus solchen Zuständen herausführen, wurden hier weggelassen.

Zustandsübergangsdiagramme

Bemerkungen:

- Der senkrechte Strich | steht hier für den Fluss. Links und rechts davon befinden sich die Akteure/Objekte (F = Farmer, W = Wolf, Z = Ziege, K = Kohlkopf). Deren Reihenfolge ist egal, und das Boot braucht man nicht anzugeben, denn es ist immer beim Farmer.
- Übergänge sind hier aus Gründen der Übersichtlichkeit nicht beschriftet. Sinnvolle Beschriftungen wären die ausgeführten Aktionen (z.B. „Farmer bringt Ziege über den Fluss“).
- Eckige (rote) Zustände symbolisieren hier Misserfolg (z.B. „Ziege frisst Kohlkopf“; solche Aktionen haben Priorität vor Überfahrten). Kanten, die aus solchen Zuständen herausführen, wurden hier weggelassen.
- Die doppelte (blaue) Ellipse symbolisiert hier Erfolg (erwünschter Zielzustand ist erreicht).

Zustandsübergangsdiagramme

Weitere Bemerkungen:

- Es gibt mehrere (sogar unendlich viele und beliebig lange) Wege zum Zielzustand. Die zwei kürzesten enthalten jeweils sieben Übergänge.

Zustandsübergangsdiagramme

Weitere Bemerkungen:

- Es gibt mehrere (sogar unendlich viele und beliebig lange) Wege zum Zielzustand. Die zwei kürzesten enthalten jeweils sieben Übergänge.
- Zustandsübergangsdiagramme für selbst relativ einfache Systeme werden oft erstaunlich groß (sogenannte Zustandsexplosion).

Wichtig:

Zustandsübergangsdiagramme stellen im Allgemeinen alle Zustände und alle Übergänge eines Systems dar.

Petrinetze: Grundlagen und Erreichbarkeitsgraphen

Motivation: Petrinetze

Petrinetze sind ein Formalismus zur Modellierung mit folgenden Eigenschaften:

- Vorstellung von Systemübergängen, bei denen (gemeinsame) Ressourcen konsumiert und neu erzeugt werden können.

Motivation: Petrinetze

Petrinetze sind ein Formalismus zur Modellierung mit folgenden Eigenschaften:

- Vorstellung von Systemübergängen, bei denen (gemeinsame) Ressourcen konsumiert und neu erzeugt werden können.
- Einfache Modellierung von Kapazitäten, räumlicher Verteilung der Ressourcen, von Nebenläufigkeit, Parallelität und (Zugriffs-)Konflikten.

Motivation: Petrinetze

Petrinetze sind ein Formalismus zur Modellierung mit folgenden Eigenschaften:

- Vorstellung von Systemübergängen, bei denen (gemeinsame) Ressourcen konsumiert und neu erzeugt werden können.
- Einfache Modellierung von Kapazitäten, räumlicher Verteilung der Ressourcen, von Nebenläufigkeit, Parallelität und (Zugriffs-)Konflikten.
- Intuitive grafische Darstellung.

Motivation: Petrinetze

Petrinetze sind ein Formalismus zur Modellierung mit folgenden Eigenschaften:

- Vorstellung von Systemübergängen, bei denen (gemeinsame) Ressourcen konsumiert und neu erzeugt werden können.
- Einfache Modellierung von Kapazitäten, räumlicher Verteilung der Ressourcen, von Nebenläufigkeit, Parallelität und (Zugriffs-)Konflikten.
- Intuitive grafische Darstellung.

Petrinetze werden in der Praxis vielfach benutzt.

In UML sind sie abgewandelt als sogenannte Aktivitätsdiagramme (englisch: activity diagrams) eingegangen.

Motivation: Petrinetze

Parallelität versus Nebenläufigkeit:

Parallelität

Zwei Ereignisse finden parallel statt, wenn sie gleichzeitig ausgeführt werden.

Motivation: Petrinetze

Parallelität versus Nebenläufigkeit:

Parallelität

Zwei Ereignisse finden parallel statt, wenn sie gleichzeitig ausgeführt werden.

Nebenläufigkeit

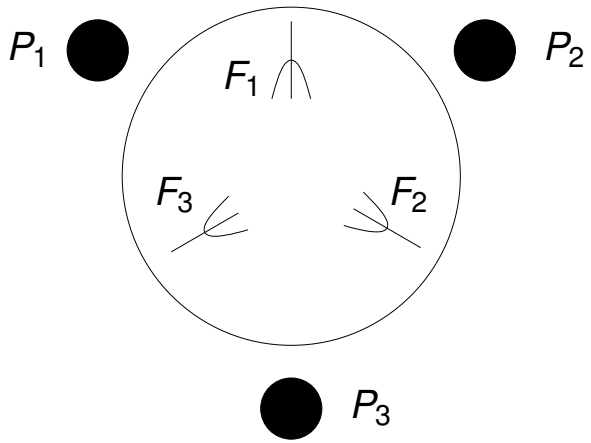
Zwei Ereignisse sind nebenläufig, wenn sie parallel ausgeführt werden können (jedoch nicht müssen), das heißt, wenn zwischen ihnen keine kausale Abhängigkeit besteht.

Das bedeutet: Nebenläufigkeit ist der allgemeinere Begriff.

Motivation: Petrinetze

Ein verbreitetes Beispiel: Dining Philosophers

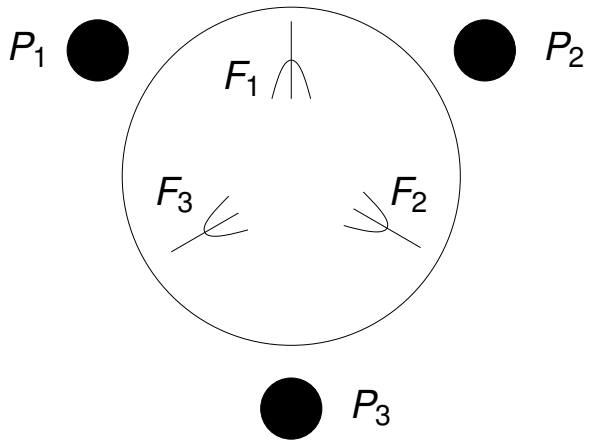
- Es sitzen drei (oder vier, oder fünf, . . .) Philosophen P_i um einen runden Tisch, zwischen je zwei Philosophen liegt eine Gabel (fork) F_i.



Motivation: Petrinetze

Ein verbreitetes Beispiel: Dining Philosophers

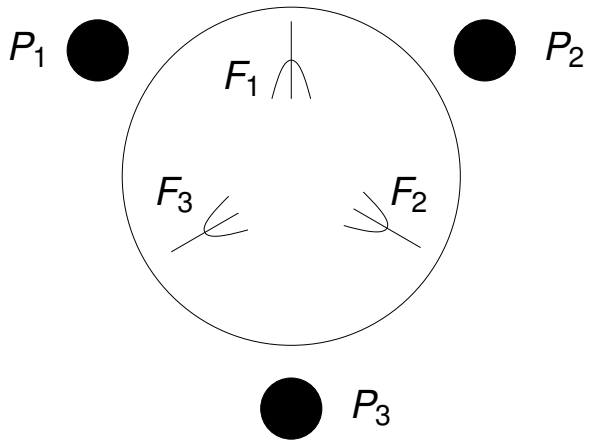
- Es sitzen drei (oder vier, oder fünf, . . .) Philosophen P_i um einen runden Tisch, zwischen je zwei Philosophen liegt eine Gabel (fork) F_i.
- Philosophen werden von Zeit zu Zeit hungrig und benötigen dann zum Essen beide benachbarte Gabeln.



Motivation: Petrinetze

Ein verbreitetes Beispiel: Dining Philosophers

- Es sitzen drei (oder vier, oder fünf, . . .) Philosophen P_i um einen runden Tisch, zwischen je zwei Philosophen liegt eine Gabel (fork) F_i.
- Philosophen werden von Zeit zu Zeit hungrig und benötigen dann zum Essen beide benachbarte Gabeln.
- Jeder Philosoph nimmt zu einem beliebigen Zeitpunkt beide Gabeln nacheinander auf (die rechte zuerst), isst und legt anschließend beide Gabeln wieder zurück.



Motivation: Petrinetze

Fragen, die man (zum Beispiel) mit Petrinetzen untersuchen kann:

- Kann das modellierte System kontinuierlich Fortschritt machen?

Motivation: Petrinetze

Fragen, die man (zum Beispiel) mit Petrinetzen untersuchen kann:

- Kann das modellierte System kontinuierlich Fortschritt machen, oder lässt es sich in Sackgassen manövrieren?

Motivation: Petrinetze

Fragen, die man (zum Beispiel) mit Petrinetzen untersuchen kann:

- Kann das modellierte System kontinuierlich Fortschritt machen, oder lässt es sich in Sackgassen manövrieren?
- Bekommt jede modellierte Aktion die Chance, auch tatsächlich ausgeführt zu werden?

Motivation: Petrinetze

Fragen, die man (zum Beispiel) mit Petrinetzen untersuchen kann:

- Kann das modellierte System kontinuierlich Fortschritt machen, oder lässt es sich in Sackgassen manövrieren?
- Bekommt jede modellierte Aktion die Chance, auch tatsächlich ausgeführt zu werden?
  Nur einmalig oder sogar beliebig oft wiederholt?

Motivation: Petrinetze

Fragen, die man (zum Beispiel) mit Petrinetzen untersuchen kann:

- Kann das modellierte System kontinuierlich Fortschritt machen, oder lässt es sich in Sackgassen manövrieren?
- Bekommt jede modellierte Aktion die Chance, auch tatsächlich ausgeführt zu werden?
  Nur einmalig oder sogar beliebig oft wiederholt?
- Besteht Fairness für verschiedene Akteure?

Motivation: Petrinetze

Fragen, die man (zum Beispiel) mit Petrinetzen untersuchen kann:

- Kann das modellierte System kontinuierlich Fortschritt machen, oder lässt es sich in Sackgassen manövrieren?
- Bekommt jede modellierte Aktion die Chance, auch tatsächlich ausgeführt zu werden?
  Nur einmalig oder sogar beliebig oft wiederholt?
- Besteht Fairness für verschiedene Akteure?
- Bedingen bestimmte Aktionen einander

Motivation: Petrinetze

Fragen, die man (zum Beispiel) mit Petrinetzen untersuchen kann:

- Kann das modellierte System kontinuierlich Fortschritt machen, oder lässt es sich in Sackgassen manövrieren?
- Bekommt jede modellierte Aktion die Chance, auch tatsächlich ausgeführt zu werden?
  Nur einmalig oder sogar beliebig oft wiederholt?
- Besteht Fairness für verschiedene Akteure?
- Bedingen bestimmte Aktionen einander, oder schließen sich gegenseitig aus?

Motivation: Petrinetze

Fragen, die man (zum Beispiel) mit Petrinetzen untersuchen kann:

- Kann das modellierte System kontinuierlich Fortschritt machen, oder lässt es sich in Sackgassen manövrieren?
- Bekommt jede modellierte Aktion die Chance, auch tatsächlich ausgeführt zu werden?
  Nur einmalig oder sogar beliebig oft wiederholt?
- Besteht Fairness für verschiedene Akteure?
- Bedingen bestimmte Aktionen einander, oder schließen sich gegenseitig aus?
- Gibt es Beschränkungen für eventuellen Ressourcenverbrauch und -erzeugung?

Motivation: Petrinetze

Fragen, die man (zum Beispiel) mit Petrinetzen untersuchen kann:

- Kann das modellierte System kontinuierlich Fortschritt machen, oder lässt es sich in Sackgassen manövrieren?
- Bekommt jede modellierte Aktion die Chance, auch tatsächlich ausgeführt zu werden?
  Nur einmalig oder sogar beliebig oft wiederholt?
- Besteht Fairness für verschiedene Akteure?
- Bedingen bestimmte Aktionen einander, oder schließen sich gegenseitig aus?
- Gibt es Beschränkungen für eventuellen Ressourcenverbrauch und -erzeugung?

Dabei sind bestimmte Analysen sogar für Systeme möglich, deren Zustandsübergangsdiagramm unendlich wäre.

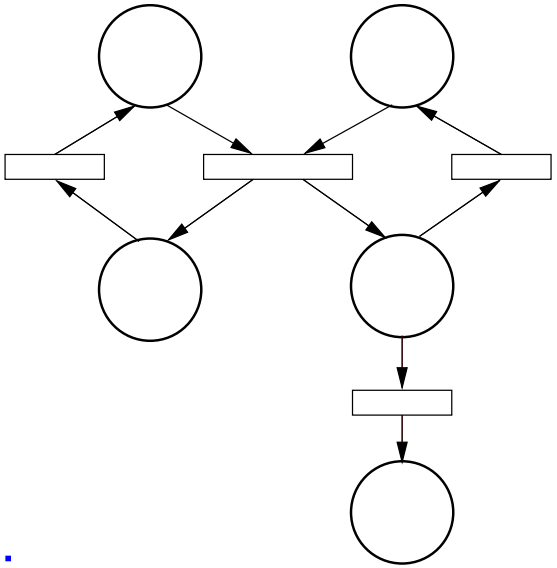
Motivation: Petrinetze

Praktische Anwendungen für Petrinetze:

- Modellierung von Arbeitsabläufen
  (work flow, business processes)
- Modellierung und Analyse von Web Services
- Beschreibung von grafischen Benutzungsoberflächen
- Prozessmodellierung bei Betriebssystemen
- Ablaufbeschreibungen in ingenieurwissenschaftlichen Anwendungen
- ...

Petrinetze: Informelle Einführung

Beispiel für ein Petrinetz:

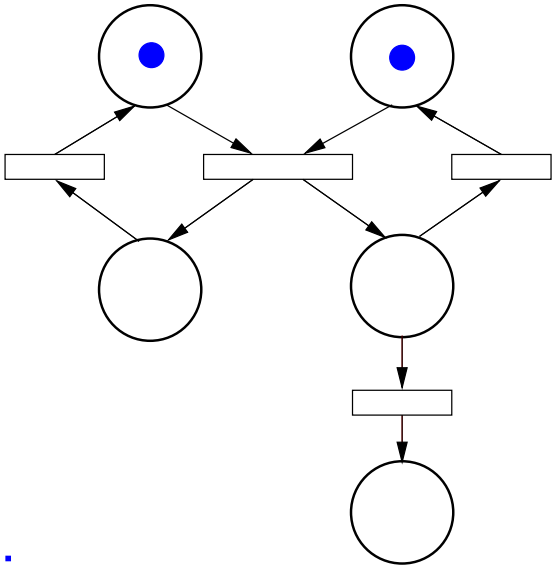


Grafische Darstellung:

- Stellen (dargestellt als Kreise):
  Mögliche Plätze für Ressourcen

Petrinetze: Informelle Einführung

Beispiel für ein Petrinetz:

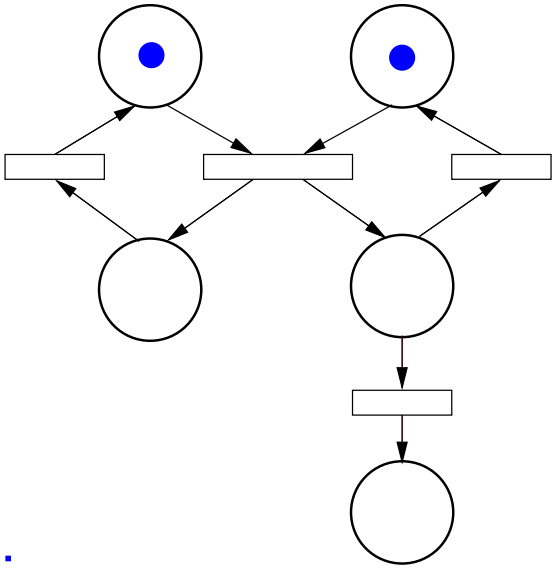


Grafische Darstellung:

- Stellen (dargestellt als Kreise):
  Mögliche Plätze für Ressourcen
- Marken (dargestellt als kleine ausgefüllte Kreise):
  Ressourcen

Petrinetze: Informelle Einführung

Beispiel für ein Petrinetz:

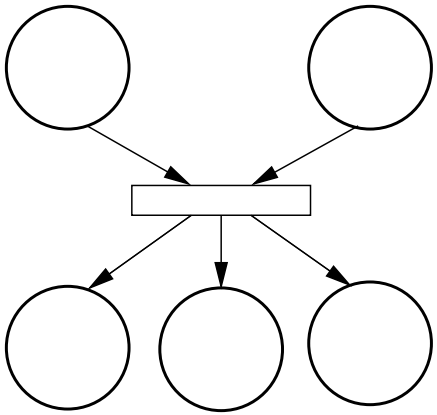


Grafische Darstellung:

- Stellen (dargestellt als Kreise):
  Mögliche Plätze für Ressourcen
- Marken (dargestellt als kleine ausgefüllte Kreise):
  Ressourcen
- Transitionen (dargestellt durch Rechtecke):
  Systemübergänge

Petrinetze: Informelle Einführung

Mehr zur Darstellung einer Transition:

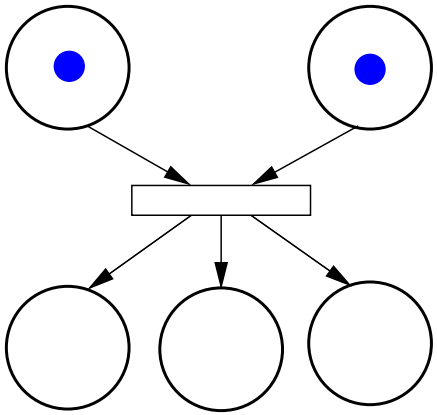


Vorbedingung (Marken, die konsumiert werden)

Nachbedingung (Marken, die erzeugt werden)

Petrinetze: Informelle Einführung

Mehr zur Darstellung einer Transition:

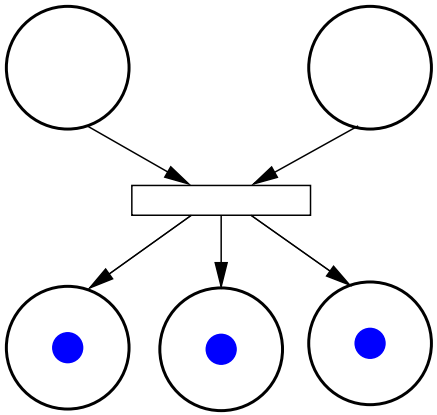


Vorbedingung (Marken, die konsumiert werden)

Nachbedingung (Marken, die erzeugt werden)

Petrinetze: Informelle Einführung

Mehr zur Darstellung einer Transition:

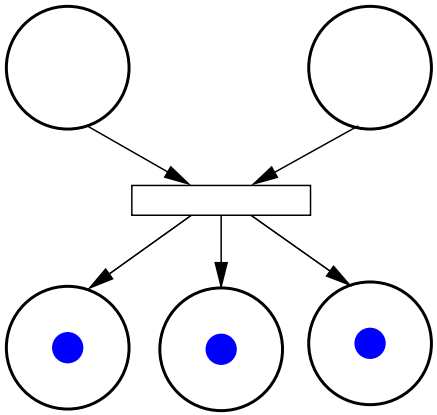


Vorbedingung (Marken, die konsumiert werden)

Nachbedingung (Marken, die erzeugt werden)

Petrinetze: Informelle Einführung

Mehr zur Darstellung einer Transition:



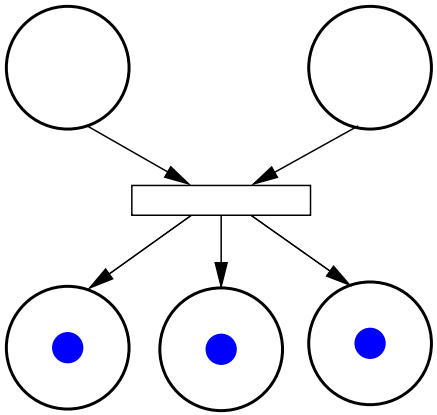
Vorbedingung (Marken, die konsumiert werden)

Nachbedingung (Marken, die erzeugt werden)

Das Entfernen der Marken der Vorbedingung und Erzeugen der Marken der Nachbedingung nennt man Schalten bzw. Feuern der Transition.

Petrinetze: Informelle Einführung

Mehr zur Darstellung einer Transition:



Vorbedingung (Marken, die konsumiert werden)

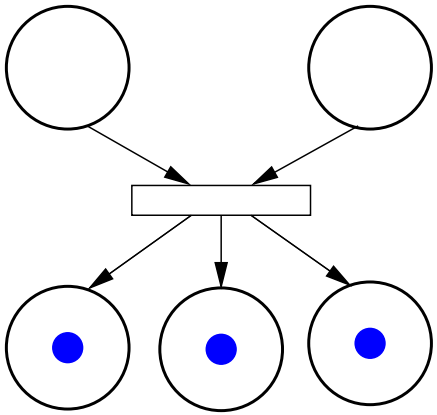
Nachbedingung (Marken, die erzeugt werden)

Das Entfernen der Marken der Vorbedingung und Erzeugen der Marken der Nachbedingung nennt man Schalten bzw. Feuern der Transition.

Allgemeiner als im Beispiel oben muss nicht unbedingt genau eine Marke pro Pfeil konsumiert oder erzeugt werden.

Petrinetze: Informelle Einführung

Mehr zur Darstellung einer Transition:



Vorbedingung (Marken, die konsumiert werden)

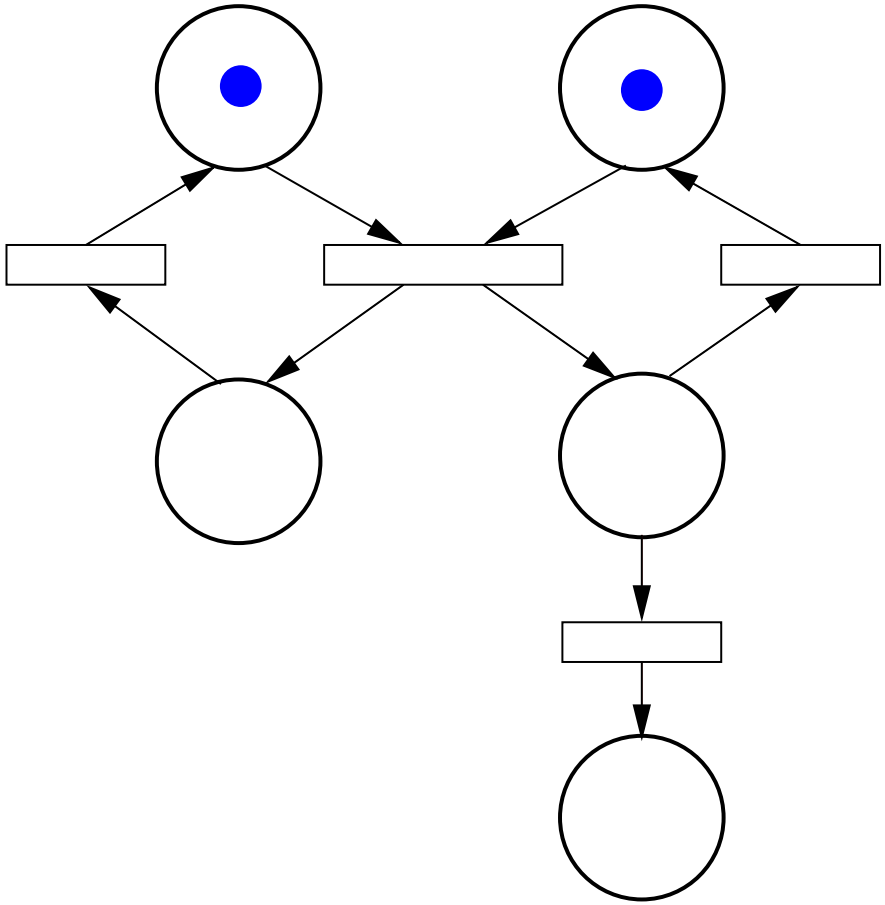
Nachbedingung (Marken, die erzeugt werden)

Das Entfernen der Marken der Vorbedingung und Erzeugen der Marken der Nachbedingung nennt man Schalten bzw. Feuern der Transition.

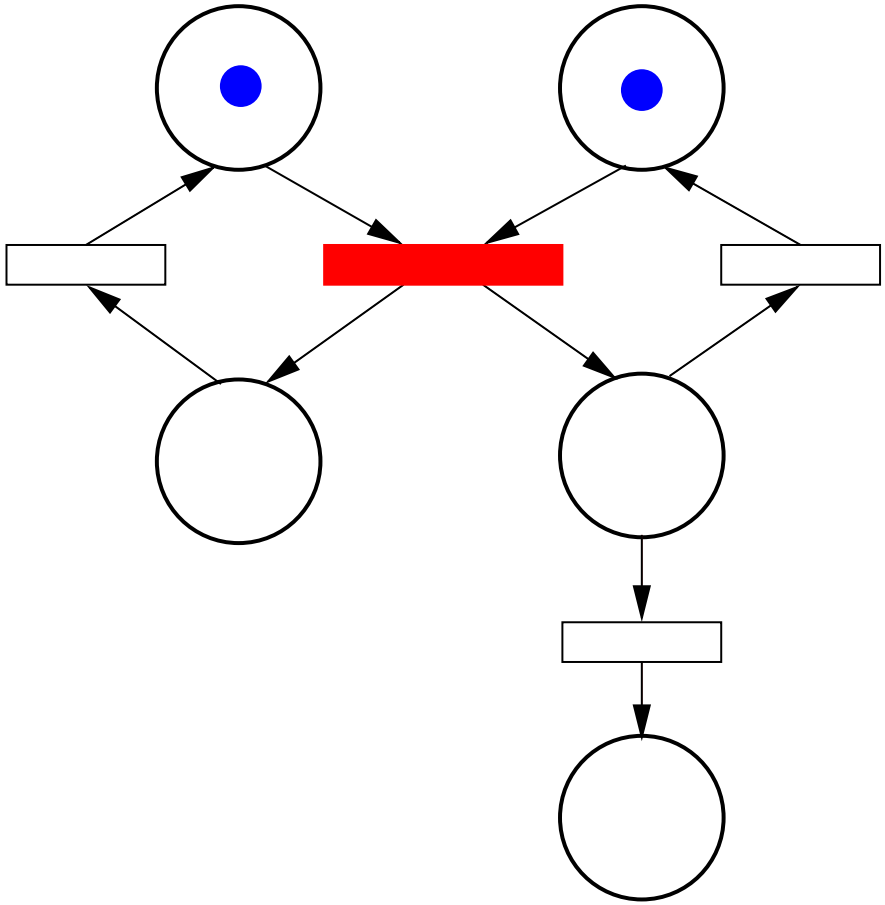
Allgemeiner als im Beispiel oben muss nicht unbedingt genau eine Marke pro Pfeil konsumiert oder erzeugt werden.

Und weder müssen die Stellen der Nachbedingung vor dem Schalten leer sein, noch die Stellen der Vorbedingung danach.

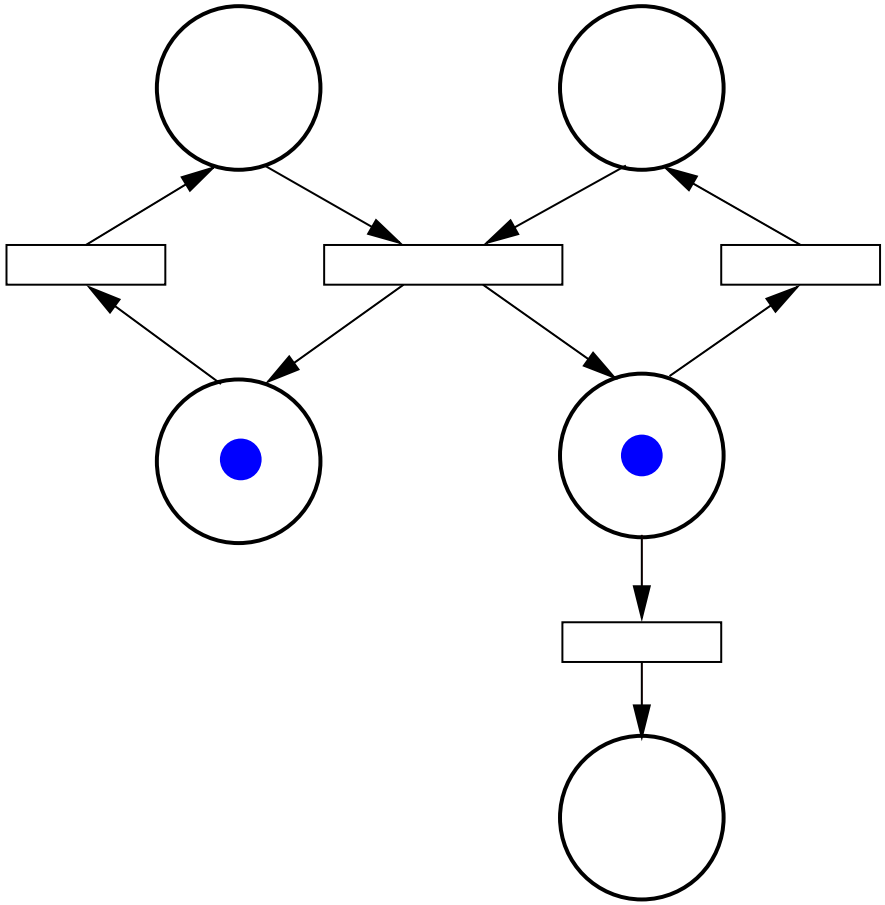
Petrinetze: Informelle Einführung



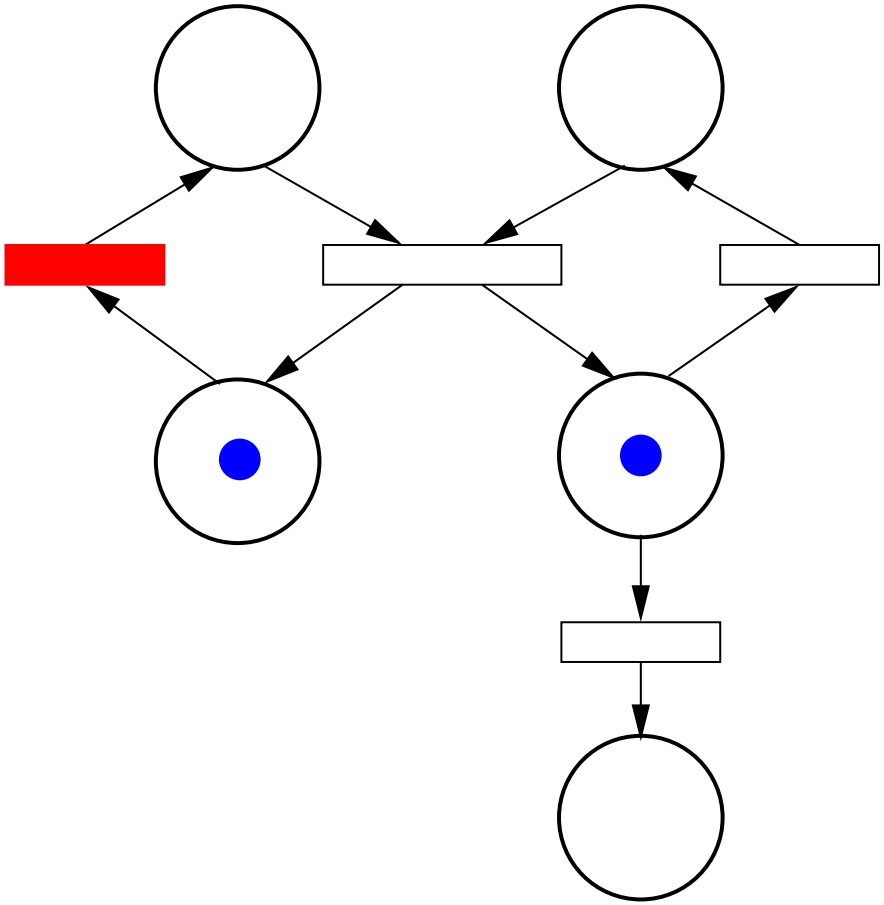
Petrinetze: Informelle Einführung



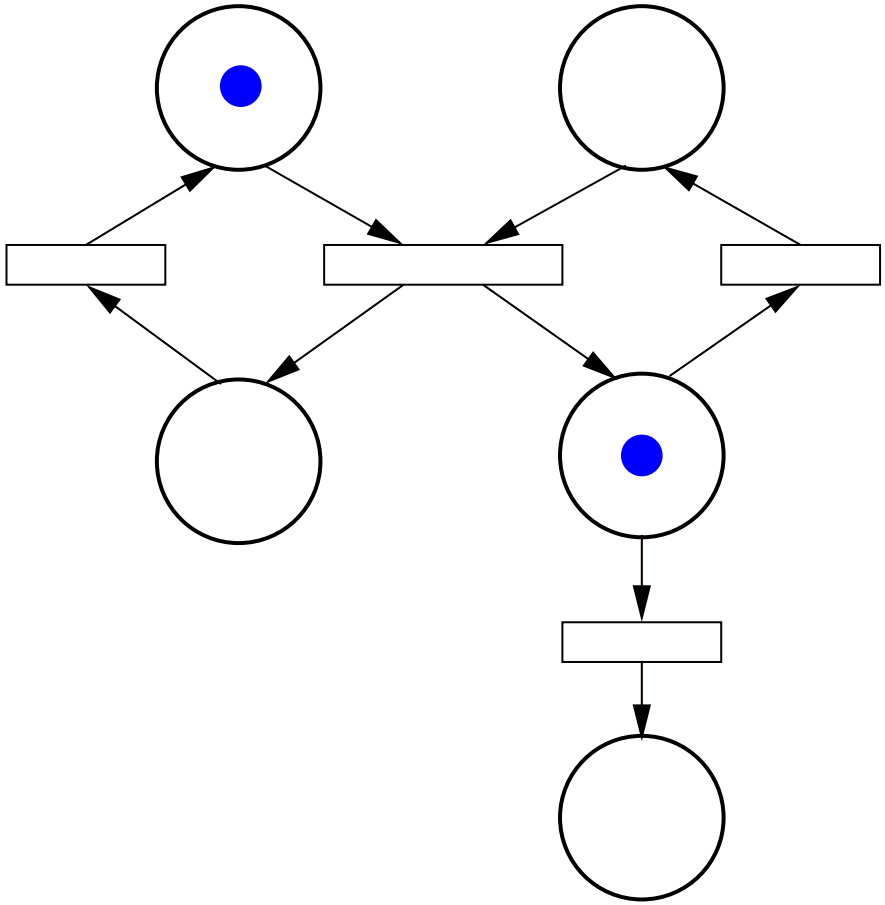
Petrinetze: Informelle Einführung



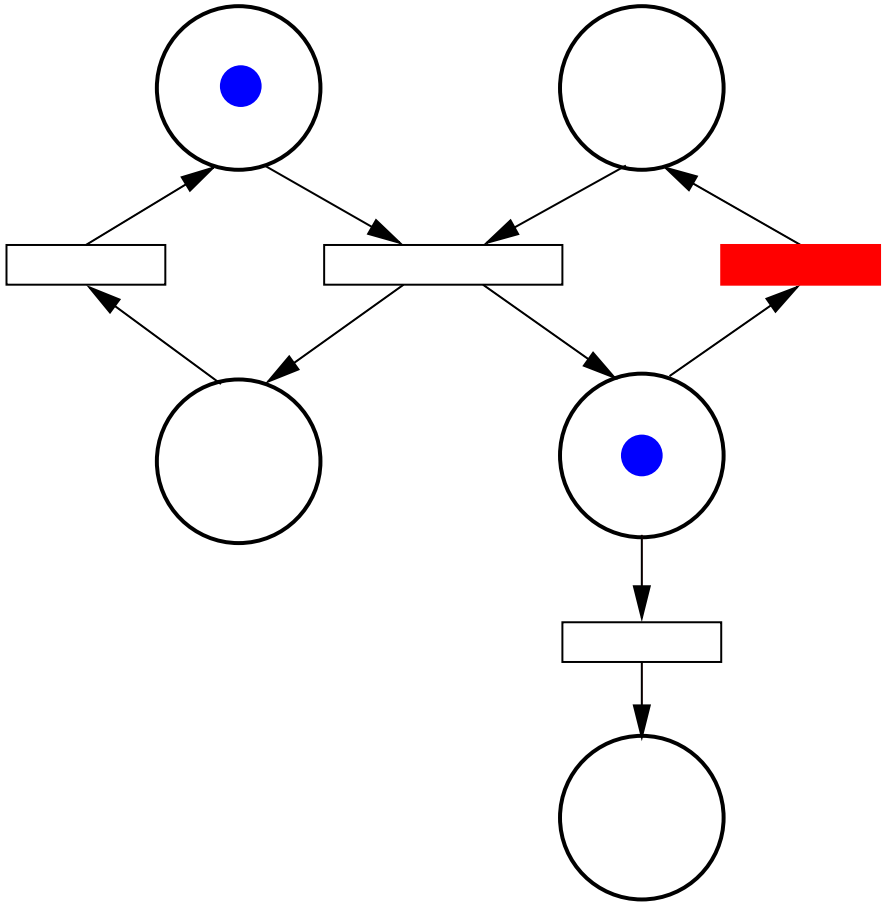
Petrinetze: Informelle Einführung



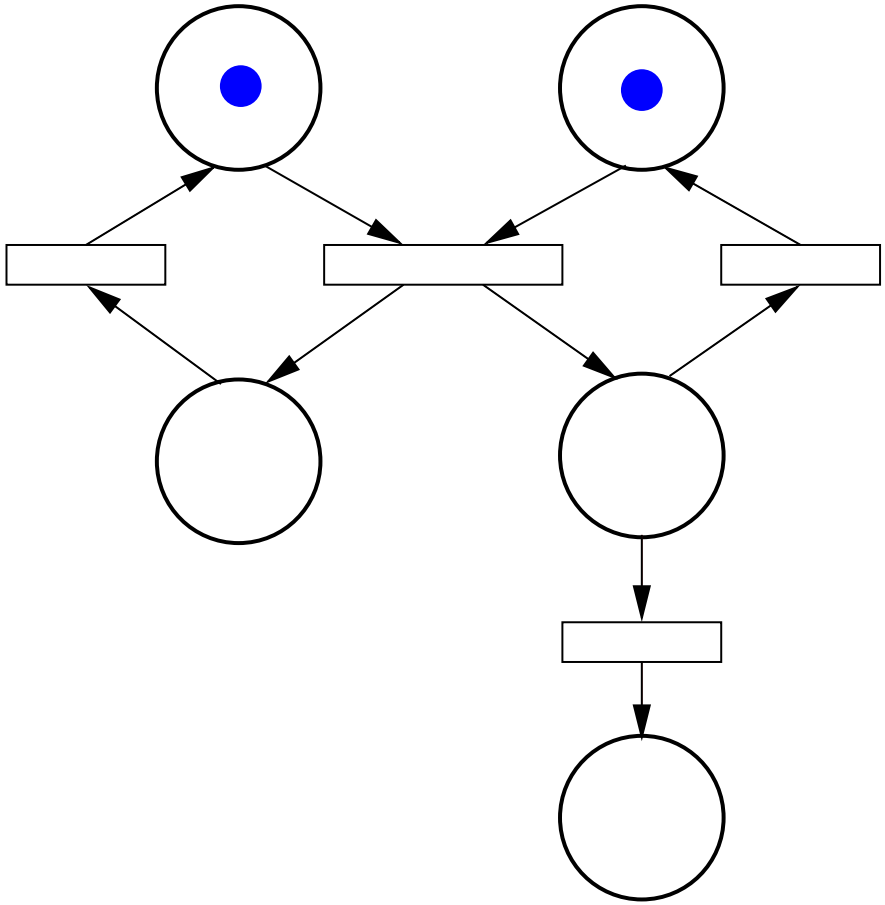
Petrinetze: Informelle Einführung



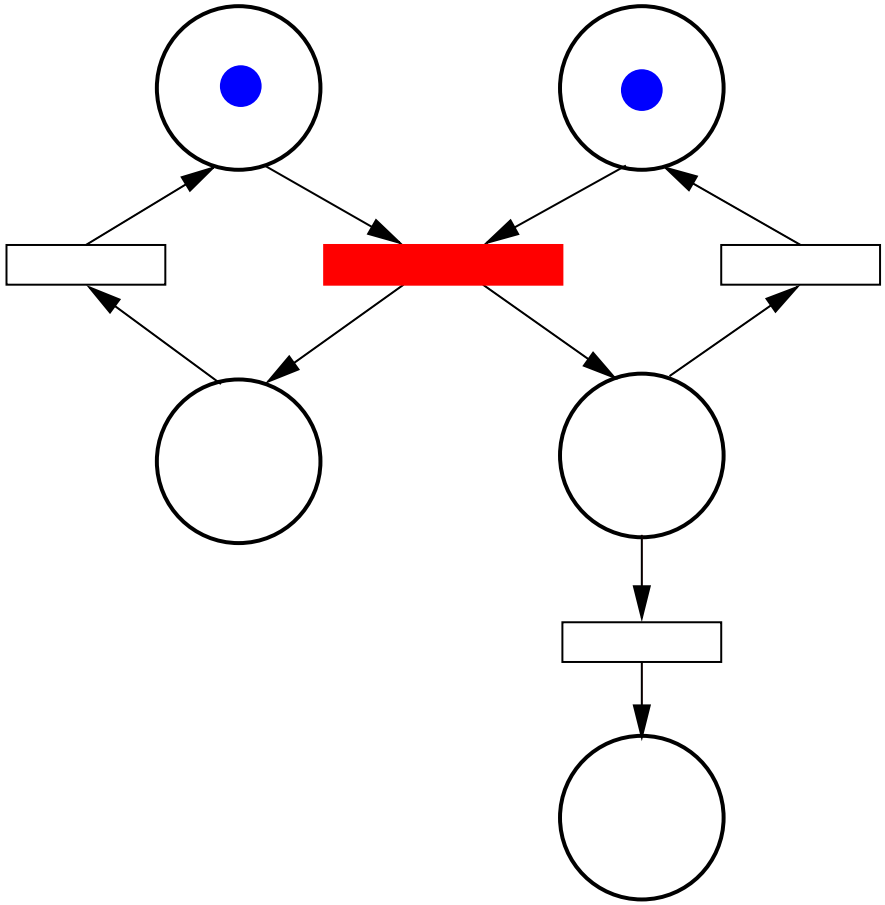
Petrinetze: Informelle Einführung



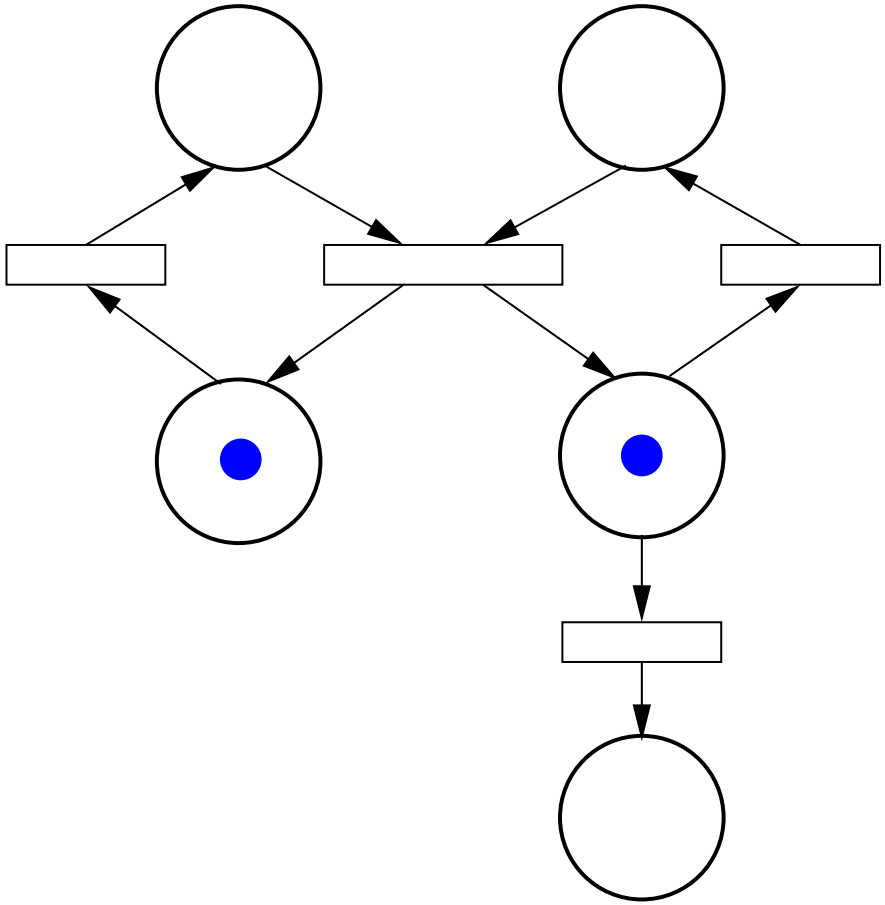
Petrinetze: Informelle Einführung



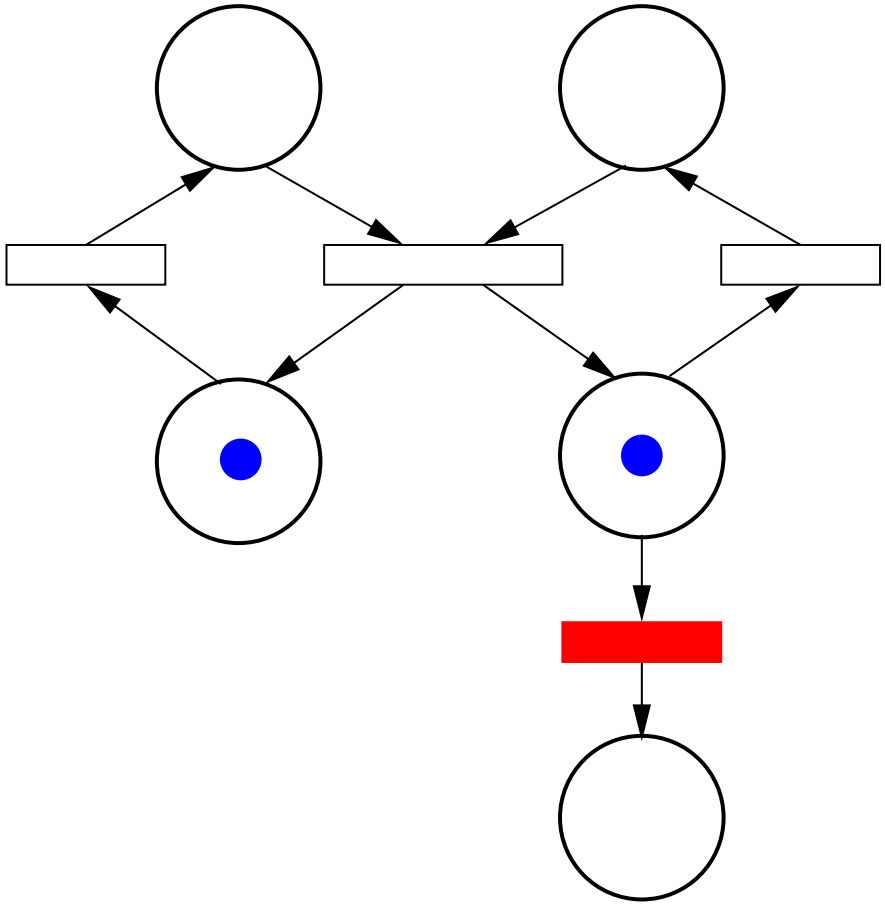
Petrinetze: Informelle Einführung



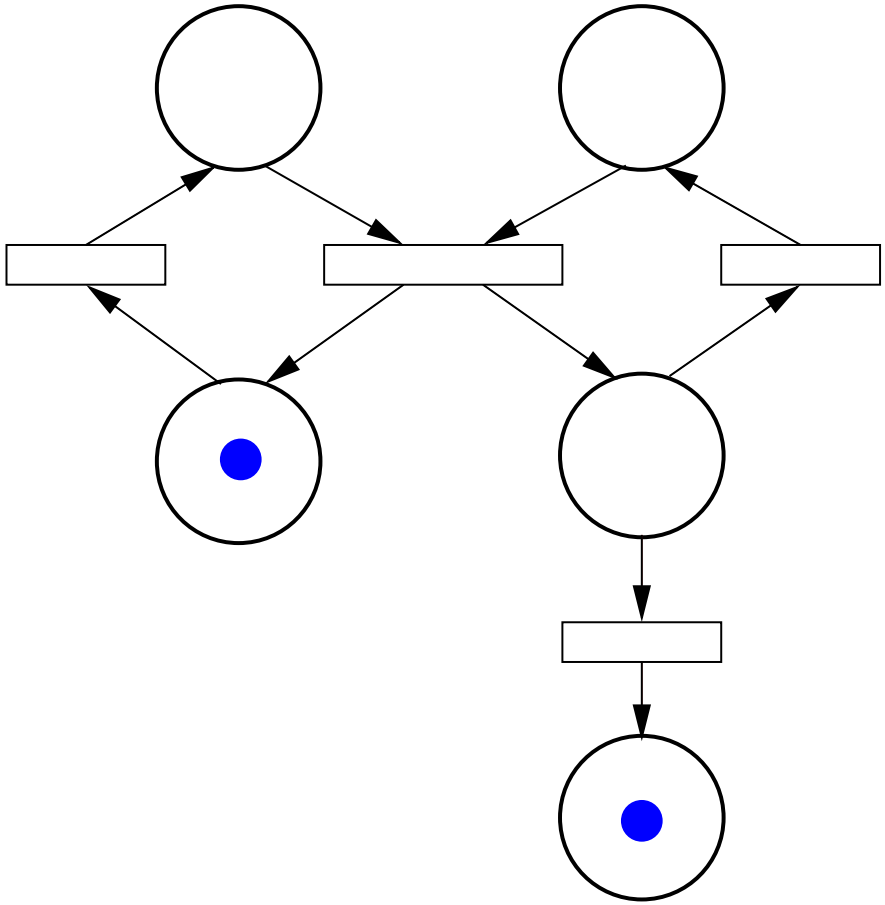
Petrinetze: Informelle Einführung



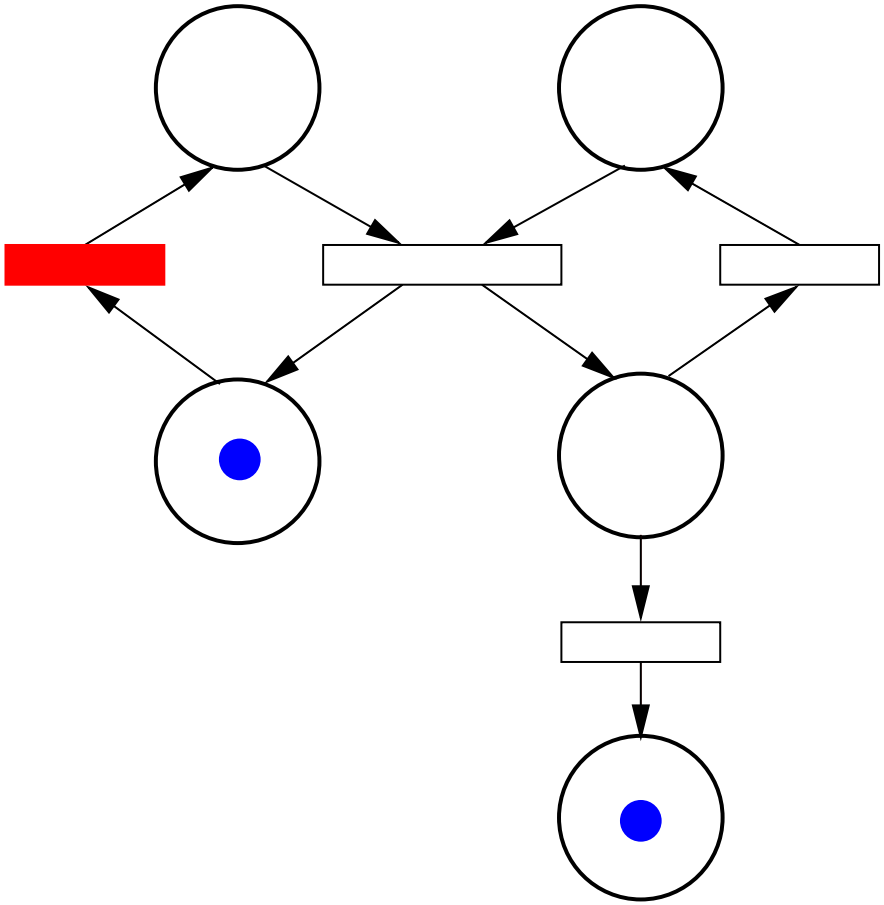
Petrinetze: Informelle Einführung



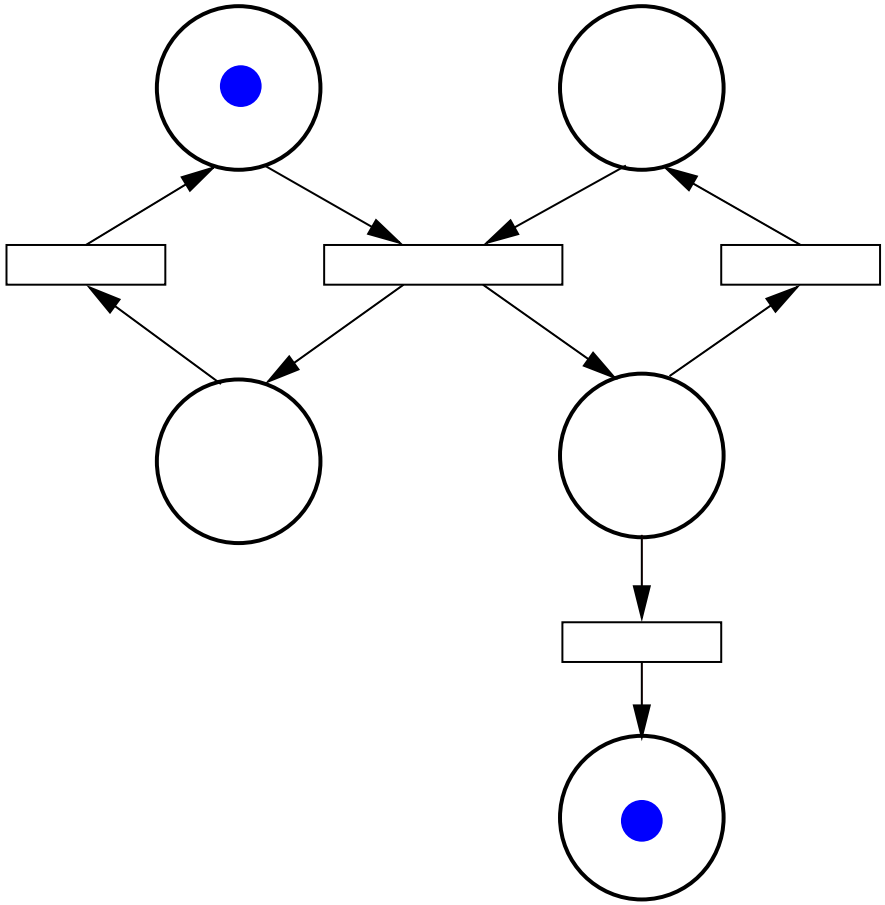
Petrinetze: Informelle Einführung



Petrinetze: Informelle Einführung



Petrinetze: Informelle Einführung



Petrinetze: Definitionen

Petrinetz (Definition)

Ein Petrinetz ist ein Tupel $N = (S, T, {}^\bullet(), ()^\bullet, m_0)$, wobei

- S eine Menge von Stellen und
- T eine Menge von Transitionen ist.
- Außerdem gibt es für jede Transition t zwei Funktionen ${}^\bullet t : S \to \mathbb{N}_0$ und $t^\bullet : S \to \mathbb{N}_0$, die angeben, wie viele Marken durch t aus einer Stelle entnommen bzw. in eine Stelle gelegt werden.
- Und $m_0 : S \to \mathbb{N}_0$ ist die Anfangsmarkierung (oder initiale Markierung).

Petrinetze: Definitionen

Petrinetz (Definition)

Ein Petrinetz ist ein Tupel $N = (S, T, {}^\bullet(), ()^\bullet, m_0)$, wobei

- S eine Menge von Stellen und
- T eine Menge von Transitionen ist.
- Außerdem gibt es für jede Transition t zwei Funktionen ${}^\bullet t : S \to \mathbb{N}_0$ und $t^\bullet : S \to \mathbb{N}_0$, die angeben, wie viele Marken durch t aus einer Stelle entnommen bzw. in eine Stelle gelegt werden.
- Und $m_0 : S \to \mathbb{N}_0$ ist die Anfangsmarkierung (oder initiale Markierung).

Der Wert ${}^\bullet t(s)$ bzw. $t^\bullet(s)$ wird jeweils als Gewicht bezeichnet.

Petrinetze: Definitionen

Markierung

Eine Markierung ist eine Funktion $m : S \to \mathbb{N}_0$.

Sie kann festhalten:

- wie viele Marken aktuell in einzelnen Stellen liegen

Petrinetze: Definitionen

Markierung

Eine Markierung ist eine Funktion $m : S \to \mathbb{N}_0$.

Sie kann festhalten:

- wie viele Marken aktuell in einzelnen Stellen liegen,
- wie viele Marken einzelnen Stellen zu entnehmen sind,

Petrinetze: Definitionen

Markierung

Eine Markierung ist eine Funktion $m : S \to \mathbb{N}_0$.

Sie kann festhalten:

- wie viele Marken aktuell in einzelnen Stellen liegen,
- wie viele Marken einzelnen Stellen zu entnehmen sind, oder
- wie viele Marken einzelnen Stellen hinzuzufügen sind.

Petrinetze: Definitionen

Markierung

Eine Markierung ist eine Funktion $m : S \to \mathbb{N}_0$.

Sie kann festhalten:

- wie viele Marken aktuell in einzelnen Stellen liegen,
- wie viele Marken einzelnen Stellen zu entnehmen sind, oder
- wie viele Marken einzelnen Stellen hinzuzufügen sind.

Falls eine Reihenfolge $s_1, \ldots, s_n$ der Stellen fixiert wurde, kann eine Markierung m auch durch ein Tupel $(m(s_1), \ldots, m(s_n))$ ausgedrückt werden.

Petrinetze: Definitionen

Eine andere Definition von Petrinetzen stellt die Verbindungen zwischen Stellen und Transitionen und die dazugehörigen Gewichte mittels einer Flussrelation dar:

$$F \subseteq (S \cup T) \times (\mathbb{N}_0 \setminus \{0\}) \times (S \cup T)$$

wobei nur Tripel der Formen

- (s, n, t)
- (t, n, s)

mit $s \in S$ und $t \in T$ erlaubt sind.

Petrinetze: Definitionen

Eine andere Definition von Petrinetzen stellt die Verbindungen zwischen Stellen und Transitionen und die dazugehörigen Gewichte mittels einer Flussrelation dar:

$$F \subseteq (S \cup T) \times (\mathbb{N}_0 \setminus \{0\}) \times (S \cup T)$$

wobei nur Tripel der Formen

- (s, n, t) (Kante von Stelle zu Transition)
- (t, n, s) (Kante von Transition zu Stelle)

mit $s \in S$ und $t \in T$ erlaubt sind.

Petrinetze: Definitionen

Eine andere Definition von Petrinetzen stellt die Verbindungen zwischen Stellen und Transitionen und die dazugehörigen Gewichte mittels einer Flussrelation dar:

$$F \subseteq (S \cup T) \times (\mathbb{N}_0 \setminus \{0\}) \times (S \cup T)$$

wobei nur Tripel der Formen

- (s, n, t) (Kante von Stelle zu Transition)
- (t, n, s) (Kante von Transition zu Stelle)

mit $s \in S$ und $t \in T$ erlaubt sind.

Zusammenhang zur vorherigen Definition:

$$(s, n, t) \in F \iff {}^\bullet t(s) = n \neq 0$$
$$(t, n, s) \in F \iff t^\bullet(s) = n \neq 0$$

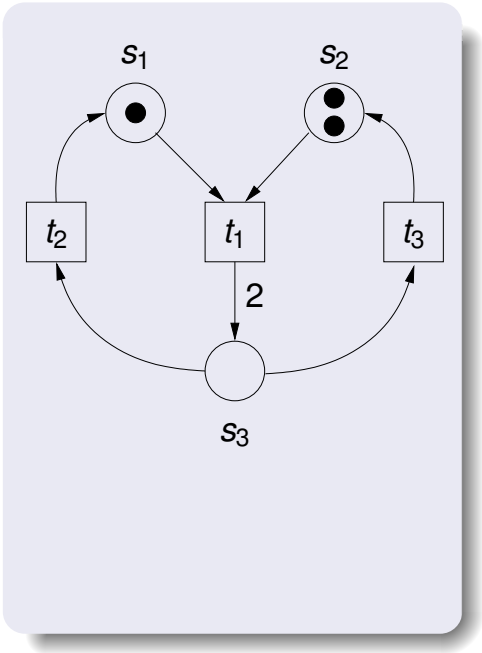
Petrinetze: Darstellung

Genaueres zur grafischen Darstellung:

- Stellen werden als Kreise, Transitionen als Quadrate oder Rechtecke, Marken als kleine ausgefüllte Kreise dargestellt.
- Kanten zwischen Stellen und Transitionen werden als Pfeile dargestellt.
- Die Kanten sind eigentlich mit dem jeweiligen Gewicht beschriftet. Dieses kann jedoch weggelassen werden, falls es den Wert 1 hat. Nur falls ein Gewicht den Wert 0 hat, wird die ganze Kante weggelassen.

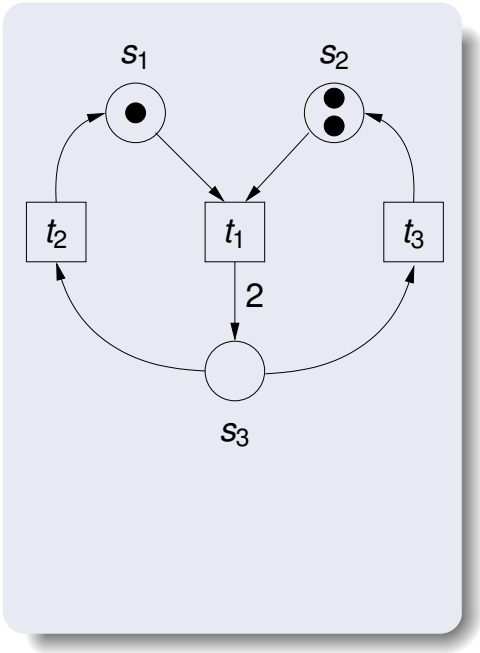
Petrinetze: Darstellung

Betrachten wir den Zusammenhang zwischen der mathematischen Notation und der grafischen Darstellung an einem Beispiel:



Petrinetze: Darstellung

Betrachten wir den Zusammenhang zwischen der mathematischen Notation und der grafischen Darstellung an einem Beispiel:



$$S = \{s_1, s_2, s_3\}$$

$$T = \{t_1, t_2, t_3\}$$

$${}^\bullet t_1 \colon s_1 \mapsto 1,\ s_2 \mapsto 1,\ s_3 \mapsto 0$$

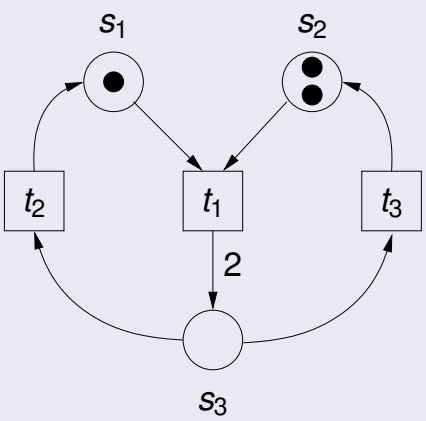
$$t_1^\bullet \colon s_1 \mapsto 0,\ s_2 \mapsto 0,\ s_3 \mapsto 2$$

$$\ldots$$

$$m_0 \colon s_1 \mapsto 1,\ s_2 \mapsto 2,\ s_3 \mapsto 0$$

Petrinetze: Darstellung

Betrachten wir den Zusammenhang zwischen der mathematischen Notation und der grafischen Darstellung an einem Beispiel:



Stellenordnung:

s_1, s_2, s_3

$$S = \{s_1, s_2, s_3\}$$

$$T = \{t_1, t_2, t_3\}$$

$${}^\bullet t_1 \colon s_1 \mapsto 1,\ s_2 \mapsto 1,\ s_3 \mapsto 0$$

$$t_1^\bullet \colon s_1 \mapsto 0,\ s_2 \mapsto 0,\ s_3 \mapsto 2$$

$$\underline{\text{oder:}}\ {}^\bullet t_1 = (1,1,0) \quad t_1^\bullet = (0,0,2)$$

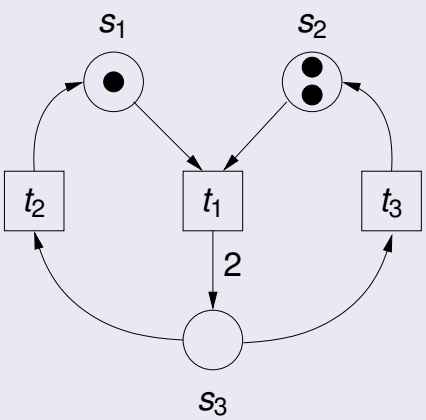
$$\ldots$$

$$m_0 \colon s_1 \mapsto 1,\ s_2 \mapsto 2,\ s_3 \mapsto 0$$

$$\underline{\text{oder:}}\ m_0 = (1,2,0)$$

Petrinetze: Darstellung

Betrachten wir den Zusammenhang zwischen der mathematischen Notation und der grafischen Darstellung an einem Beispiel:



Stellenordnung:

s_1, s_2, s_3

Alternative Notation (mit Flussrelation):

$$
\begin{aligned}
S &= \{s_1, s_2, s_3\} \\
T &= \{t_1, t_2, t_3\} \\
F &= \{(s_1, 1, t_1), (s_2, 1, t_1), (t_1, 2, s_3), \\
&\qquad (s_3, 1, t_2), (t_2, 1, s_1), \\
&\qquad (s_3, 1, t_3), (t_3, 1, s_2)\} \\
m_0 &= (1, 2, 0)
\end{aligned}
$$

Petrinetze: Wiederholung

Wichtige Konzepte:

- Markierungen: Funktionen $m : S \to \mathbb{N}_0$
- Vor- und Nachgewichte: ${}^\bullet t$ und $t^\bullet$, selbst Markierungen

Petrinetze: Wiederholung

Wichtige Konzepte:

- Markierungen: Funktionen $m : S \to \mathbb{N}_0$
- Vor- und Nachgewichte: ${}^\bullet t$ und $t^\bullet$, selbst Markierungen

Zusammenhang mit grafischer Darstellung:

- Anfangsmarkierung zur Belegung der Stellen

Petrinetze: Wiederholung

Wichtige Konzepte:

- Markierungen: Funktionen $m : S \to \mathbb{N}_0$
- Vor- und Nachgewichte: ${}^\bullet t$ und $t^\bullet$, selbst Markierungen

Zusammenhang mit grafischer Darstellung:

- Anfangsmarkierung zur Belegung der Stellen
- kein Pfeil von s zu t, falls ${}^\bullet t(s) = 0$
- kein Pfeil von t zu s, falls $t^\bullet(s) = 0$
- Pfeil von s zu t, falls ${}^\bullet t(s) = 1$
- Pfeil von t zu s, falls $t^\bullet(s) = 1$
- Pfeil mit Beschriftung n von s zu t, falls ${}^\bullet t(s) = n > 1$
- Pfeil mit Beschriftung n von t zu s, falls $t^\bullet(s) = n > 1$

Petrinetze: Dynamik

Ordnung und Operationen auf Markierungen:

Seien $m, m' : S \to \mathbb{N}_0$ zwei Markierungen, also Abbildungen von Stellen auf natürliche Zahlen.

Ordnung (Definition)

Es gilt $m' \leq m$ falls für alle $s \in S$ gilt: $m'(s) \leq m(s)$.

In diesem Fall sagt man, dass m' durch m überdeckt wird.

Petrinetze: Dynamik

Ordnung und Operationen auf Markierungen:

Seien $m, m' : S \to \mathbb{N}_0$ zwei Markierungen, also Abbildungen von Stellen auf natürliche Zahlen.

Ordnung (Definition)

Es gilt $m' \leq m$ falls für alle $s \in S$ gilt: $m'(s) \leq m(s)$.

In diesem Fall sagt man, dass m' durch m überdeckt wird.

Beispiele: Sei $|S| = 3$, $m = (0,1,2)$, $m' = (0,0,1)$.
Dann gilt $m' \leq m$, aber nicht $m \leq m'$.

Petrinetze: Dynamik

Ordnung und Operationen auf Markierungen:

Seien $m, m' : S \to \mathbb{N}_0$ zwei Markierungen, also Abbildungen von Stellen auf natürliche Zahlen.

Ordnung (Definition)

Es gilt $m' \leq m$ falls für alle $s \in S$ gilt: $m'(s) \leq m(s)$.

In diesem Fall sagt man, dass m' durch m überdeckt wird.

Beispiele: Sei $|S| = 3$, $m = (0,1,2)$, $m' = (0,0,1)$.
Dann gilt $m' \leq m$, aber nicht $m \leq m'$.

Es gilt auch $(0,1,0) \leq (0,1,0)$.

Petrinetze: Dynamik

Ordnung und Operationen auf Markierungen:

Seien $m, m' : S \to \mathbb{N}_0$ zwei Markierungen, also Abbildungen von Stellen auf natürliche Zahlen.

Ordnung (Definition)

Es gilt $m' \leq m$ falls für alle $s \in S$ gilt: $m'(s) \leq m(s)$.

In diesem Fall sagt man, dass m' durch m überdeckt wird.

Beispiele: Sei $|S| = 3$, $m = (0,1,2)$, $m' = (0,0,1)$.
Dann gilt $m' \leq m$, aber nicht $m \leq m'$.

Es gilt auch $(0,1,0) \leq (0,1,0)$.

Aber nicht $(3,1,2) \leq (5,1000,1)$.

Petrinetze: Dynamik

Ordnung und Operationen auf Markierungen:

Seien $m, m' : S \to \mathbb{N}_0$ zwei Markierungen, also Abbildungen von Stellen auf natürliche Zahlen.

Ordnung (Definition)

Es gilt $m' \leq m$ falls für alle $s \in S$ gilt: $m'(s) \leq m(s)$.

In diesem Fall sagt man, dass m' durch m überdeckt wird.

Beispiele: Sei $|S| = 3$, $m = (0,1,2)$, $m' = (0,0,1)$.
Dann gilt $m' \leq m$, aber nicht $m \leq m'$.

Es gilt auch $(0,1,0) \leq (0,1,0)$.

Aber nicht $(3,1,2) \leq (5,1000,1)$.

Und weder $(0,1,2) \leq (0,2,1)$, noch $(0,2,1) \leq (0,1,2)$.

Petrinetze: Dynamik

Ordnung und Operationen auf Markierungen:

Seien $m, m' : S \to \mathbb{N}_0$ zwei Markierungen, also Abbildungen von Stellen auf natürliche Zahlen.

Addition (Definition)

Wir definieren $m'' = m \oplus m'$, wobei $m'' : S \to \mathbb{N}_0$ mit $m''(s) = m(s) + m'(s)$ für alle $s \in S$.

Petrinetze: Dynamik

Ordnung und Operationen auf Markierungen:

Seien $m, m' : S \to \mathbb{N}_0$ zwei Markierungen, also Abbildungen von Stellen auf natürliche Zahlen.

Addition (Definition)

Wir definieren $m'' = m \oplus m'$, wobei $m'' : S \to \mathbb{N}_0$ mit $m''(s) = m(s) + m'(s)$ für alle $s \in S$.

Beispiel: $(0,1,2) \oplus (0,0,1) = (0,1,3)$

Petrinetze: Dynamik

Ordnung und Operationen auf Markierungen:

Seien $m, m' : S \to \mathbb{N}_0$ zwei Markierungen, also Abbildungen von Stellen auf natürliche Zahlen.

Addition (Definition)

Wir definieren $m'' = m \oplus m'$, wobei $m'' : S \to \mathbb{N}_0$ mit $m''(s) = m(s) + m'(s)$ für alle $s \in S$.

Beispiel: $(0,1,2) \oplus (0,0,1) = (0,1,3)$

Subtraktion (Definition)

Falls $m' \leq m$, definieren wir $m'' = m \ominus m'$, wobei $m'' : S \to \mathbb{N}_0$ mit $m''(s) = m(s) - m'(s)$ für alle $s \in S$.

Petrinetze: Dynamik

Ordnung und Operationen auf Markierungen:

Seien $m, m' : S \to \mathbb{N}_0$ zwei Markierungen, also Abbildungen von Stellen auf natürliche Zahlen.

Addition (Definition)

Wir definieren $m'' = m \oplus m'$, wobei $m'' : S \to \mathbb{N}_0$ mit $m''(s) = m(s) + m'(s)$ für alle $s \in S$.

Beispiel: $(0,1,2) \oplus (0,0,1) = (0,1,3)$

Subtraktion (Definition)

Falls $m' \leq m$, definieren wir $m'' = m \ominus m'$, wobei $m'' : S \to \mathbb{N}_0$ mit $m''(s) = m(s) - m'(s)$ für alle $s \in S$.

Beispiel: $(0,1,2) \ominus (0,0,1) = (0,1,1)$

Petrinetze: Dynamik

Ordnung und Operationen auf Markierungen:

Seien $m, m' : S \to \mathbb{N}_0$ zwei Markierungen, also Abbildungen von Stellen auf natürliche Zahlen.

Addition (Definition)

Wir definieren $m'' = m \oplus m'$, wobei $m'' : S \to \mathbb{N}_0$ mit $m''(s) = m(s) + m'(s)$ für alle $s \in S$.

Beispiel: $(0,1,2) \oplus (0,0,1) = (0,1,3) = (0,0,1) \oplus (0,1,2)$

Subtraktion (Definition)

Falls $m' \leq m$, definieren wir $m'' = m \ominus m'$, wobei $m'' : S \to \mathbb{N}_0$ mit $m''(s) = m(s) - m'(s)$ für alle $s \in S$.

Beispiel: $(0,1,2) \ominus (0,0,1) = (0,1,1)$

Petrinetze: Dynamik

Weitere Konzepte:

Aktivierung (Definition)

Eine Transition t ist für eine Markierung m aktiviert falls ${}^\bullet t \leq m$ gilt. (Das heißt, falls in m je Stelle genug Marken vorhanden sind, um die Vorbedingung von t zu erfüllen.)

Petrinetze: Dynamik

Weitere Konzepte:

Aktivierung (Definition)

Eine Transition t ist für eine Markierung m aktiviert falls ${}^\bullet t \leq m$ gilt. (Das heißt, falls in m je Stelle genug Marken vorhanden sind, um die Vorbedingung von t zu erfüllen.)

Schalten (Definition)

Sei t eine Transition und m eine Markierung, für die t aktiviert ist. Dann kann t schalten, was zu der Nachfolgemarkierung $m' = m \ominus {}^\bullet t \oplus t^\bullet$ führt.

Petrinetze: Dynamik

Weitere Konzepte:

Aktivierung (Definition)

Eine Transition t ist für eine Markierung m aktiviert falls ${}^\bullet t \leq m$ gilt. (Das heißt, falls in m je Stelle genug Marken vorhanden sind, um die Vorbedingung von t zu erfüllen.)

Schalten (Definition)

Sei t eine Transition und m eine Markierung, für die t aktiviert ist. Dann kann t schalten, was zu der Nachfolgemarkierung $m' = m \ominus {}^\bullet t \oplus t^\bullet$ führt; zu lesen als $(m \ominus {}^\bullet t) \oplus t^\bullet$.

Petrinetze: Dynamik

Weitere Konzepte:

Aktivierung (Definition)

Eine Transition t ist für eine Markierung m aktiviert falls ${}^\bullet t \leq m$ gilt. (Das heißt, falls in m je Stelle genug Marken vorhanden sind, um die Vorbedingung von t zu erfüllen.)

Schalten (Definition)

Sei t eine Transition und m eine Markierung, für die t aktiviert ist. Dann kann t schalten, was zu der Nachfolgemarkierung $m' = m \ominus {}^\bullet t \oplus t^\bullet$ führt; zu lesen als $(m \ominus {}^\bullet t) \oplus t^\bullet$.

Symbolisch dargestellt: $m \,[t\rangle\, m'$.

Petrinetze: Dynamik

Weitere Konzepte:

Erreichbarkeit (Definition)

Man nennt eine Markierung m erreichbar, wenn es eine endliche Folge von Transitionen $t_1, \ldots, t_n$ gibt mit

$$m_0 \, [t_1\rangle \, m_1 \, [t_2\rangle \, \cdots \, m_{n-1} \, [t_n\rangle \, m \,,$$

wobei m_0 die Anfangsmarkierung des Petrinetzes ist.

Petrinetze: Dynamik

Weitere Konzepte:

Erreichbarkeit (Definition)

Man nennt eine Markierung m erreichbar, wenn es eine endliche Folge von Transitionen $t_1, \ldots, t_n$ gibt mit

$$m_0 \, [t_1\rangle \, m_1 \, [t_2\rangle \, \cdots \, m_{n-1} \, [t_n\rangle \, m \,,$$

wobei m_0 die Anfangsmarkierung des Petrinetzes ist.

Man schreibt auch $m_0 \, [t_1 \ldots t_n\rangle \, m$, oder $m_0 \, [\tilde{t}\,\rangle \, m$ mit $\tilde{t} = t_1 \ldots t_n$.

Petrinetze: Dynamik

Weitere Konzepte:

Erreichbarkeit (Definition)

Man nennt eine Markierung m erreichbar, wenn es eine endliche Folge von Transitionen $t_1, \ldots, t_n$ gibt mit

$$m_0 \,[t_1\rangle\, m_1 \,[t_2\rangle\, \cdots\, m_{n-1} \,[t_n\rangle\, m\,,$$

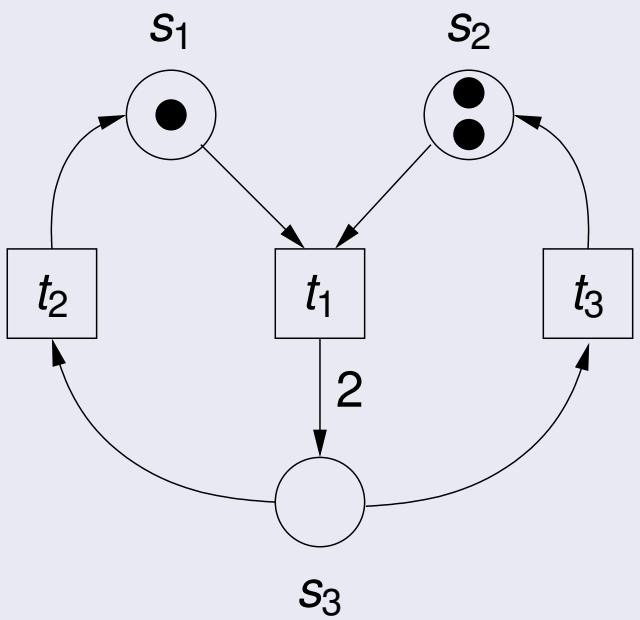
wobei m_0 die Anfangsmarkierung des Petrinetzes ist.

Man schreibt auch $m_0 \,[t_1 \ldots t_n\rangle\, m$, oder $m_0 \,[\tilde{t}\,\rangle\, m$ mit $\tilde{t} = t_1 \ldots t_n$.

Die Sequenz $\tilde{t}$ nennt man Schaltfolge.

Auch die leere Schaltfolge $\tilde{t} = \varepsilon$ is möglich. In diesem Fall ändert sich die Markierung nicht: $m \,[\varepsilon\rangle\, m$, für jede Markierung m.

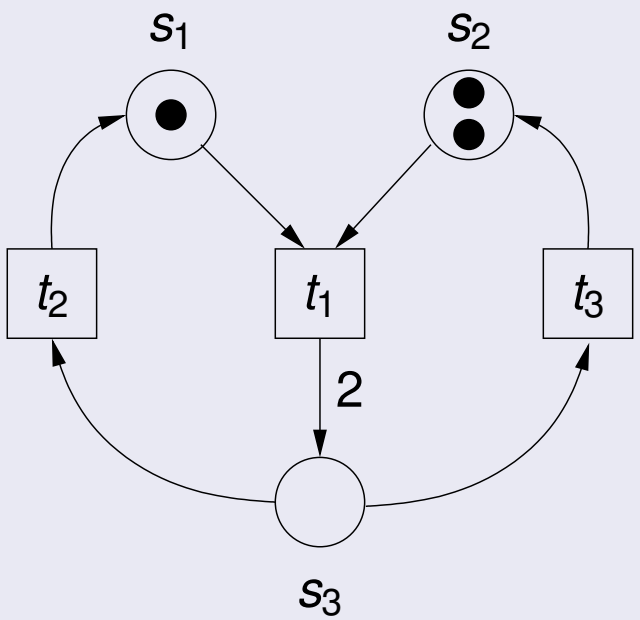
Petrinetze: Dynamik



Stellenordnung: s_1, s_2, s_3

Die Markierung $m_2 = (1,1,1)$ ist in zwei Schritten erreichbar:

Petrinetze: Dynamik



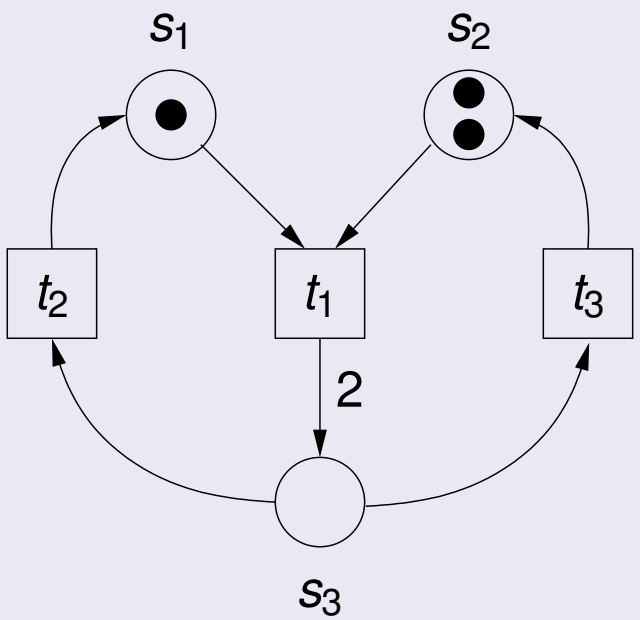
Stellenordnung: s_1, s_2, s_3

Die Markierung $m_2 = (1,1,1)$ ist in zwei Schritten erreichbar:

1. $^\bullet t_1 = (1,1,0) \leq (1,2,0) = m_0$

   $m_1 = m_0 \ominus {}^\bullet t_1 \oplus t_1^\bullet$
   $= (1,2,0) \ominus (1,1,0) \oplus (0,0,2)$
   $= (0,1,2)$

Petrinetze: Dynamik



Stellenordnung: s_1, s_2, s_3

Die Markierung $m_2 = (1,1,1)$ ist in zwei Schritten erreichbar:

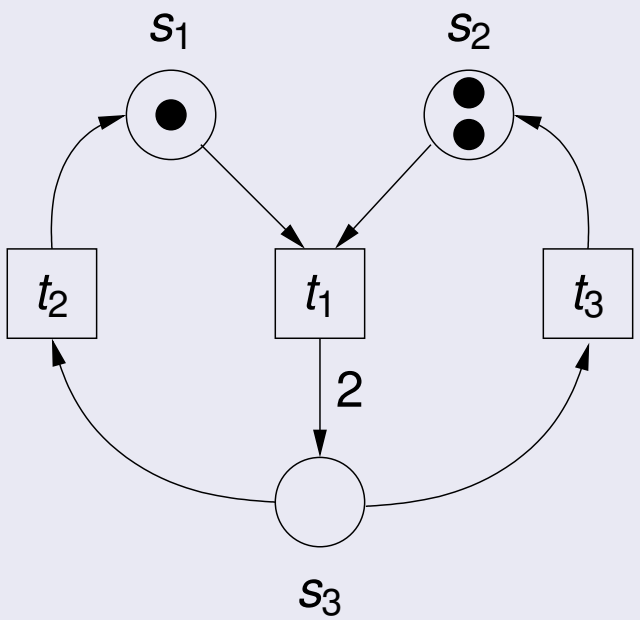
1. $^\bullet t_1 = (1,1,0) \leq (1,2,0) = m_0$

   $m_1 = m_0 \ominus {}^\bullet t_1 \oplus t_1^\bullet$
   $= (1,2,0) \ominus (1,1,0) \oplus (0,0,2)$
   $= (0,1,2)$

2. $^\bullet t_2 = (0,0,1) \leq (0,1,2) = m_1$

   $m_2 = m_1 \ominus {}^\bullet t_2 \oplus t_2^\bullet$
   $= (0,1,2) \ominus (0,0,1) \oplus (1,0,0)$
   $= (1,1,1)$

Petrinetze: Dynamik



Stellenordnung: s_1, s_2, s_3

Die Markierung $m_2 = (1,1,1)$ ist in zwei Schritten erreichbar:

1. $^\bullet t_1 = (1,1,0) \leq (1,2,0) = m_0$

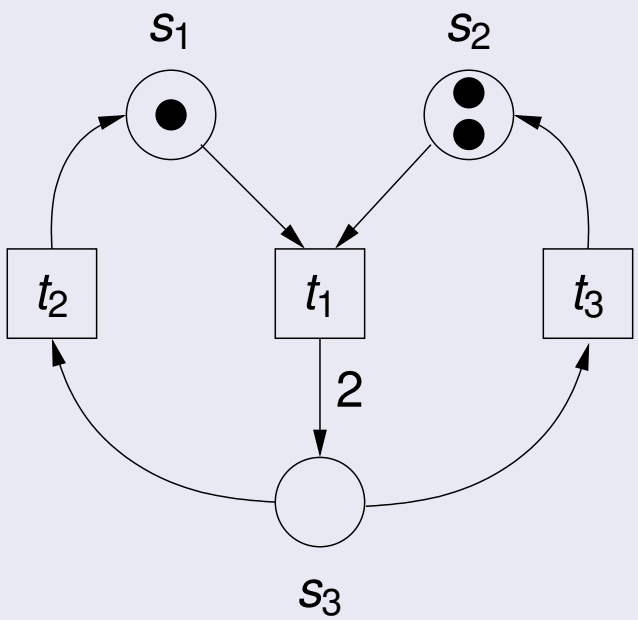
   $m_1 = m_0 \ominus {}^\bullet t_1 \oplus t_1^\bullet$
   $= (1,2,0) \ominus (1,1,0) \oplus (0,0,2)$
   $= (0,1,2)$

2. $^\bullet t_2 = (0,0,1) \leq (0,1,2) = m_1$

   $m_2 = m_1 \ominus {}^\bullet t_2 \oplus t_2^\bullet$
   $= (0,1,2) \ominus (0,0,1) \oplus (1,0,0)$
   $= (1,1,1)$

Es gilt also: $m_0 \,[t_1\rangle\, m_1 \,[t_2\rangle\, m_2$, oder: $m_0 \,[t_1\, t_2\rangle\, m_2$.

Petrinetze: Dynamik



Stellenordnung: s_1, s_2, s_3

Die Markierung $m_2 = (1,1,1)$ ist in zwei Schritten erreichbar:

1. $^\bullet t_1 = (1,1,0) \leq (1,2,0) = m_0$

   $m_1 = m_0 \ominus {}^\bullet t_1 \oplus t_1^\bullet$
   $= (1,2,0) \ominus (1,1,0) \oplus (0,0,2)$
   $= (0,1,2)$

2. $^\bullet t_2 = (0,0,1) \leq (0,1,2) = m_1$

   $m_2 = m_1 \ominus {}^\bullet t_2 \oplus t_2^\bullet$
   $= (0,1,2) \ominus (0,0,1) \oplus (1,0,0)$
   $= (1,1,1)$

Es gilt also: $m_0 \,[t_1\rangle\, m_1 \,[t_2\rangle\, m_2$, oder: $m_0 \,[t_1\, t_2\rangle\, m_2$.

Es gilt auch: $m_0 \,[t_1\rangle\, m_1 \,[t_3\rangle\, (0,2,1)$.

Petrinetze: Dynamik

Nicht-Determinismus

Petrinetze sind ein nicht-deterministischer Mechanismus.

Das heißt, zu einer Markierung kann es mehrere Nachfolgemarkierungen geben.

Petrinetze: Dynamik

Nicht-Determinismus

Petrinetze sind ein nicht-deterministischer Mechanismus.

Das heißt, zu einer Markierung kann es mehrere Nachfolgemarkierungen geben.

Die Frage, wer die Nachfolgemarkierung auswählt, stellt sich für die Modellierung nicht. Das Modell beschreibt, dass alle diese Nachfolgemarkierungen möglich sind und trifft keine Auswahl.

Petrinetze: Dynamik

Zustandsübergangsdiagramm eines Petrinetzes (Definition)

Sei $N = (S, T, {}^\bullet(), ()^\bullet, m_0)$ ein Petrinetz.

Dann besteht das zu N gehörende Zustandsübergangsdiagramm aus folgenden Komponenten:

- Zustandsmenge Z: Menge aller erreichbaren Markierungen
- Kantenbeschriftungsmenge L: Menge aller Transitionen
- Übergangsmenge U: $(m, t, m') \in U \iff m\,[t\rangle\, m'$
- Startzustand z_0: die Anfangsmarkierung m_0

Petrinetze: Dynamik

Zustandsübergangsdiagramm eines Petrinetzes (Definition)

Sei $N = (S, T, {}^\bullet(), ()^\bullet, m_0)$ ein Petrinetz.

Dann besteht das zu N gehörende Zustandsübergangsdiagramm aus folgenden Komponenten:

- Zustandsmenge Z: Menge aller erreichbaren Markierungen
- Kantenbeschriftungsmenge L: Menge aller Transitionen
- Übergangsmenge U: $(m, t, m') \in U \iff m\,[t\rangle\, m'$
- Startzustand z_0: die Anfangsmarkierung m_0

Das Zustandsübergangsdiagramm eines Petrinetzes nennt man auch dessen Erreichbarkeitsgraph.

Petrinetze: Dynamik

Zustandsübergangsdiagramm eines Petrinetzes (Definition)

Sei $N = (S, T, {}^\bullet(), ()^\bullet, m_0)$ ein Petrinetz.

Dann besteht das zu N gehörende Zustandsübergangsdiagramm aus folgenden Komponenten:

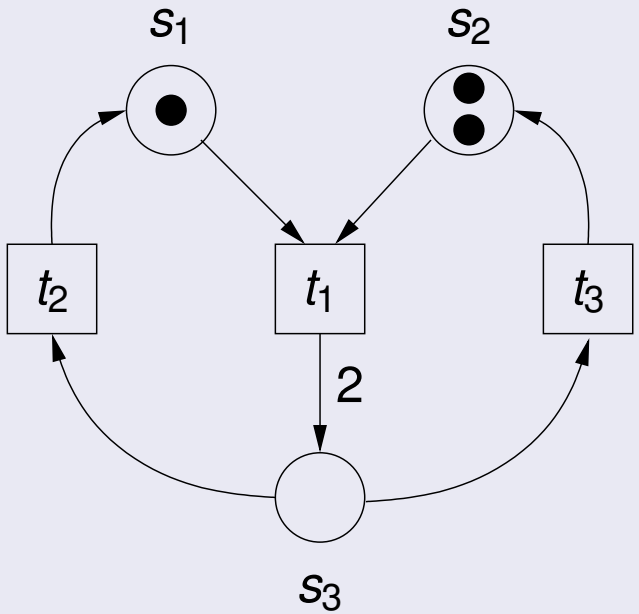
- Zustandsmenge Z: Menge aller erreichbaren Markierungen
- Kantenbeschriftungsmenge L: Menge aller Transitionen
- Übergangsmenge U: $(m, t, m') \in U \iff m\,[t\rangle\, m'$
- Startzustand z_0: die Anfangsmarkierung m_0

Das Zustandsübergangsdiagramm eines Petrinetzes nennt man auch dessen Erreichbarkeitsgraph.

Trotz Endlichkeit des Petrinetzes kann der Erreichbarkeitsgraph unendlich werden!

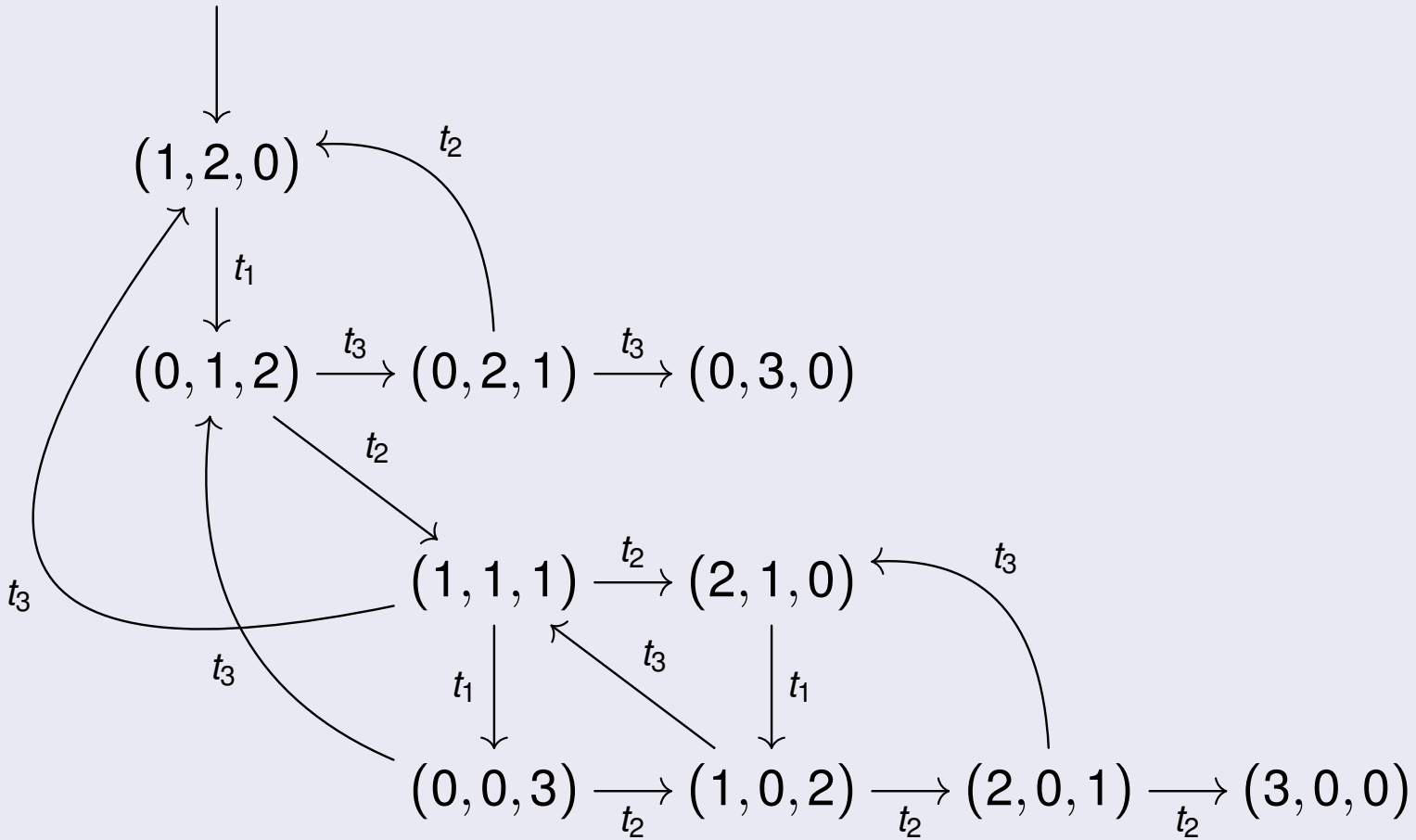
Petrinetze: Dynamik

Beispiel: Bestimme den Erreichbarkeitsgraph für das folgende Petrinetz



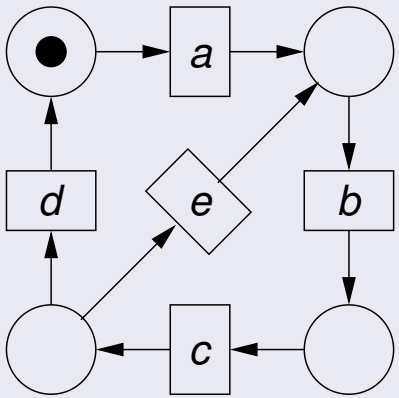
Petrinetze: Dynamik

Erreichbarkeitsgraph für das Beispielnetz:



Petrinetze: Dynamik

Beispiel: Bestimme den Erreichbarkeitsgraph für das folgende Petrinetz



Petrinetze: Dynamik

Frage: Gibt es für jedes (endliche) Zustandsübergangsdiagramm ein Petrinetz, dessen Erreichbarkeitsgraph gerade jenes Zustandsübergangsdiagramm (bzw. dessen erreichbarer Teil) ist?

Petrinetze: Dynamik

Frage: Gibt es für jedes (endliche) Zustandsübergangsdiagramm ein Petrinetz, dessen Erreichbarkeitsgraph gerade jenes Zustandsübergangsdiagramm (bzw. dessen erreichbarer Teil) ist?

Idee:

- Zustände werden zu Stellen.
- Übergänge werden zu Transitionen.
- Die Stelle, die den Startzustand darstellt, ist als einzige zu Beginn mit einer Marke belegt.

Petrinetze: Dynamik

Frage: Gibt es für jedes (endliche) Zustandsübergangsdiagramm ein Petrinetz, dessen Erreichbarkeitsgraph gerade jenes Zustandsübergangsdiagramm (bzw. dessen erreichbarer Teil) ist?

Idee:

- Zustände werden zu Stellen.
- Übergänge werden zu Transitionen.
- Die Stelle, die den Startzustand darstellt, ist als einzige zu Beginn mit einer Marke belegt.

Jedoch:

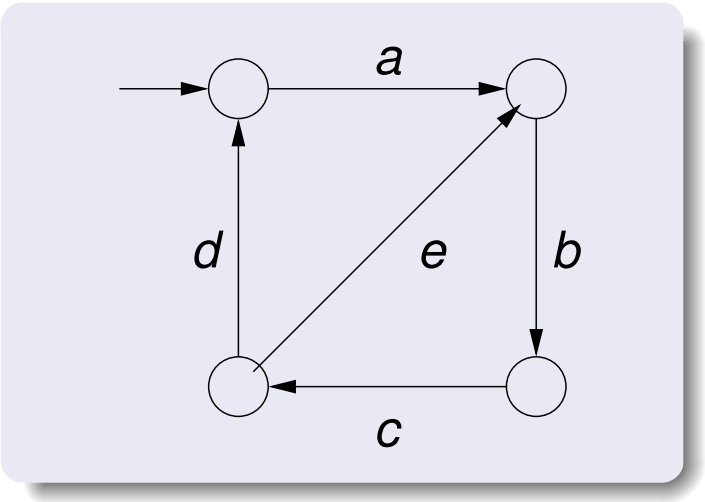
- Das entstandene Petrinetz enthält keinerlei Nebenläufigkeit.
- Bei der Umwandlung

  $$\textit{Petrinetz} \rightarrow \textit{Zustandsübergangsdiagramm} \rightarrow \textit{Petrinetz}$$

  wird das zweite Petrinetz im Allgemeinen viel größer als das erste.

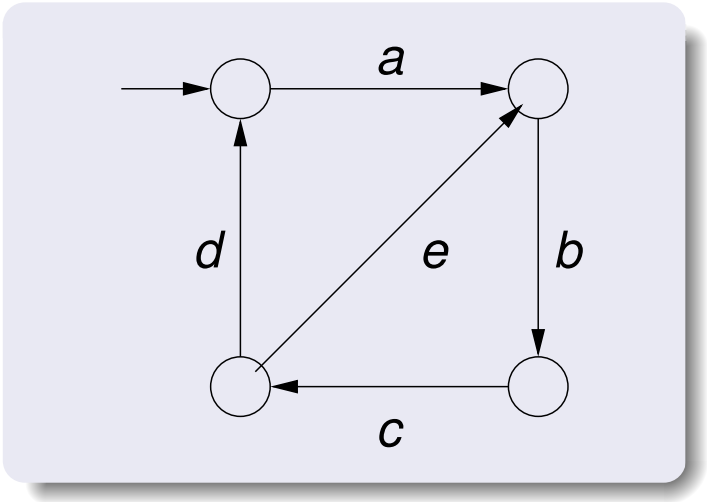
Petrinetze: Dynamik

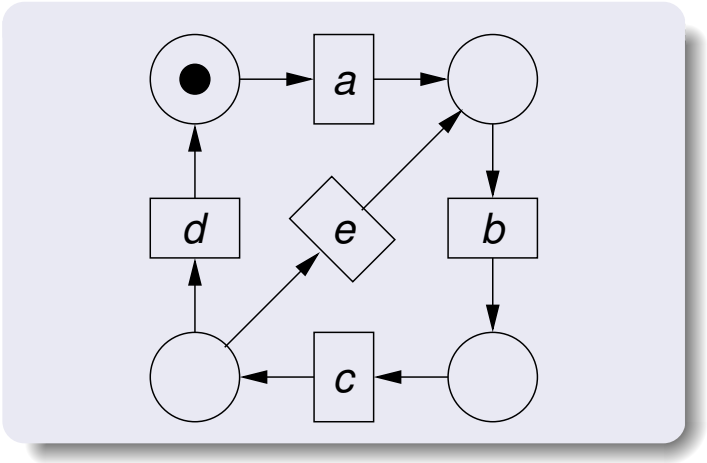
Beispiel: Umwandlung eines Zustandsübergangsdiagramms in ein Petrinetz (mit entsprechendem Erreichbarkeitsgraph)



Petrinetze: Dynamik

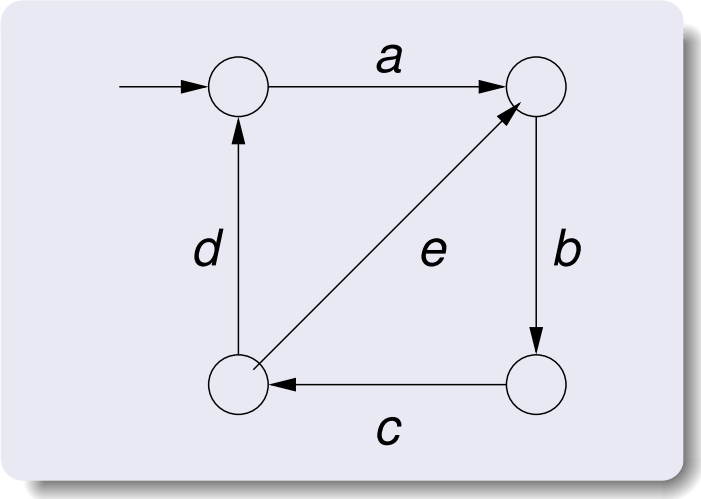
Beispiel: Umwandlung eines Zustandsübergangsdiagramms in ein Petrinetz (mit entsprechendem Erreichbarkeitsgraph)

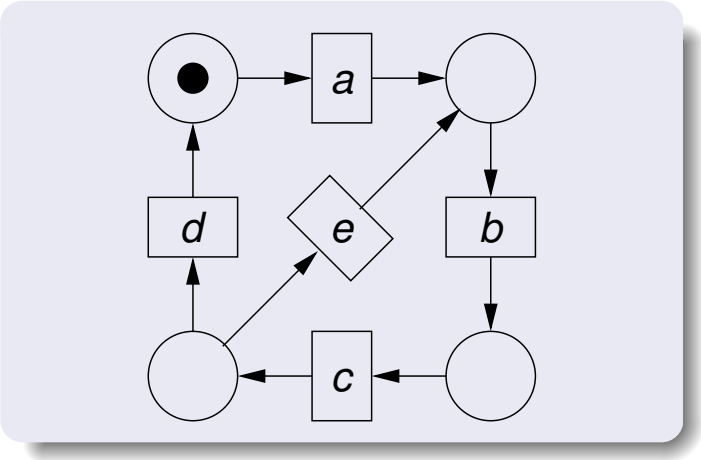




Petrinetze: Dynamik

Beispiel: Umwandlung eines Zustandsübergangsdiagramms in ein Petrinetz (mit entsprechendem Erreichbarkeitsgraph)





Bemerkung:

Die Konstruktion funktioniert so immer dann, wenn jede Beschriftung im Zustandsübergangsdiagramm höchstens einmal vorkommt.

Eigenschaften von Petrinetzen, Überdeckungsgraphen

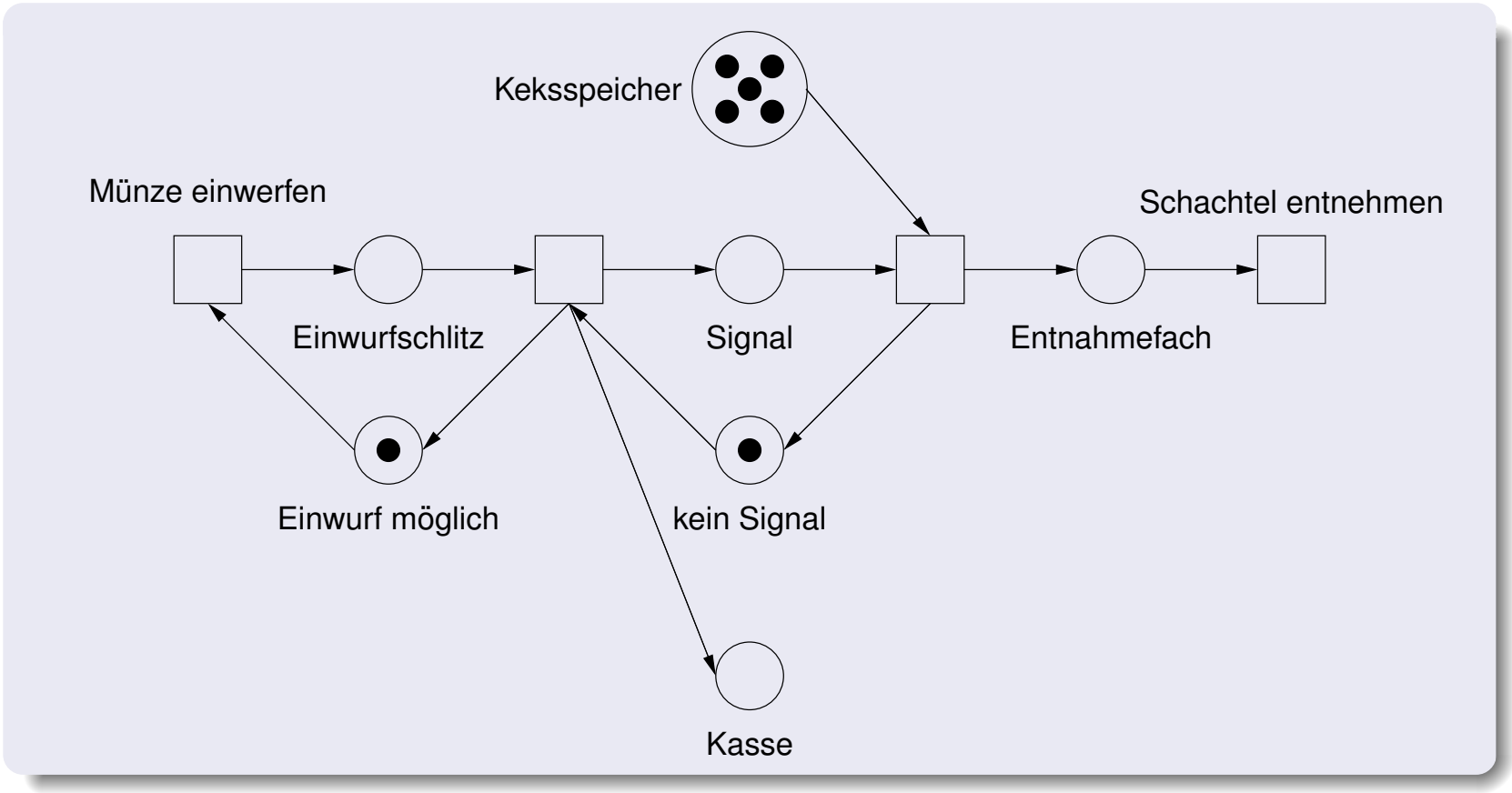
Petrinetz-Beispiel: Keksautomat

Wir modellieren einen Keksautomaten mit folgenden Bestandteilen:

- extern: Einwurfschlitz, Entnahmefach
- intern: Keksspeicher, Kasse, Signalweiterleitung (Einwurf einer Münze soll Signal erzeugen, danach Ausgabe Keks)

Nach: „Petrinetze – Modellierungstechnik, Analysemethoden, Fallstudien“ von W. Reisig

Petrinetz-Beispiel: Keksautomat



Petrinetz-Beispiel: Keksautomat

Ist der Keksautomat so in Ordnung?

Petrinetz-Beispiel: Keksautomat

Ist der Keksautomat so in Ordnung?

Problem: Wenn der Keksspeicher leer ist, ist immer noch Münzeinwurf möglich, ohne Rückgabe.

Es gibt verschiedene denkbare Lösungen für dieses Problem: Rückgabe der Münze, Keks-Zähler, ...

Petrinetze: Lebendigkeit und Verklemmungen

Wir betrachten zunächst Begriffe wie Lebendigkeit und Deadlock (= Verklemmung).

Petrinetze: Lebendigkeit und Verklemmungen

Wir betrachten zunächst Begriffe wie Lebendigkeit und Deadlock (= Verklemmung).

Starke Lebendigkeit (Definition)

Man nennt ein Petrinetz stark lebendig, wenn es für jede Transition t und jede (von m_0 aus) erreichbare Markierung m eine Markierung m' gibt, die von m aus erreichbar ist und für die t aktiviert ist.

Petrinetze: Lebendigkeit und Verklemmungen

Wir betrachten zunächst Begriffe wie Lebendigkeit und Deadlock (= Verklemmung).

Starke Lebendigkeit (Definition)

Man nennt ein Petrinetz stark lebendig, wenn es für jede Transition t und jede (von m_0 aus) erreichbare Markierung m eine Markierung m' gibt, die von m aus erreichbar ist und für die t aktiviert ist.

Bezüglich des Erreichbarkeitsgraphen bedeutet dies, für jede Transition t: von jedem Knoten des Graphen aus ist ein Übergang erreichbar, der mit t beschriftet ist.

Petrinetze: Lebendigkeit und Verklemmungen

Schwache Lebendigkeit (Definition)

Man nennt ein Petrinetz schwach lebendig, wenn es für jede Transition t eine (von m_0 aus) erreichbare Markierung gibt, für die t aktiviert ist.

Petrinetze: Lebendigkeit und Verklemmungen

Schwache Lebendigkeit (Definition)

Man nennt ein Petrinetz schwach lebendig, wenn es für jede Transition t eine (von m_0 aus) erreichbare Markierung gibt, für die t aktiviert ist.

Bezüglich des Erreichbarkeitsgraphen bedeutet dies, dass es für jede Transition t mindestens einen Übergang gibt, der mit t beschriftet ist.

Petrinetze: Lebendigkeit und Verklemmungen

Verklemmung (Definition)

Man sagt, dass ein Petrinetz eine Verklemmung (oder einen Deadlock) enthält, wenn es eine (von m_0 aus) erreichbare Markierung gibt, für die keine Transition aktiviert ist.

Petrinetze: Lebendigkeit und Verklemmungen

Verklemmung (Definition)

Man sagt, dass ein Petrinetz eine Verklemmung (oder einen Deadlock) enthält, wenn es eine (von m_0 aus) erreichbare Markierung gibt, für die keine Transition aktiviert ist.

Bezüglich des Erreichbarkeitsgraphen bedeutet dies, dass es einen Knoten gibt, von dem aus es keinen Übergang gibt.

Petrinetze: Lebendigkeit und Verklemmungen

Verklemmung (Definition)

Man sagt, dass ein Petrinetz eine Verklemmung (oder einen Deadlock) enthält, wenn es eine (von m_0 aus) erreichbare Markierung gibt, für die keine Transition aktiviert ist.

Bezüglich des Erreichbarkeitsgraphen bedeutet dies, dass es einen Knoten gibt, von dem aus es keinen Übergang gibt.

Ein Petrinetz, das keine Verklemmung enthält, nennt man verklemmungsfrei.

Petrinetze: Lebendigkeit und Verklemmungen

Stärke der starken Lebendigkeit

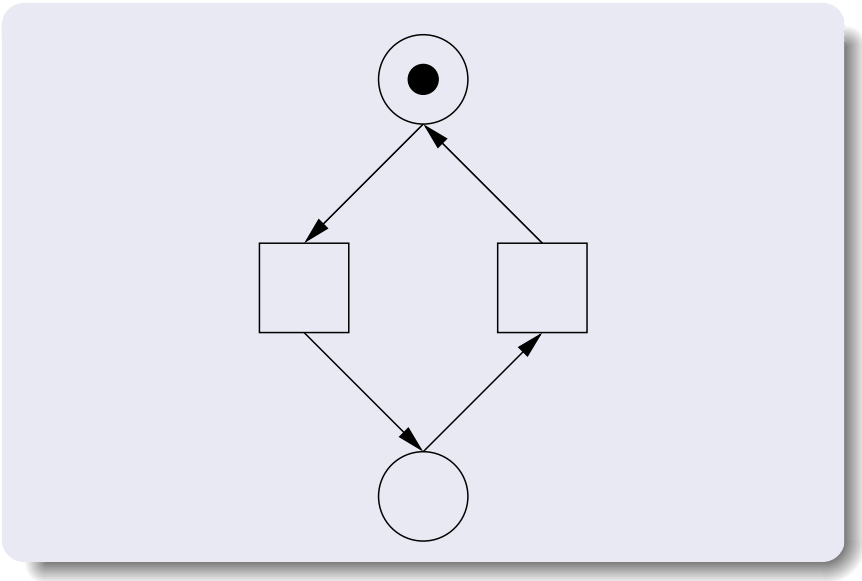
Wenn wir nur Petrinetze betrachten, deren Transitionsmenge nicht leer ist, gilt:

Jedes stark lebendige Petrinetz ist sowohl schwach lebendig als auch verklemmungsfrei.

Petrinetze: Lebendigkeit und Verklemmungen

Beispiele für Lebendigkeit und Verklemmungen:

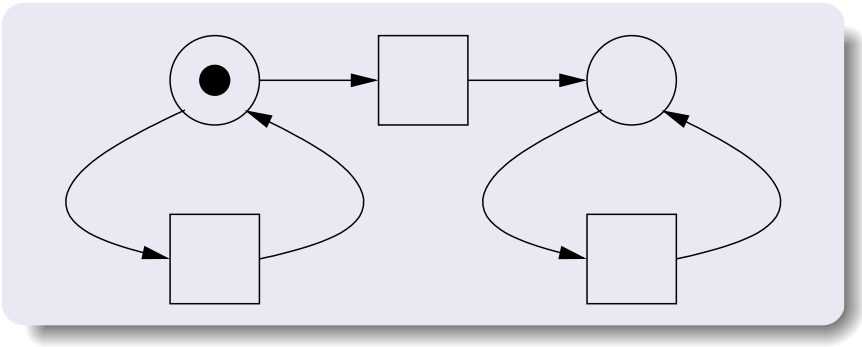
Ein Beispiel für ein stark lebendiges Petrinetz . . .



Petrinetze: Lebendigkeit und Verklemmungen

Beispiele für Lebendigkeit und Verklemmungen:

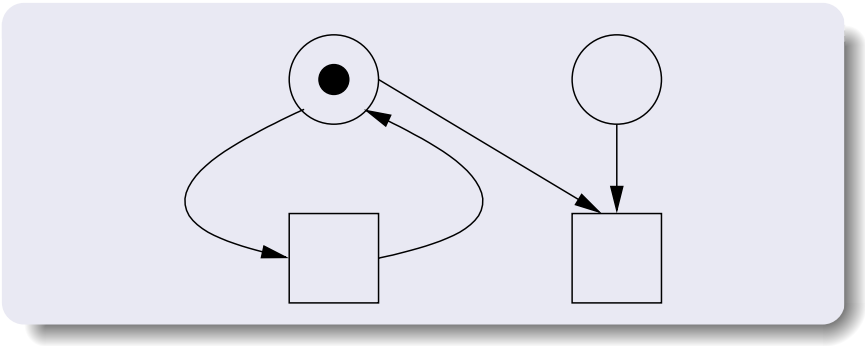
Ein Beispiel für ein schwach lebendiges und verklemmungsfreies Petrinetz, das jedoch nicht stark lebendig ist . . .



Petrinetze: Lebendigkeit und Verklemmungen

Beispiele für Lebendigkeit und Verklemmungen:

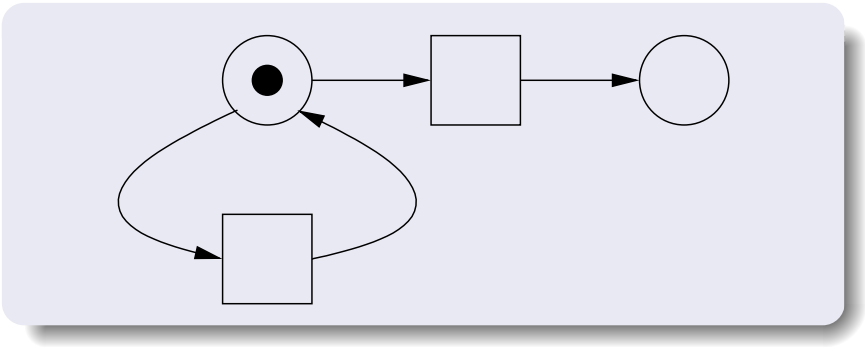
Ein Beispiel für ein verklemmungsfreies Petrinetz, das jedoch nicht schwach lebendig ist . . .



Petrinetze: Lebendigkeit und Verklemmungen

Beispiele für Lebendigkeit und Verklemmungen:

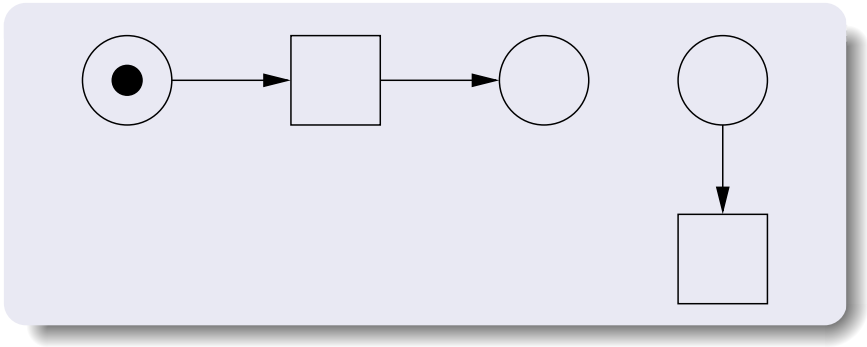
Ein Beispiel für ein schwach lebendiges Petrinetz, das jedoch eine Verklemmung enthält . . .



Petrinetze: Lebendigkeit und Verklemmungen

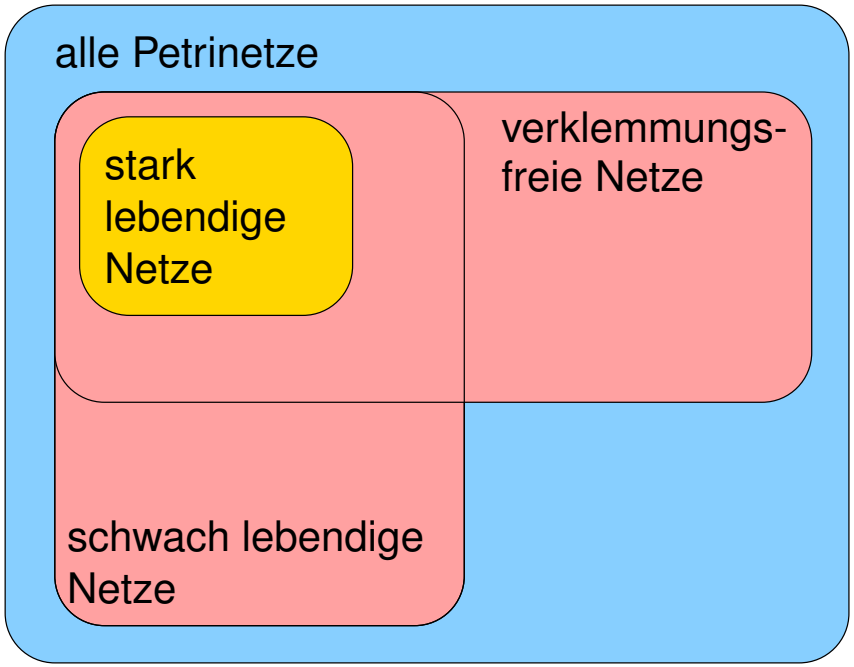
Beispiele für Lebendigkeit und Verklemmungen:

Ein Beispiel für ein Petrinetz, das eine Verklemmung enthält und das auch nicht schwach lebendig ist . . .



Petrinetze: Lebendigkeit und Verklemmungen

Überblick über die verschiedenen Petrinetzklassen (unter der Voraussetzung, dass jedes betrachtete Petrinetz mindestens eine Transition enthält):



Petrinetze: Beschränktheit

Sichere, beschränkte und unbeschränkte Petrinetze (Definition)

Man nennt ein Petrinetz ...

- sicher (oder 1-sicher), wenn
  - Für jede Transition t und für jede Stelle s gilt: ${}^\bullet t(s) \leq 1$ und $t^\bullet(s) \leq 1$, also alle Gewichte sind höchstens 1, und
  - für jede erreichbare Markierung m und jede Stelle s gilt, dass $m(s) \leq 1$.

Petrinetze: Beschränktheit

Sichere, beschränkte und unbeschränkte Petrinetze (Definition)

Man nennt ein Petrinetz ...

- sicher (oder 1-sicher), wenn
  - Für jede Transition t und für jede Stelle s gilt: ${}^\bullet t(s) \leq 1$ und $t^\bullet(s) \leq 1$, also alle Gewichte sind höchstens 1, und
  - für jede erreichbare Markierung m und jede Stelle s gilt, dass $m(s) \leq 1$.
- beschränkt, wenn es eine Konstante $c \in \mathbb{N}_0$ gibt, so dass für jede erreichbare Markierung m und jede Stelle s gilt, dass $m(s) \leq c$.

Petrinetze: Beschränktheit

Sichere, beschränkte und unbeschränkte Petrinetze (Definition)

Man nennt ein Petrinetz ...

- sicher (oder 1-sicher), wenn
  - Für jede Transition t und für jede Stelle s gilt: ${}^\bullet t(s) \leq 1$ und $t^\bullet(s) \leq 1$, also alle Gewichte sind höchstens 1, und
  - für jede erreichbare Markierung m und jede Stelle s gilt, dass $m(s) \leq 1$.
- beschränkt, wenn es eine Konstante $c \in \mathbb{N}_0$ gibt, so dass für jede erreichbare Markierung m und jede Stelle s gilt, dass $m(s) \leq c$.
- unbeschränkt, wenn es für jede Konstante $c \in \mathbb{N}_0$ eine erreichbare Markierung m und eine Stelle s gibt mit $m(s) > c$.

Petrinetze: Beschränktheit

Sichere, beschränkte und unbeschränkte Petrinetze (Definition)

Man nennt ein Petrinetz ...

- sicher (oder 1-sicher), wenn
  - Für jede Transition t und für jede Stelle s gilt: ${}^\bullet t(s) \leq 1$ und $t^\bullet(s) \leq 1$, also alle Gewichte sind höchstens 1, und
  - für jede erreichbare Markierung m und jede Stelle s gilt, dass $m(s) \leq 1$.
- beschränkt, wenn es eine Konstante $c \in \mathbb{N}_0$ gibt, so dass für jede erreichbare Markierung m und jede Stelle s gilt, dass $m(s) \leq c$.
- unbeschränkt, wenn es für jede Konstante $c \in \mathbb{N}_0$ eine erreichbare Markierung m und eine Stelle s gibt mit $m(s) > c$.

Beobachtung: Ein Petrinetz ist unbeschränkt genau dann, wenn sein Erreichbarkeitsgraph unendlich groß ist.

Petrinetze: Kausalität, Nebenläufigkeit, Konflikt

Weitere wichtige Begriffe bei Petrinetzen sind Kausalität, Nebenläufigkeit und Konflikt.

Wir beschäftigen uns auch damit etwas genauer.

Petrinetze: Kausalität, Nebenläufigkeit, Konflikt

Weitere wichtige Begriffe bei Petrinetzen sind Kausalität, Nebenläufigkeit und Konflikt.

Wir beschäftigen uns auch damit etwas genauer.

Kausalität (Definition)

In einem Petrinetz nennt man die Transition t_1 eine notwendige Bedingung für das Schalten der Transition t_2 genau dann, wenn für alle Schaltfolgen $\tilde{t}$ gilt:

$$\text{falls } m_0 \left[\tilde{t}\, t_2\right\rangle m \text{ für eine Markierung } m, \text{ dann enthält } \tilde{t} \text{ mit Sicherheit die Transition } t_1.$$

Petrinetze: Kausalität, Nebenläufigkeit, Konflikt

Weitere wichtige Begriffe bei Petrinetzen sind Kausalität, Nebenläufigkeit und Konflikt.

Wir beschäftigen uns auch damit etwas genauer.

Kausalität (Definition)

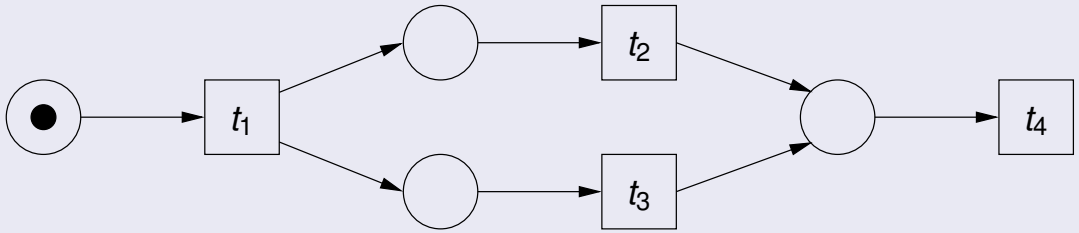
In einem Petrinetz nennt man die Transition t_1 eine notwendige Bedingung für das Schalten der Transition t_2 genau dann, wenn für alle Schaltfolgen $\tilde{t}$ gilt:

$$\text{falls } m_0 \left[\tilde{t}\, t_2\right\rangle m \text{ für eine Markierung } m, \text{ dann enthält } \tilde{t} \text{ mit Sicherheit die Transition } t_1.$$

Bezüglich des Erreichbarkeitsgraphen bedeutet dies, dass jeder Knoten, von dem aus es einen mit t_2 beschrifteten Übergang gibt, nur über Wege erreichbar ist, in denen t_1 vorkommt.

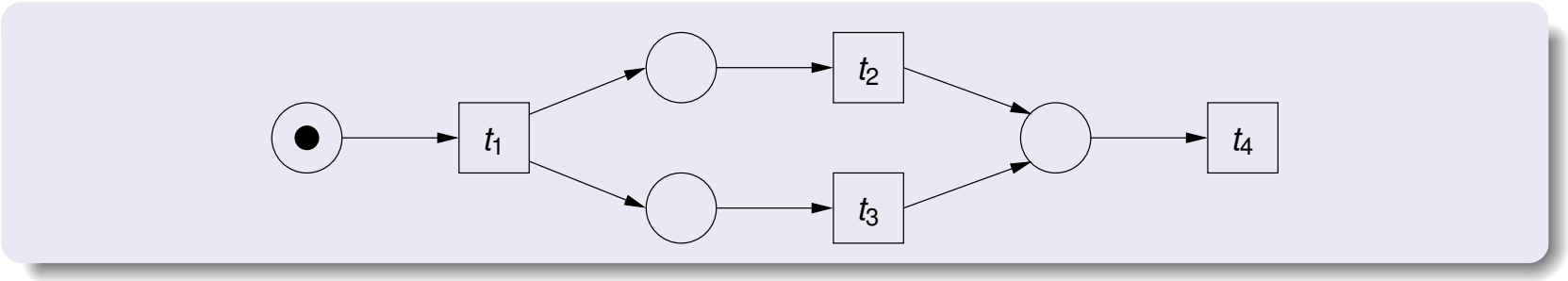
Petrinetze: Kausalität, Nebenläufigkeit, Konflikt

Beispiel für Kausalität:



Petrinetze: Kausalität, Nebenläufigkeit, Konflikt

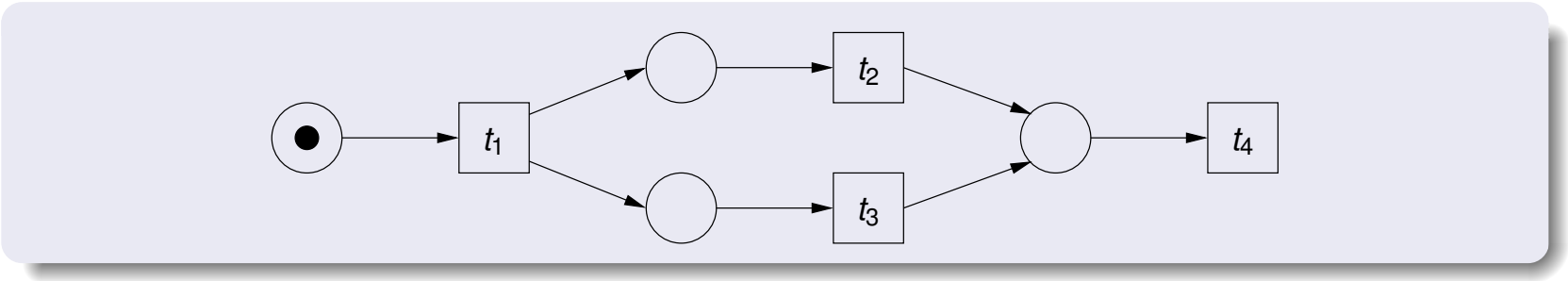
Beispiel für Kausalität:



- Hier ist t_1 eine notwendige Bedingung für t_4.

Petrinetze: Kausalität, Nebenläufigkeit, Konflikt

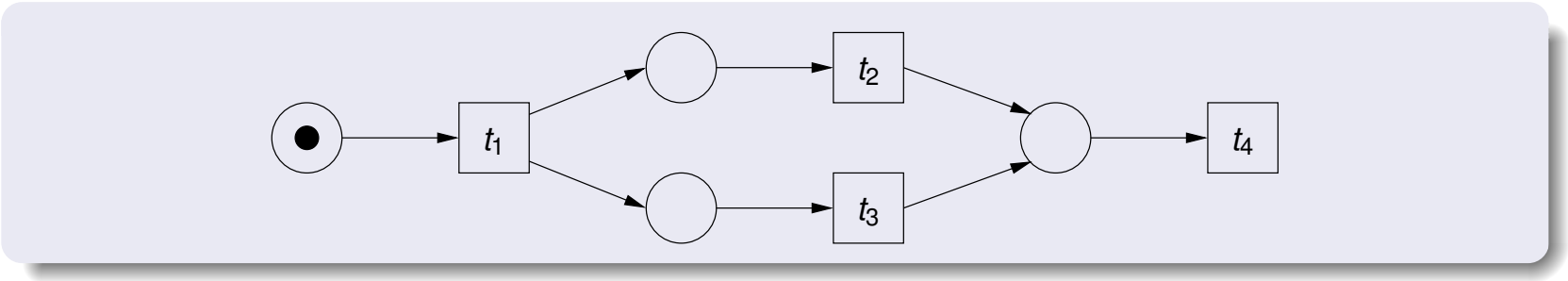
Beispiel für Kausalität:



- Hier ist t_1 eine notwendige Bedingung für t_4.
- Aber t_2 ist hier keine notwendige Bedingung für t_4. Denn nicht jede Schaltfolge, die zu t_4 führt, enthält t_2 (z.B. $\tilde{t} = t_1\, t_3$).

Petrinetze: Kausalität, Nebenläufigkeit, Konflikt

Beispiel für Kausalität:



- Hier ist t_1 eine notwendige Bedingung für t_4.
- Aber t_2 ist hier keine notwendige Bedingung für t_4. Denn nicht jede Schaltfolge, die zu t_4 führt, enthält t_2 (z.B. $\tilde{t} = t_1\, t_3$).

  Analoges gilt für t_3.

Petrinetze: Kausalität, Nebenläufigkeit, Konflikt

Transitivität der Kausalität

Wenn t_1 eine notwendige Bedingung für t_2 ist, und t_2 eine notwendige Bedingung für t_3 ist, dann ist t_1 eine notwendige Bedingung für t_3.

Petrinetze: Kausalität, Nebenläufigkeit, Konflikt

Nebenläufigkeit (Definition)

Die Transitionen $t_1, \ldots, t_n$ einer Menge $T' \subseteq T$ nennt man für die Markierung m nebenläufig aktiviert, wenn

$$ {}^\bullet t_1 \oplus \cdots \oplus {}^\bullet t_n \leq m. $$

Das heißt, wenn die Markierung m genug Marken enthält, um alle Transitionen aus T' „gleichzeitig“ zu feuern.

Petrinetze: Kausalität, Nebenläufigkeit, Konflikt

Nebenläufigkeit (Definition)

Die Transitionen $t_1, \ldots, t_n$ einer Menge $T' \subseteq T$ nennt man für die Markierung m nebenläufig aktiviert, wenn

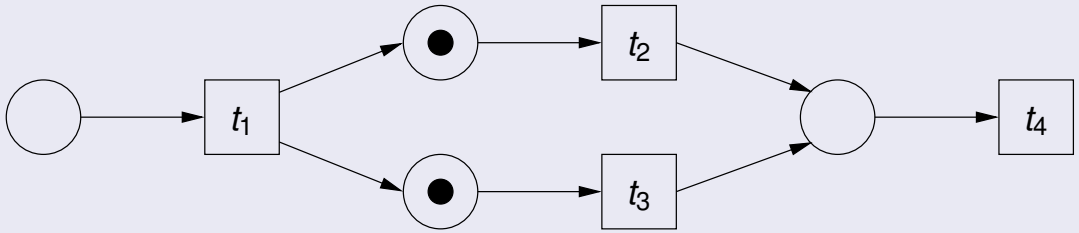
$$ {}^\bullet t_1 \oplus \cdots \oplus {}^\bullet t_n \leq m. $$

Das heißt, wenn die Markierung m genug Marken enthält, um alle Transitionen aus T' „gleichzeitig“ zu feuern.

Beobachtung: Wenn die Transitionen einer Menge T' für die Markierung m nebenläufig aktiviert sind, so ist dies auch für jede Teilmenge von T' der Fall.

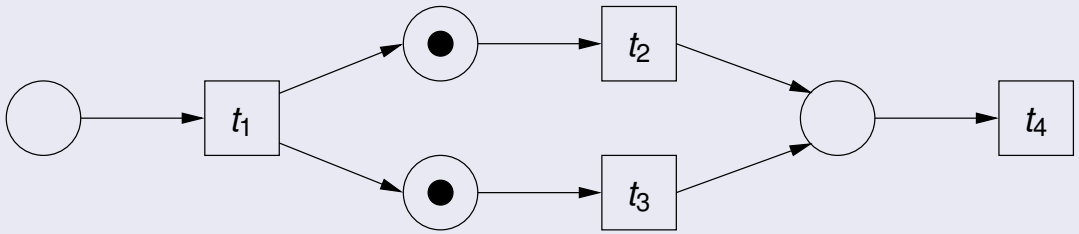
Petrinetze: Kausalität, Nebenläufigkeit, Konflikt

Beispiele für Nebenläufigkeit:



Petrinetze: Kausalität, Nebenläufigkeit, Konflikt

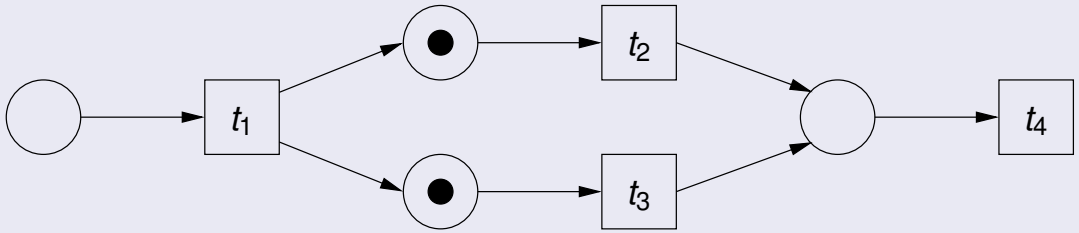
Beispiele für Nebenläufigkeit:



Die Transitionen t_2 und t_3 sind für die hier gezeigte Markierung nebenläufig aktiviert.

Petrinetze: Kausalität, Nebenläufigkeit, Konflikt

Beispiele für Nebenläufigkeit:

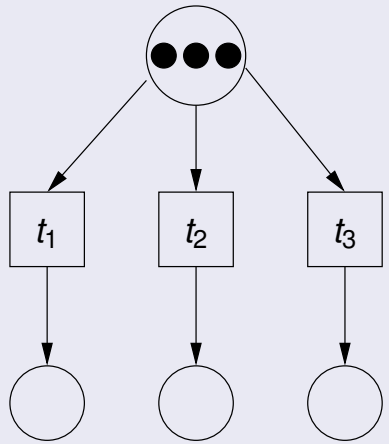


Die Transitionen t_2 und t_3 sind für die hier gezeigte Markierung nebenläufig aktiviert.

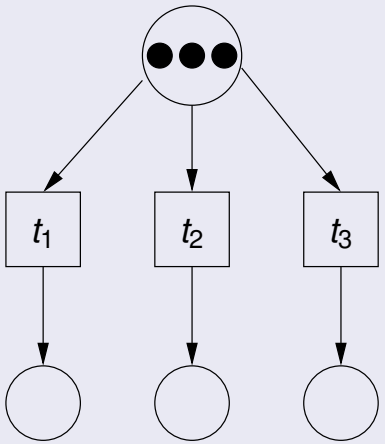
Denn:

$$\begin{aligned} {}^\bullet t_2 &= (0,1,0,0) \\ {}^\bullet t_3 &= (0,0,1,0) \\ {}^\bullet t_2 \oplus {}^\bullet t_3 &\leq (0,1,1,0) \end{aligned}$$

Petrinetze: Kausalität, Nebenläufigkeit, Konflikt

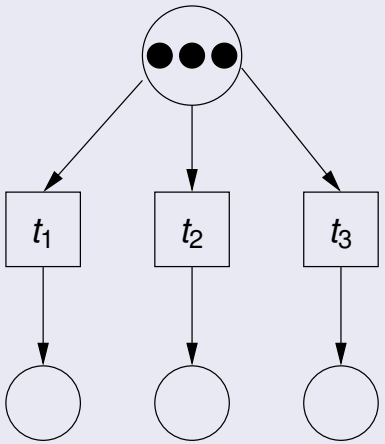


Petrinetze: Kausalität, Nebenläufigkeit, Konflikt



Die Transitionen t_1, t_2 und t_3 sind für die hier gezeigte Markierung nebenläufig aktiviert.

Petrinetze: Kausalität, Nebenläufigkeit, Konflikt

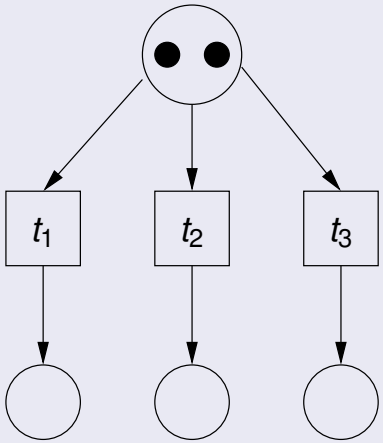


Die Transitionen t_1, t_2 und t_3 sind für die hier gezeigte Markierung nebenläufig aktiviert.

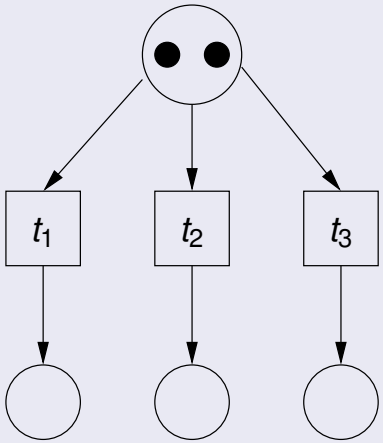
Denn:

$$\begin{aligned} {}^\bullet t_1 &= (1,0,0,0) \\ {}^\bullet t_2 &= (1,0,0,0) \\ {}^\bullet t_3 &= (1,0,0,0) \\ {}^\bullet t_1 \oplus {}^\bullet t_2 \oplus {}^\bullet t_3 &\leq (3,0,0,0) \end{aligned}$$

Petrinetze: Kausalität, Nebenläufigkeit, Konflikt

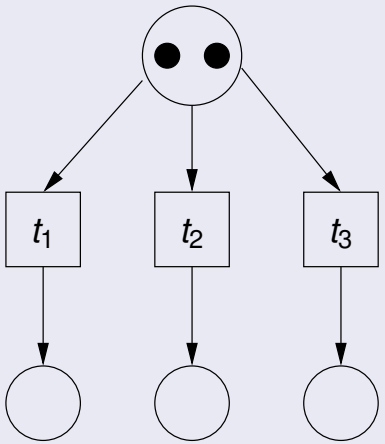


Petrinetze: Kausalität, Nebenläufigkeit, Konflikt



Die Transitionen der Mengen $\{t_1, t_2\}$, $\{t_2, t_3\}$ und $\{t_1, t_3\}$ sind für die hier gezeigte Markierung jeweils nebenläufig aktiviert. Dies gilt jedoch nicht für die Menge $\{t_1, t_2, t_3\}$ insgesamt.

Petrinetze: Kausalität, Nebenläufigkeit, Konflikt



Die Transitionen der Mengen $\{t_1, t_2\}$, $\{t_2, t_3\}$ und $\{t_1, t_3\}$ sind für die hier gezeigte Markierung jeweils nebenläufig aktiviert. Dies gilt jedoch nicht für die Menge $\{t_1, t_2, t_3\}$ insgesamt.

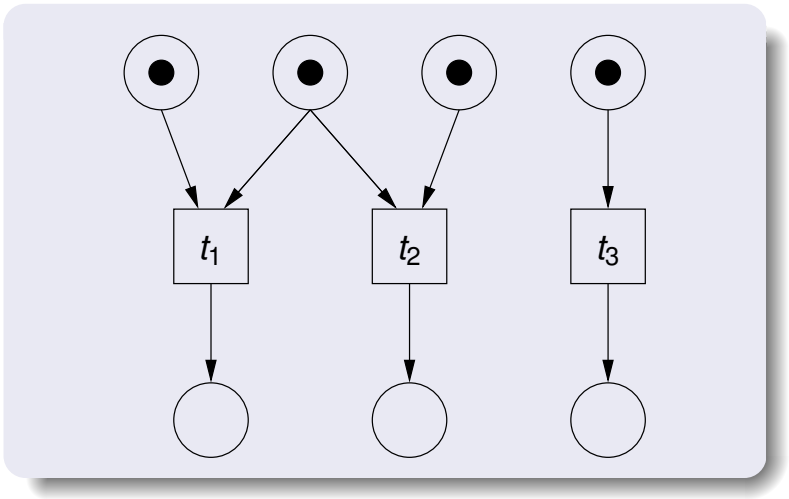
Denn:

$$\bullet t_1 = \bullet t_2 = \bullet t_3 = (1,0,0,0)$$

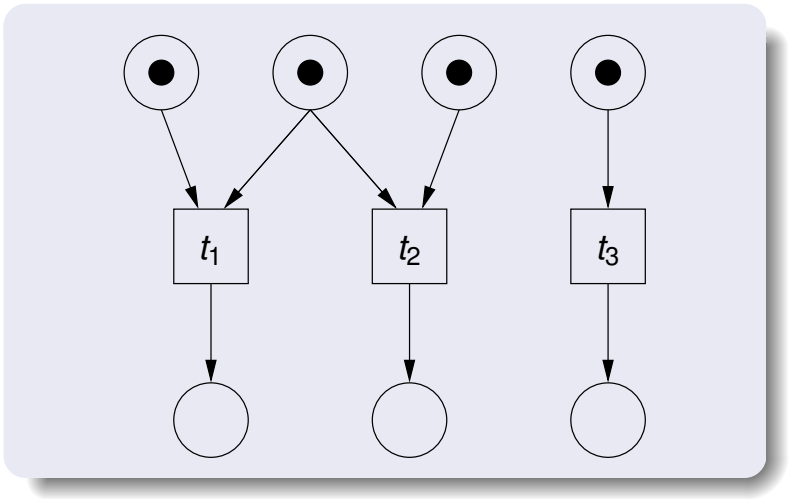
$$\bullet t_1 \oplus \bullet t_2 = \bullet t_2 \oplus \bullet t_3 = \bullet t_1 \oplus \bullet t_3 \leq (2,0,0,0)$$

$$\bullet t_1 \oplus \bullet t_2 \oplus \bullet t_3 \not\leq (2,0,0,0)$$

Petrinetze: Kausalität, Nebenläufigkeit, Konflikt

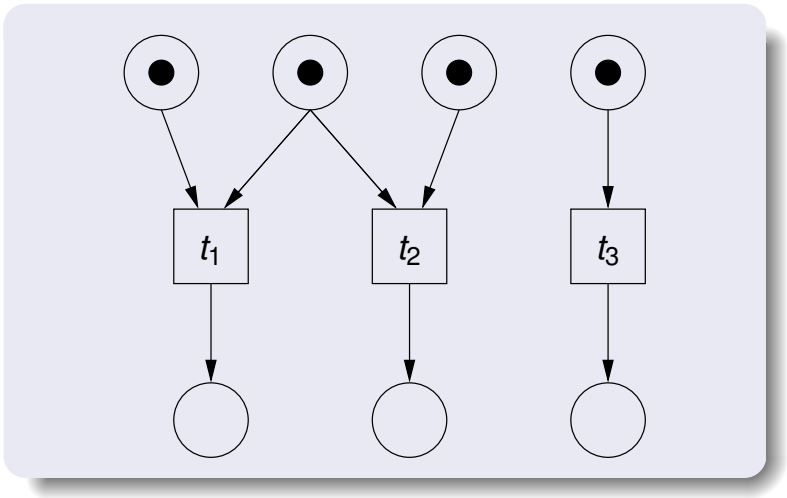


Petrinetze: Kausalität, Nebenläufigkeit, Konflikt



Die Transitionen der Mengen $\{t_1, t_3\}$ und $\{t_2, t_3\}$ sind für die hier gezeigte Markierung jeweils nebenläufig aktiviert. Dies gilt jedoch nicht für die Mengen $\{t_1, t_2\}$ und $\{t_1, t_2, t_3\}$.

Petrinetze: Kausalität, Nebenläufigkeit, Konflikt



Die Transitionen der Mengen $\{t_1, t_3\}$ und $\{t_2, t_3\}$ sind für die hier gezeigte Markierung jeweils nebenläufig aktiviert. Dies gilt jedoch nicht für die Mengen $\{t_1, t_2\}$ und $\{t_1, t_2, t_3\}$.

Dieses Beispiel zeigt auch, dass Nebenläufigkeit nicht transitiv ist: für die angegebene Markierung ist t_1 nebenläufig aktiviert zu t_3, und t_3 ist nebenläufig aktiviert zu t_2, jedoch sind t_1 und t_2 nicht nebenläufig aktiviert.

Petrinetze: Kausalität, Nebenläufigkeit, Konflikt

Konsequenzen von Nebenläufigkeit

Wenn die Transitionen einer Menge T' für eine Markierung m nebenläufig aktiviert sind, so ist jede Anordnung dieser Transitionen eine Schaltfolge ausgehend von m.

Petrinetze: Kausalität, Nebenläufigkeit, Konflikt

Konsequenzen von Nebenläufigkeit

Wenn die Transitionen einer Menge T' für eine Markierung m nebenläufig aktiviert sind, so ist jede Anordnung dieser Transitionen eine Schaltfolge ausgehend von m.

Das heißt, für jede Sequenz $\tilde{t}$, in der jede Transition aus T' genau einmal vorkommt, gibt es eine Markierung m' mit $m\,[\tilde{t}\rangle\,m'$.

Petrinetze: Kausalität, Nebenläufigkeit, Konflikt

Konsequenzen von Nebenläufigkeit

Wenn die Transitionen einer Menge T' für eine Markierung m nebenläufig aktiviert sind, so ist jede Anordnung dieser Transitionen eine Schaltfolge ausgehend von m.

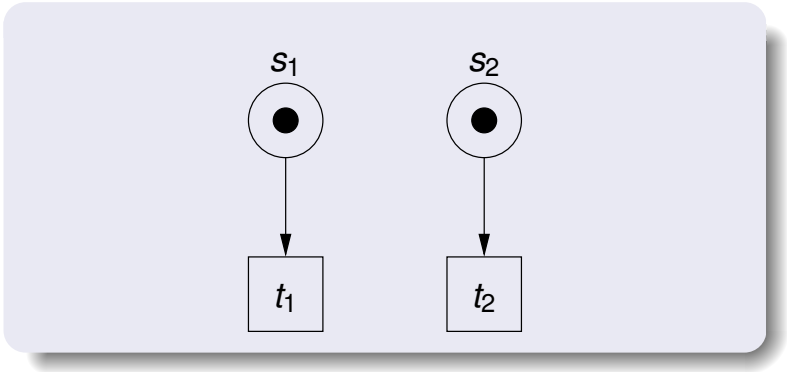
Das heißt, für jede Sequenz $\tilde{t}$, in der jede Transition aus T' genau einmal vorkommt, gibt es eine Markierung m' mit $m\,[\tilde{t}\rangle\,m'$.

Und diese Markierung m' ist durch T' eindeutig bestimmt (also unabhängig von $\tilde{t}$).

Petrinetze: Kausalität, Nebenläufigkeit, Konflikt

Nebenläufig aktivierte Transitionen führen daher in Erreichbarkeitsgraphen zu Strukturen, die die Form eines Quadrats (oft Diamond genannt) oder (höherdimensionalen) Würfels haben.

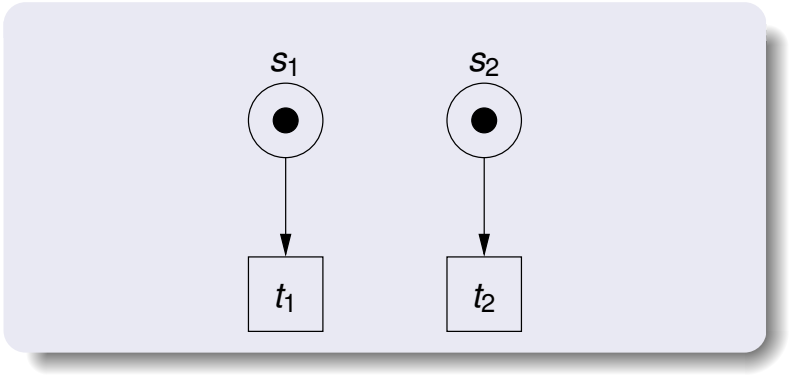
Beispiel für so ein Quadrat:

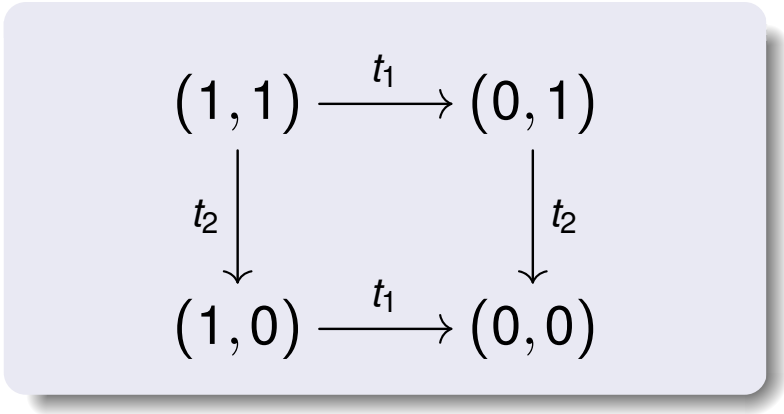


Petrinetze: Kausalität, Nebenläufigkeit, Konflikt

Nebenläufig aktivierte Transitionen führen daher in Erreichbarkeitsgraphen zu Strukturen, die die Form eines Quadrats (oft Diamond genannt) oder (höherdimensionalen) Würfels haben.

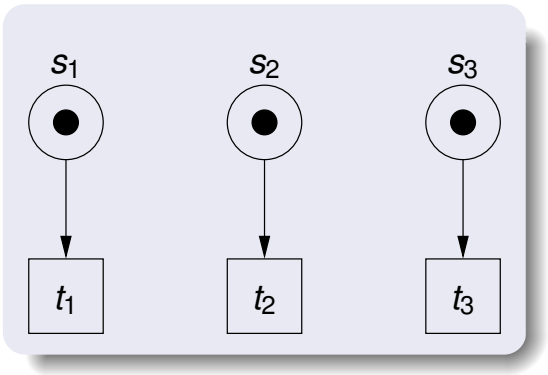
Beispiel für so ein Quadrat:





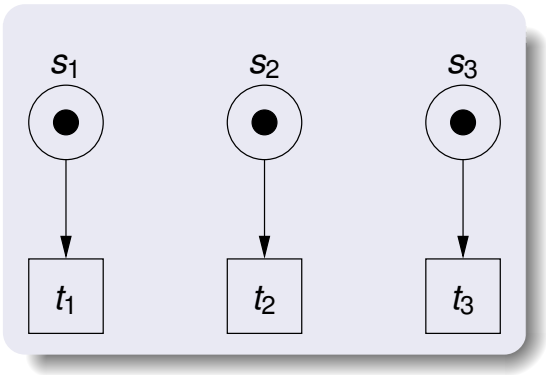
Petrinetze: Kausalität, Nebenläufigkeit, Konflikt

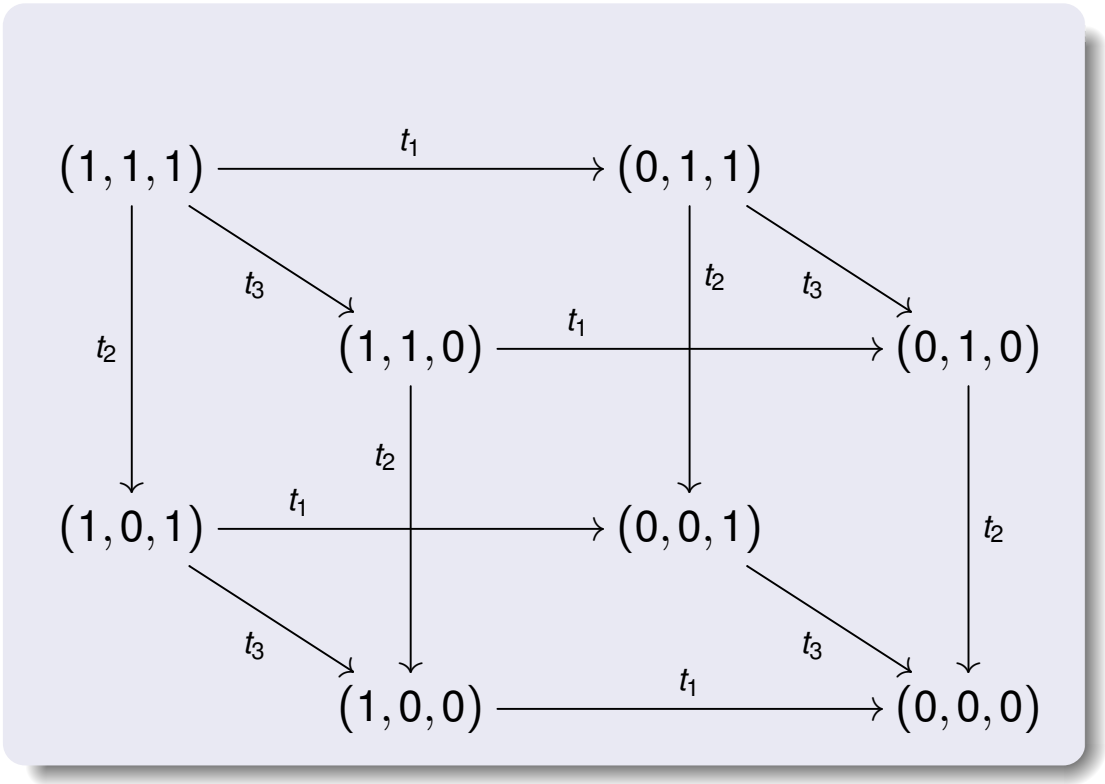
Beispiel für Entstehen eines Würfels:



Petrinetze: Kausalität, Nebenläufigkeit, Konflikt

Beispiel für Entstehen eines Würfels:





Petrinetze: Kausalität, Nebenläufigkeit, Konflikt

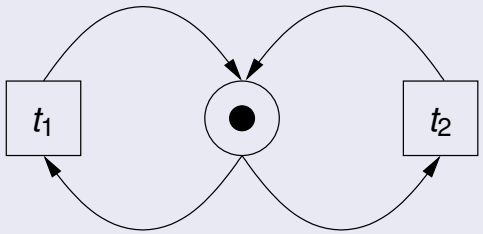
Frage: Wenn ausgehend von einer Markierung m jede Anordnung der Transitionen einer Menge T' eine Schaltfolge darstellt, sind dann die Transitionen aus T' für m nebenläufig aktiviert?

Petrinetze: Kausalität, Nebenläufigkeit, Konflikt

Frage: Wenn ausgehend von einer Markierung m jede Anordnung der Transitionen einer Menge T' eine Schaltfolge darstellt, sind dann die Transitionen aus T' für m nebenläufig aktiviert?

Nein, nicht unbedingt!

Gegenbeispiel:

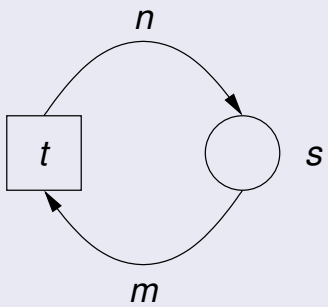


Petrinetze: Kausalität, Nebenläufigkeit, Konflikt

Schlinge (Definition)

Eine Schlinge (oder Schleife) in einem Petrinetz besteht aus einer Transition t und einer Stelle s mit ${}^\bullet t(s) > 0$ und $t^\bullet(s) > 0$.

Grafisch:

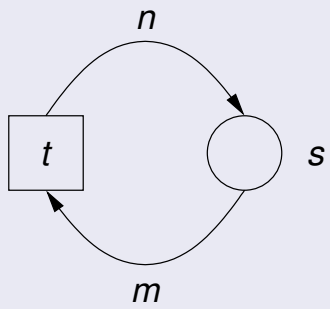


Petrinetze: Kausalität, Nebenläufigkeit, Konflikt

Schlinge (Definition)

Eine Schlinge (oder Schleife) in einem Petrinetz besteht aus einer Transition t und einer Stelle s mit ${}^\bullet t(s) > 0$ und $t^\bullet(s) > 0$.

Grafisch:



Für schlingenfreie Petrinetze gilt:

Seien eine Markierung m und eine Menge T' von Transitionen gegeben, so dass jede Anordnung dieser Transitionen von m ausgehend schaltbar ist.

Dann sind die Transitionen aus T' für m nebenläufig aktiviert.

Petrinetze: Kausalität, Nebenläufigkeit, Konflikt

Konflikt (Definition)

Zwei Transitionen $t, t' \in T$ stehen für die Markierung m in Konflikt genau dann, wenn:

- t und t' sind beide für m aktiviert und
- t und t' sind für m nicht nebenläufig aktiviert.

Petrinetze: Kausalität, Nebenläufigkeit, Konflikt

Konflikt (Definition)

Zwei Transitionen $t, t' \in T$ stehen für die Markierung m in Konflikt genau dann, wenn:

- t und t' sind beide für m aktiviert und
- t und t' sind für m nicht nebenläufig aktiviert.

Anschaulich: Jede einzelne der beiden Transitionen könnte schalten, aber nicht tatsächlich beide „gleichzeitig“.

Petrinetze: Kausalität, Nebenläufigkeit, Konflikt

Konflikt (Definition)

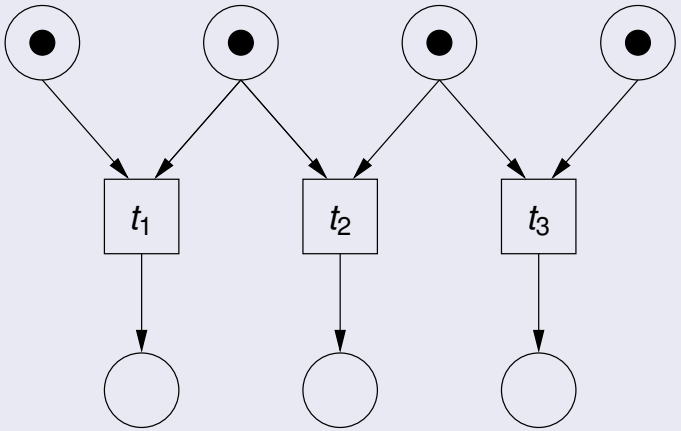
Zwei Transitionen $t, t' \in T$ stehen für die Markierung m in Konflikt genau dann, wenn:

- t und t' sind beide für m aktiviert und
- t und t' sind für m nicht nebenläufig aktiviert.

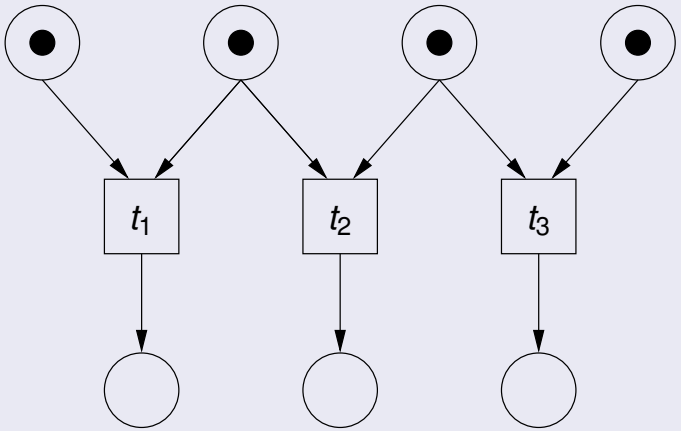
Anschaulich: Jede einzelne der beiden Transitionen könnte schalten, aber nicht tatsächlich beide „gleichzeitig“.

Das liegt immer daran, dass sie eine gemeinsame Stelle in den Vorbedingungen haben. Das heißt, es gibt eine Stelle s mit ${}^\bullet t(s) \geq 1$ und ${}^\bullet t'(s) \geq 1$.

Petrinetze: Kausalität, Nebenläufigkeit, Konflikt



Petrinetze: Kausalität, Nebenläufigkeit, Konflikt



Für die hier gezeigte Markierung steht t_1 in Konflikt mit t_2.

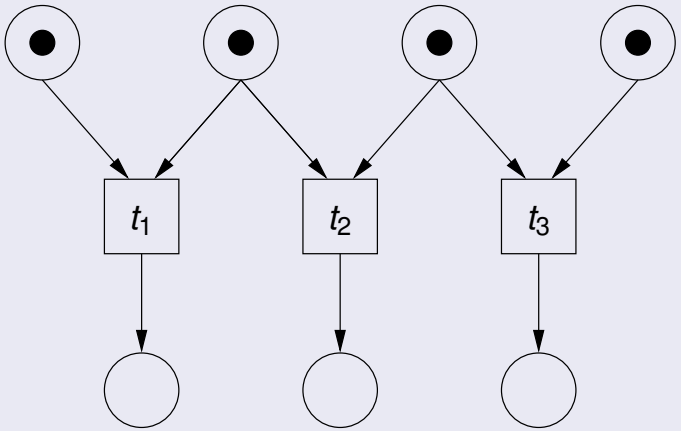
Denn:

$$(1,1,0,0,0,0,0) \leq (1,1,1,1,0,0,0)$$

$$(0,1,1,0,0,0,0) \leq (1,1,1,1,0,0,0)$$

$$(1,1,0,0,0,0,0) \oplus (0,1,1,0,0,0,0) \not\leq (1,1,1,1,0,0,0)$$

Petrinetze: Kausalität, Nebenläufigkeit, Konflikt



Für die hier gezeigte Markierung steht t_1 in Konflikt mit t_2.

Denn:

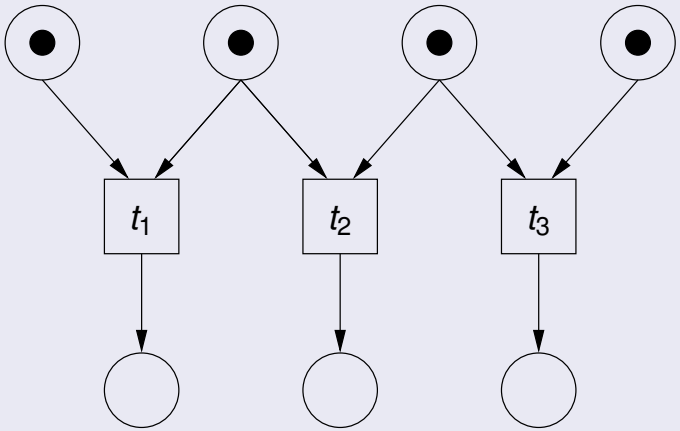
$$(1,1,0,0,0,0,0) \leq (1,1,1,1,0,0,0)$$

$$(0,1,1,0,0,0,0) \leq (1,1,1,1,0,0,0)$$

$$(1,1,0,0,0,0,0) \oplus (0,1,1,0,0,0,0) \not\leq (1,1,1,1,0,0,0)$$

Außerdem steht t_2 in Konflikt mit t_3.

Petrinetze: Kausalität, Nebenläufigkeit, Konflikt



Für die hier gezeigte Markierung steht t_1 in Konflikt mit t_2.

Denn:

$$(1,1,0,0,0,0,0) \leq (1,1,1,1,0,0,0)$$

$$(0,1,1,0,0,0,0) \leq (1,1,1,1,0,0,0)$$

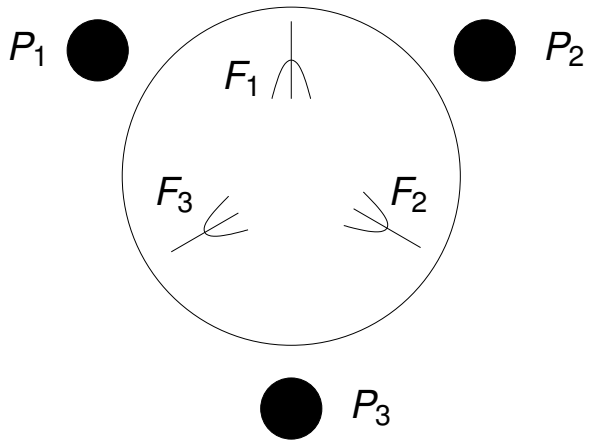
$$(1,1,0,0,0,0,0) \oplus (0,1,1,0,0,0,0) \not\leq (1,1,1,1,0,0,0)$$

Außerdem steht t_2 in Konflikt mit t_3. Jedoch steht t_1 nicht in Konflikt mit t_3 (keine Transitiviät der Konfliktrelation).

Beispiel: Dining Philosophers

Wir betrachten nochmals das Beispiel der Dining Philosophers (speisende Philosophen):

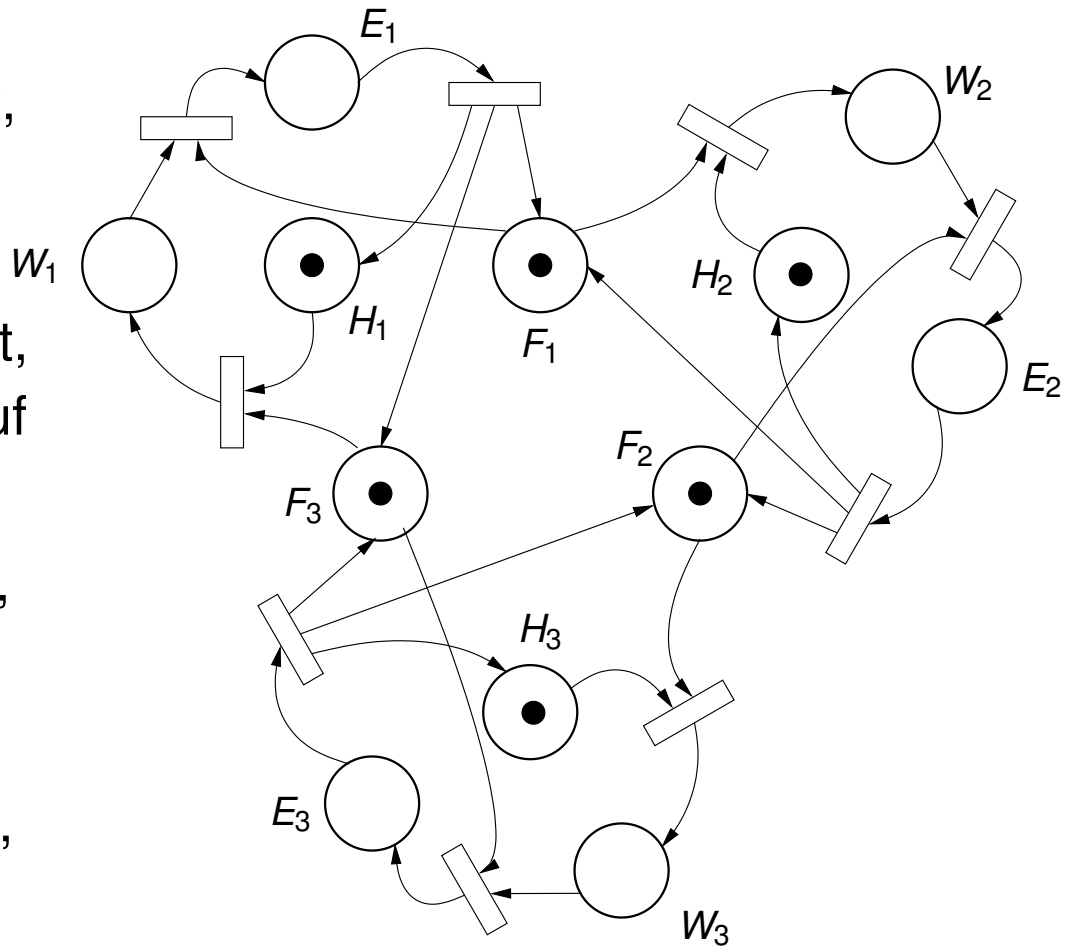
- Es sitzen drei Philosophen P_i um einen runden Tisch, zwischen je zwei Philosophen liegt eine Gabel (fork) F_i.
- Philosophen werden von Zeit zu Zeit hungrig H_i und benötigen zum Essen E_i beide benachbarte Gabeln.
- Jeder Philosoph nimmt zu einem beliebigen Zeitpunkt beide Gabeln nacheinander auf (die rechte zuerst), isst und legt anschließend beide Gabeln wieder zurück.



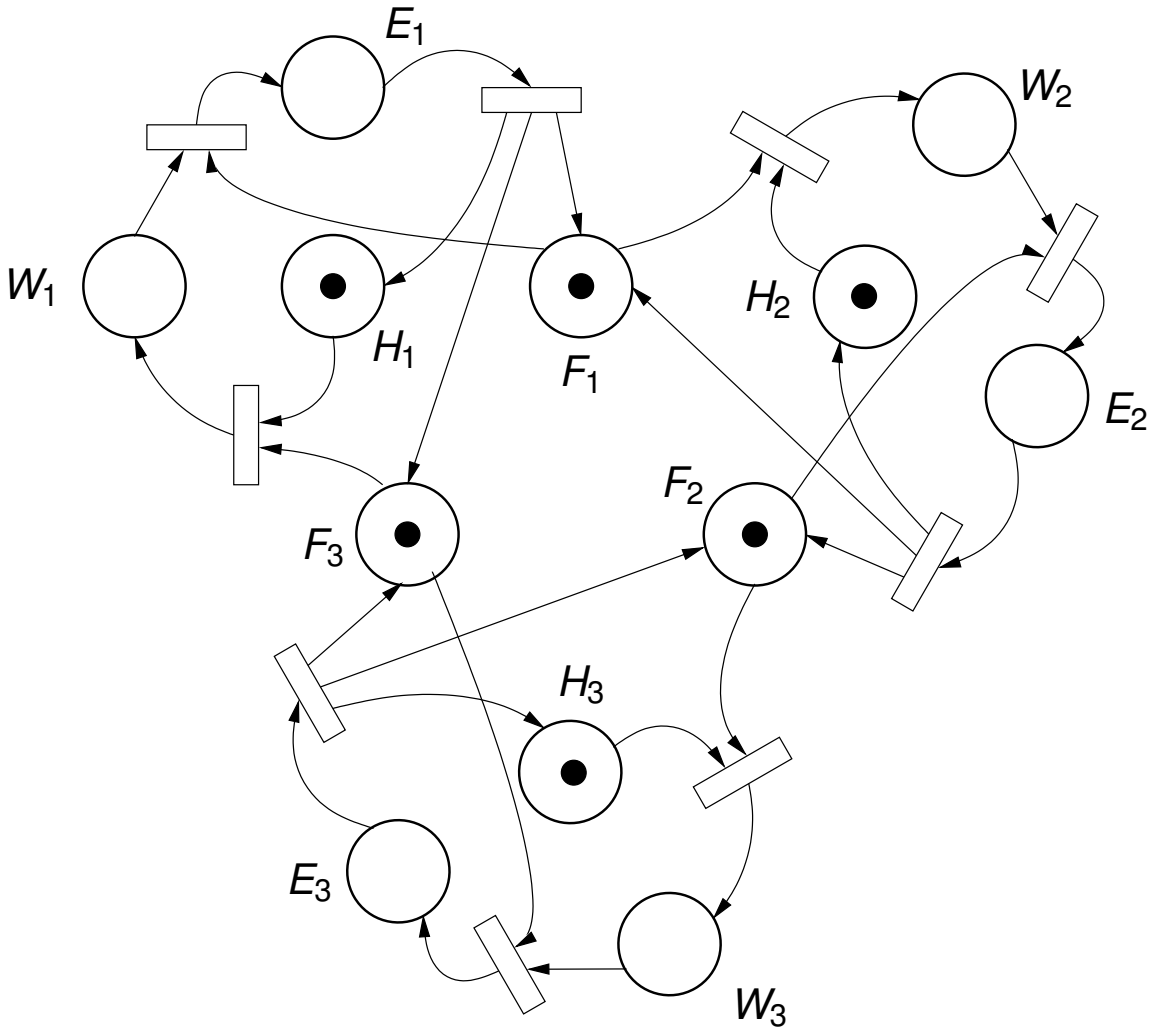
Beispiel: Dining Philosophers

Modellierung als Petrinetz:

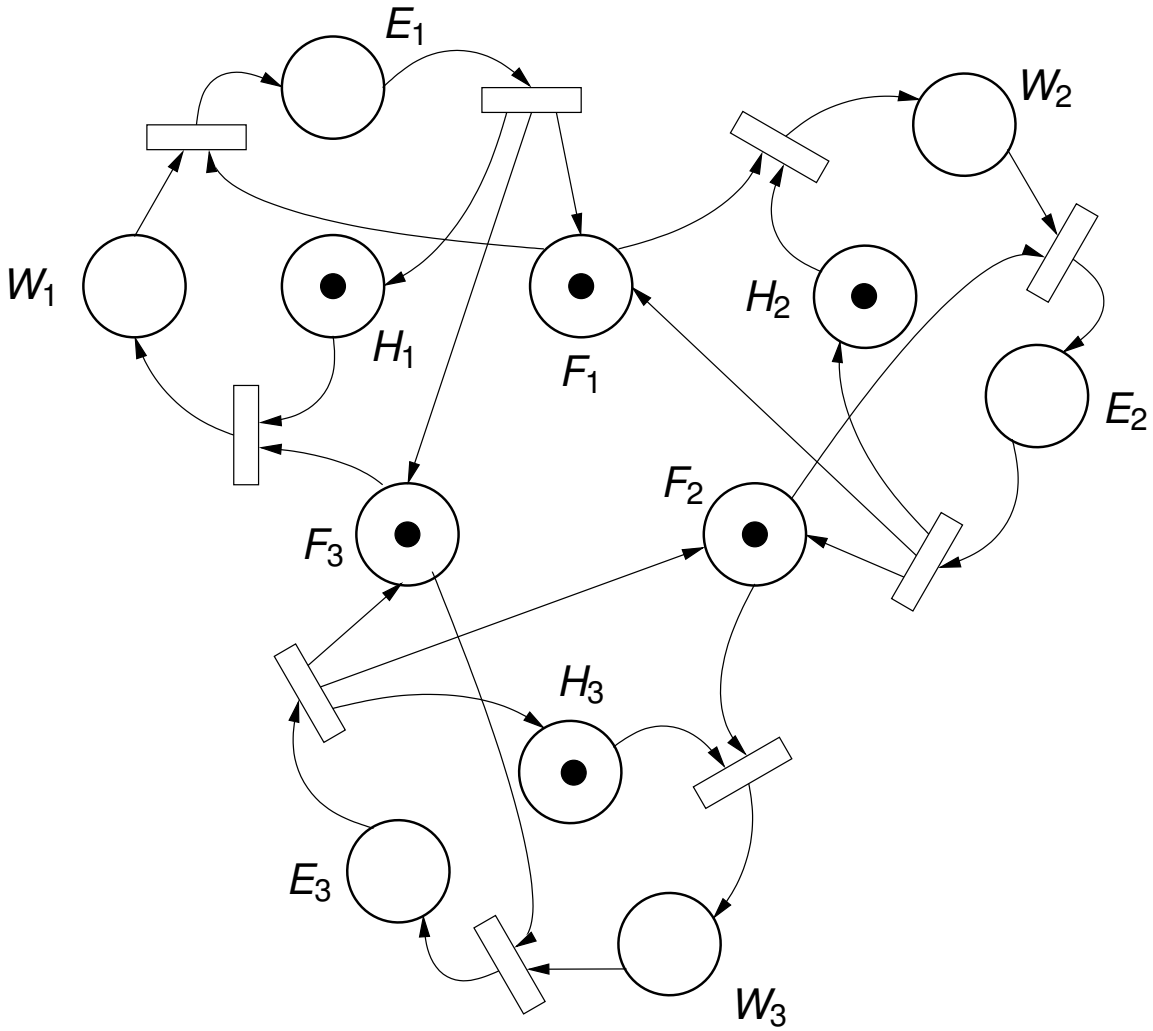
- Marke bei H_i bedeutet, Philosoph P_i ist hungrig.
- Marke bei W_i bedeutet, Philosoph P_i wartet auf linke Gabel.
- Marke bei F_i bedeutet, die Gabel F_i liegt auf dem Tisch.
- Marke bei E_i bedeutet, Philosoph P_i isst.



Beispiel: Dining Philosophers

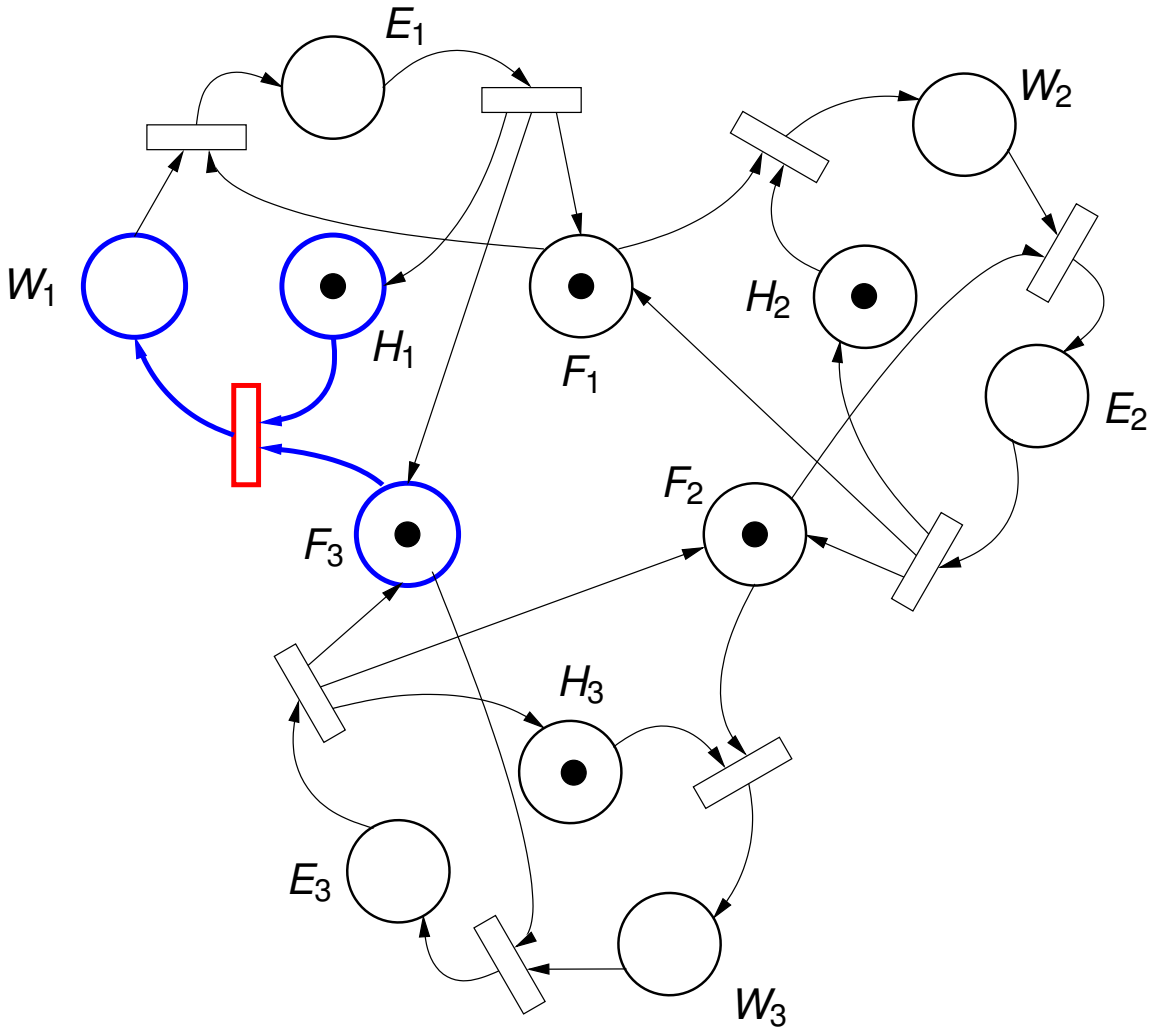


Beispiel: Dining Philosophers

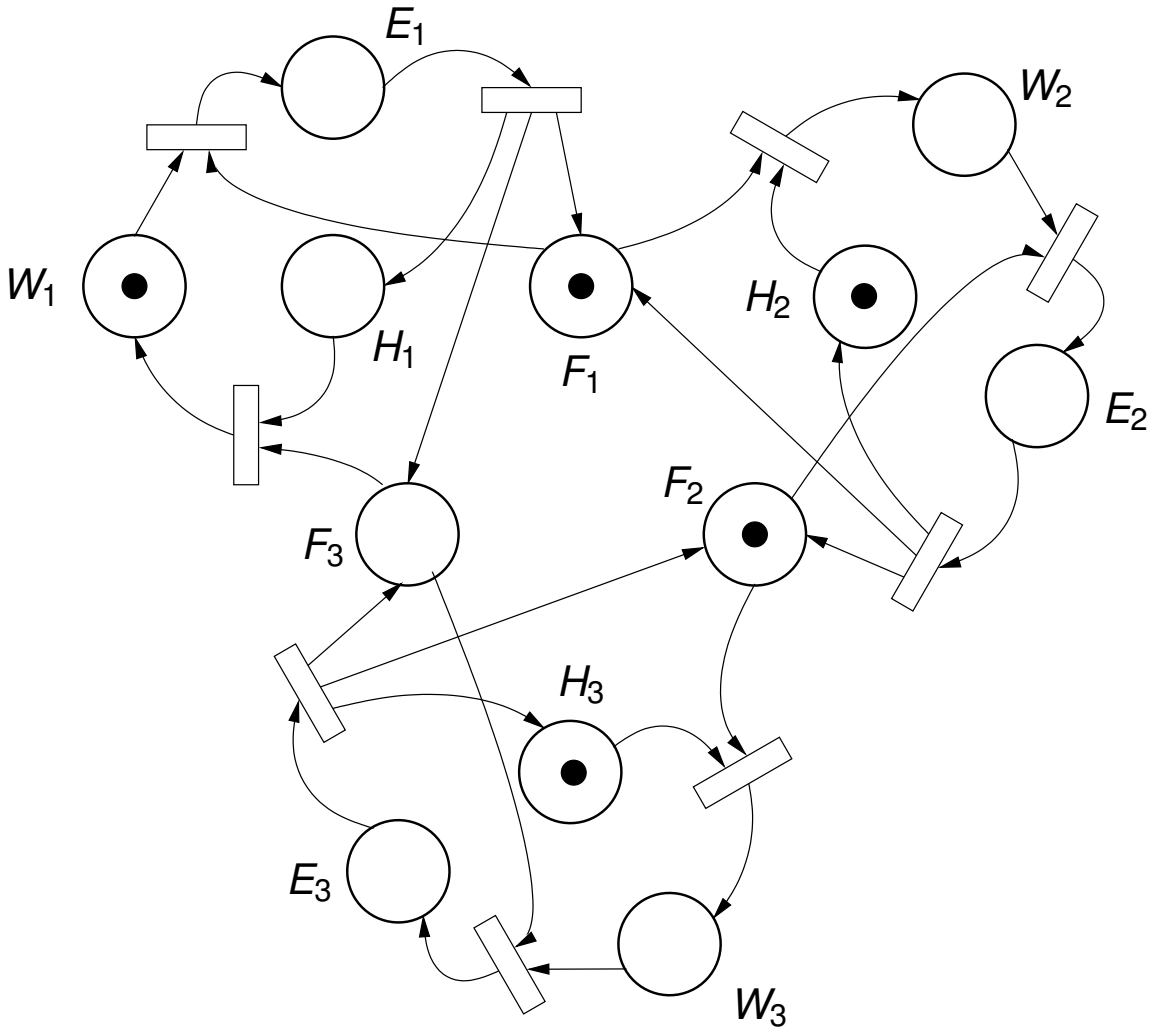


- Der erste Philosoph P_1 möchte essen.

Beispiel: Dining Philosophers

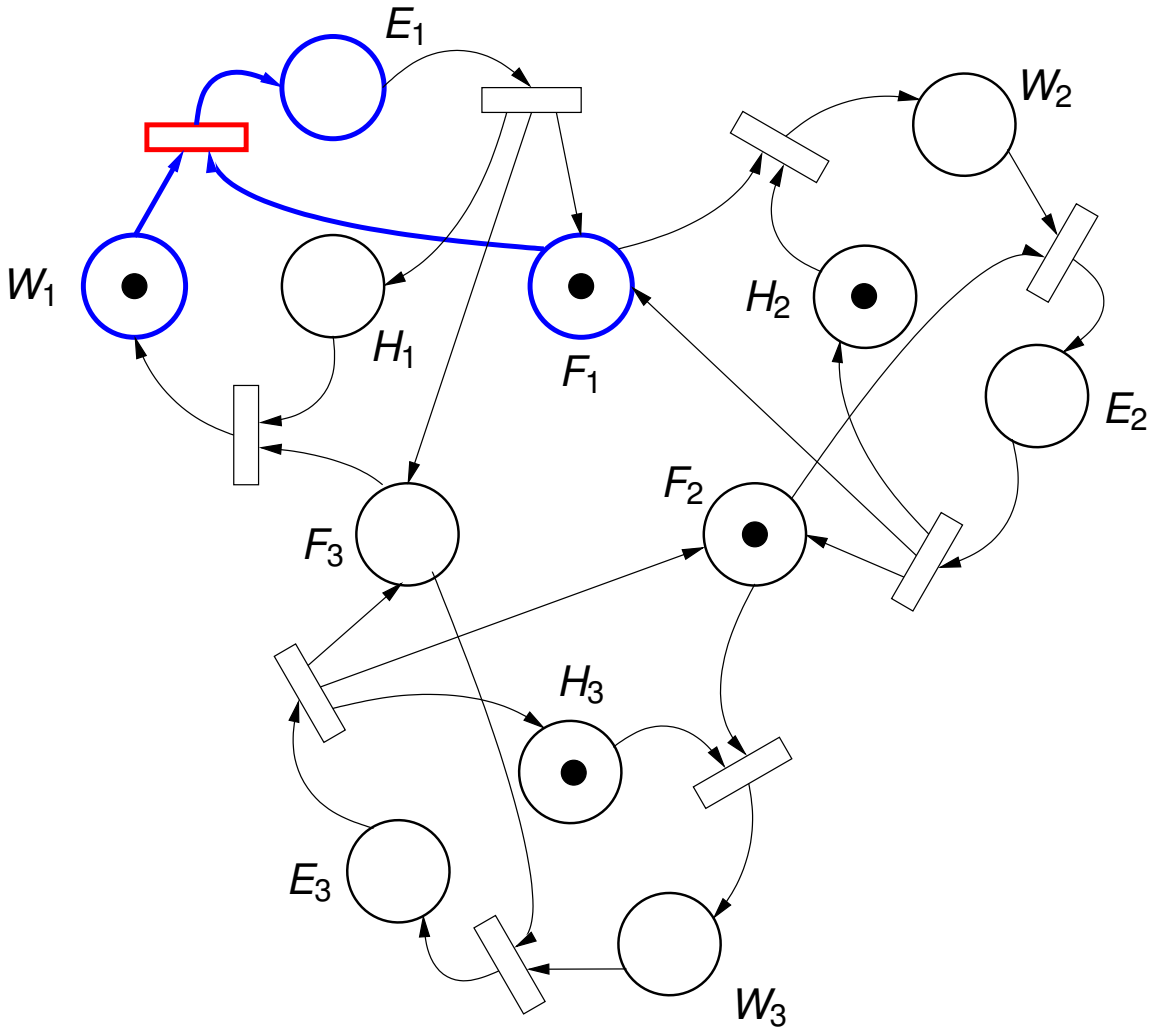


Beispiel: Dining Philosophers

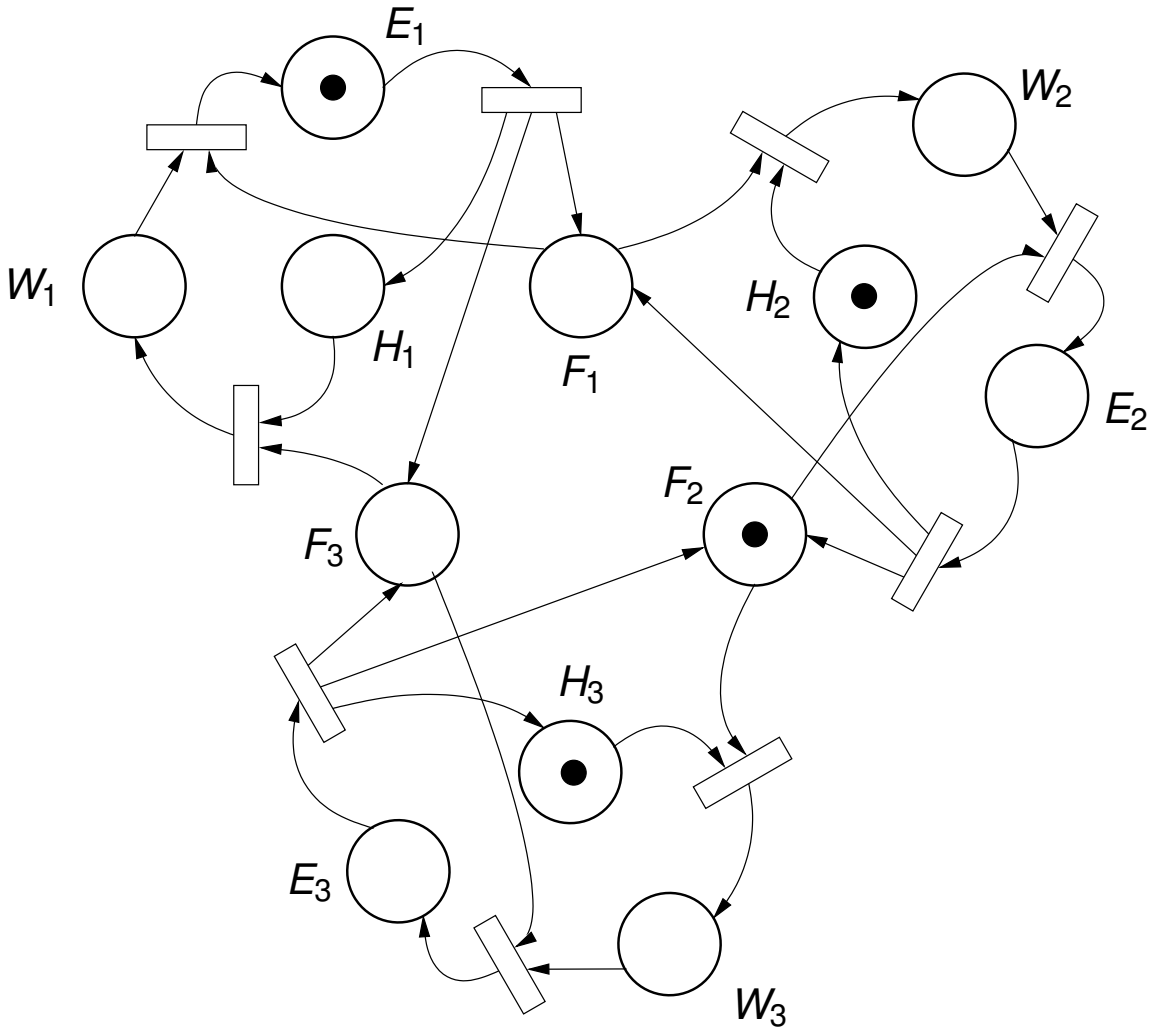


- P_1 nimmt die rechte Gabel F_3.

Beispiel: Dining Philosophers

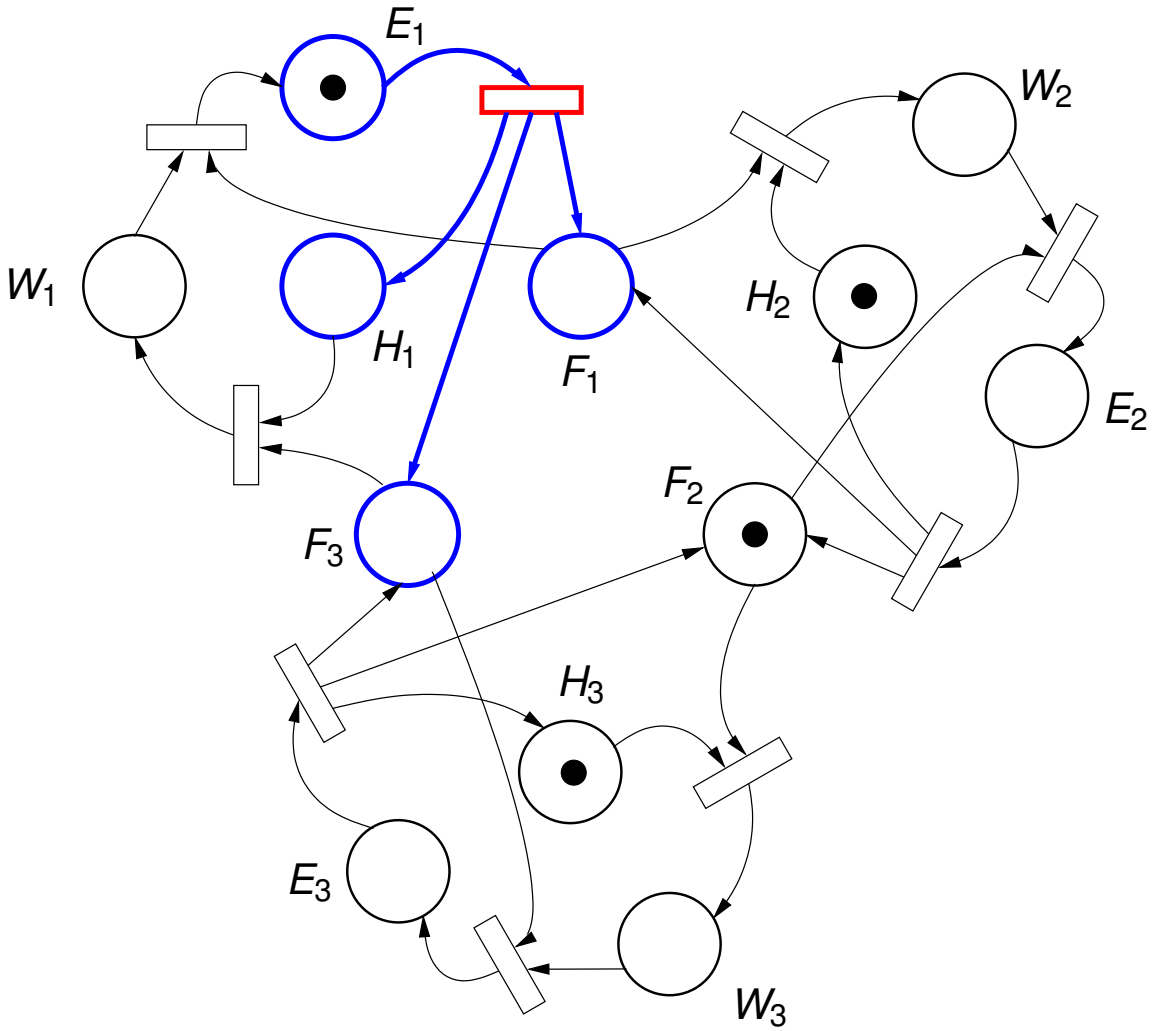


Beispiel: Dining Philosophers

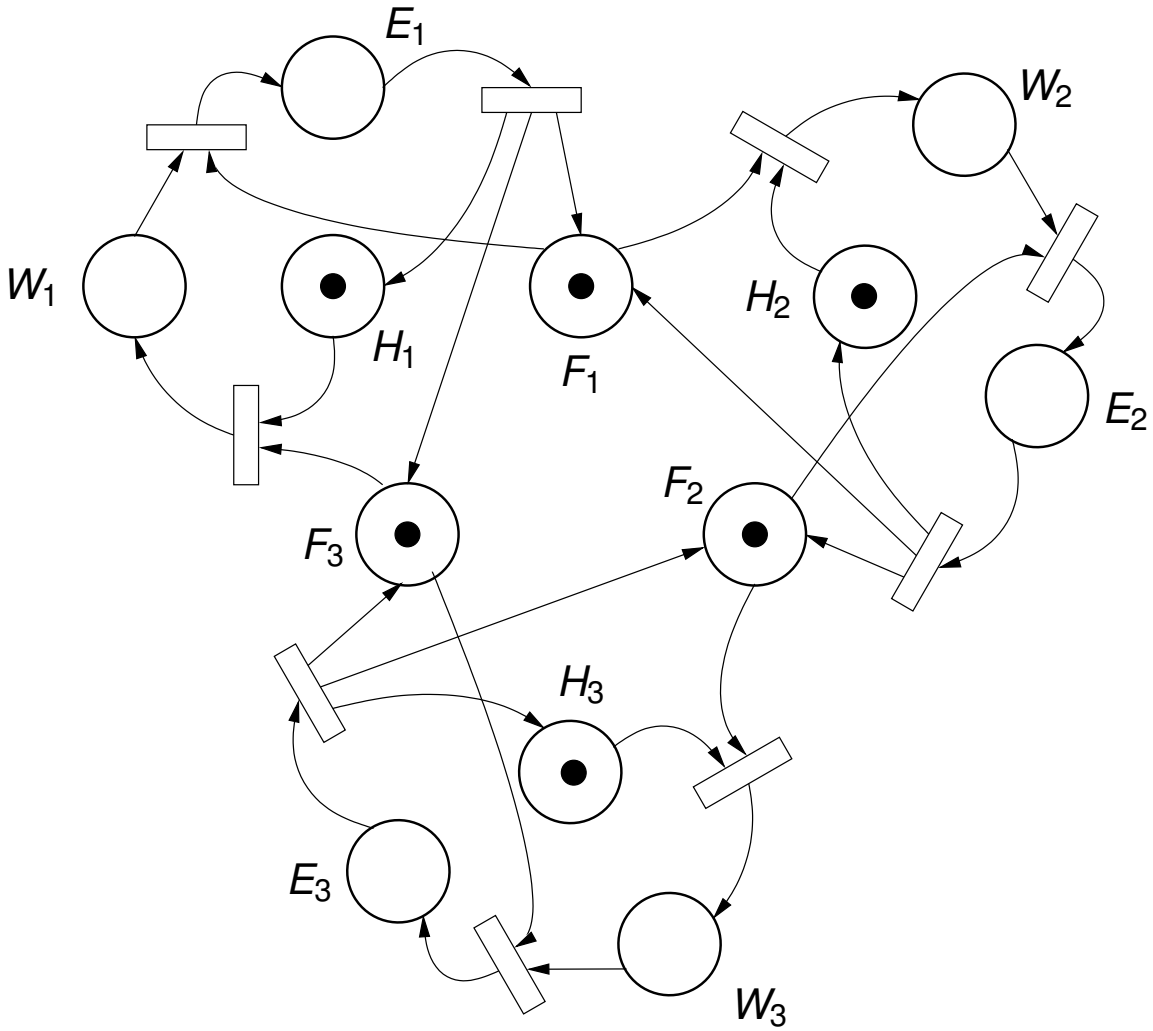


- P_1 nimmt die linke Gabel F_1 und isst.

Beispiel: Dining Philosophers

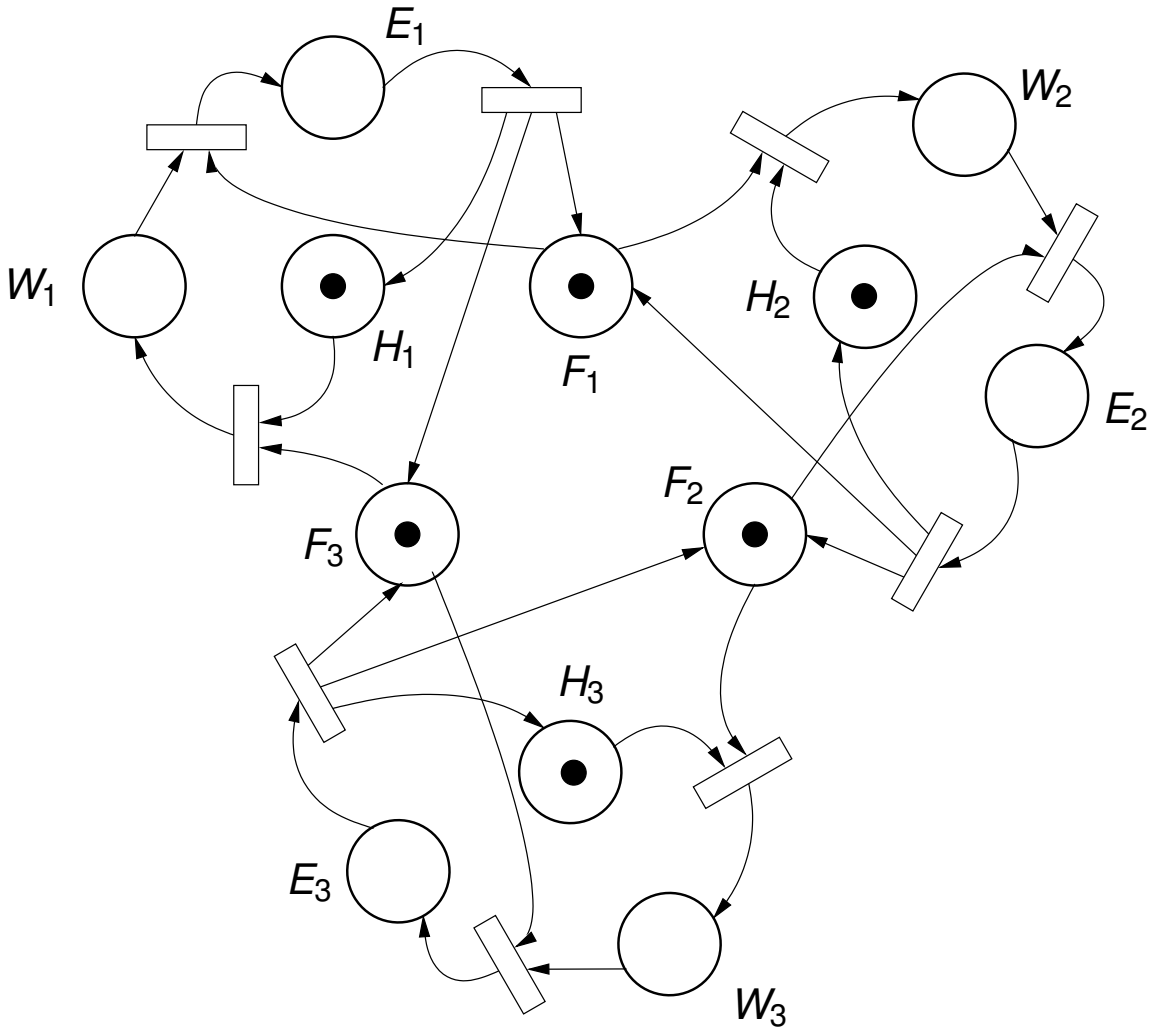


Beispiel: Dining Philosophers

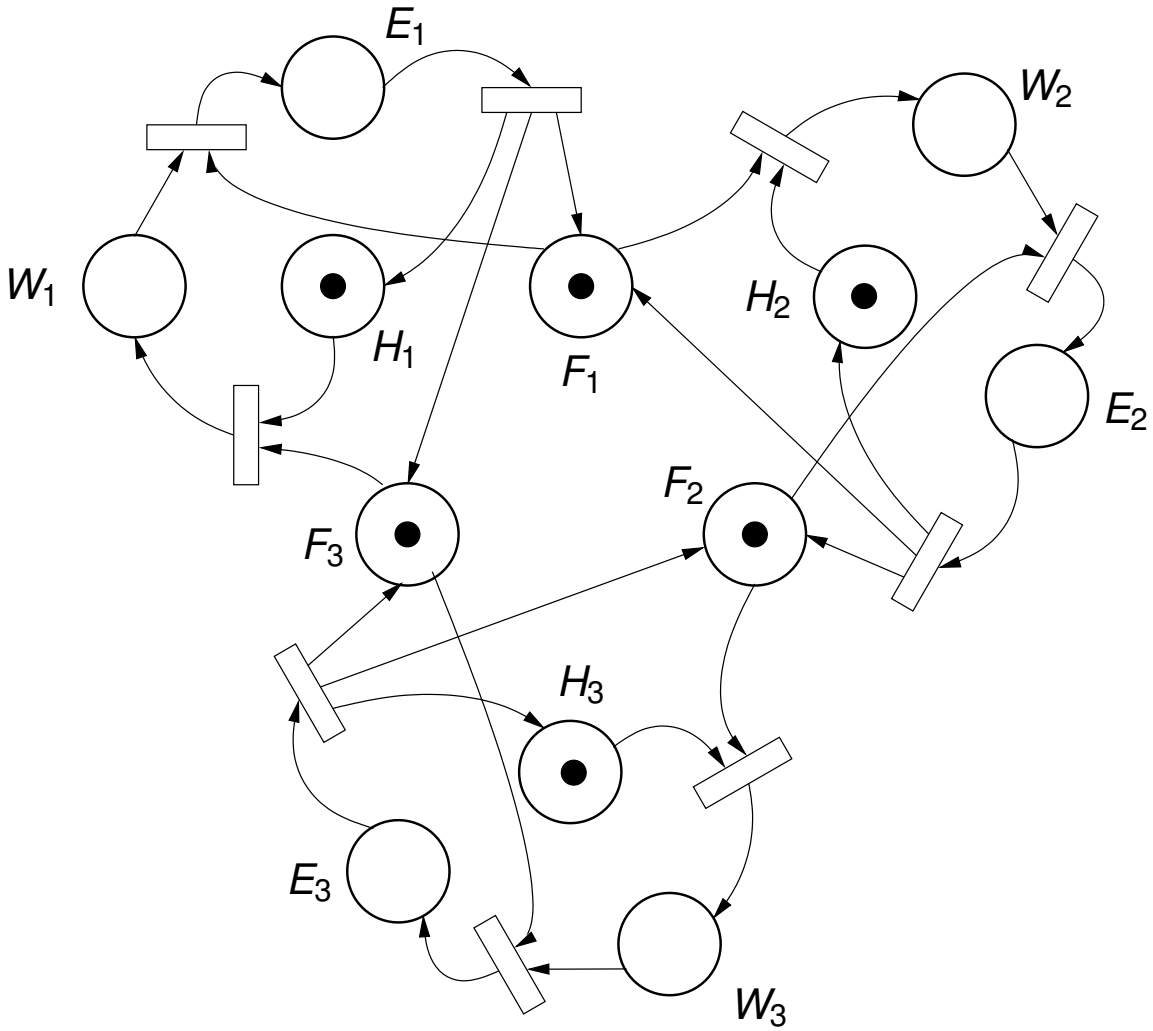


- P_1 legt Gabeln F_3 und F_1 zurück.

Beispiel: Dining Philosophers

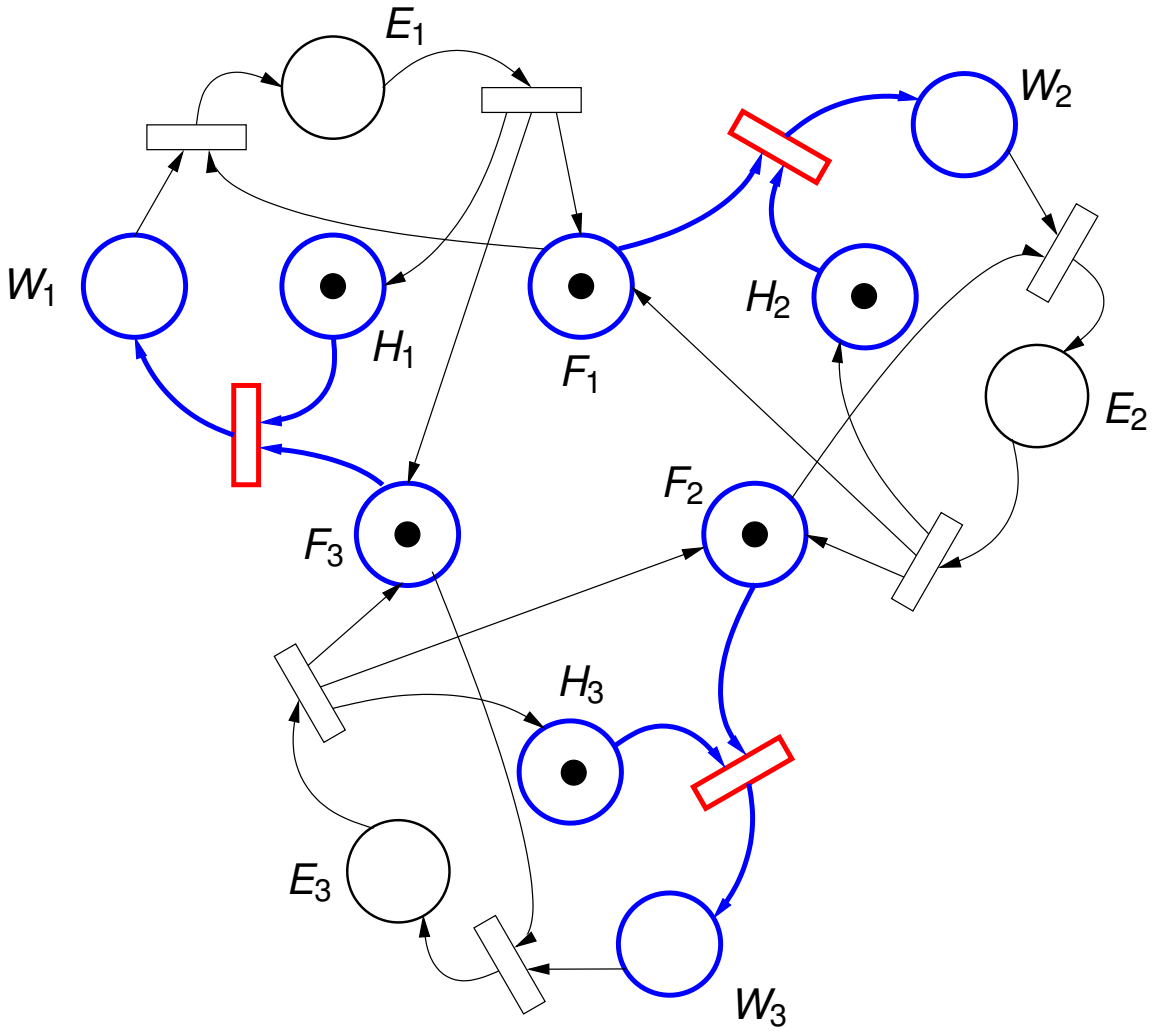


Beispiel: Dining Philosophers



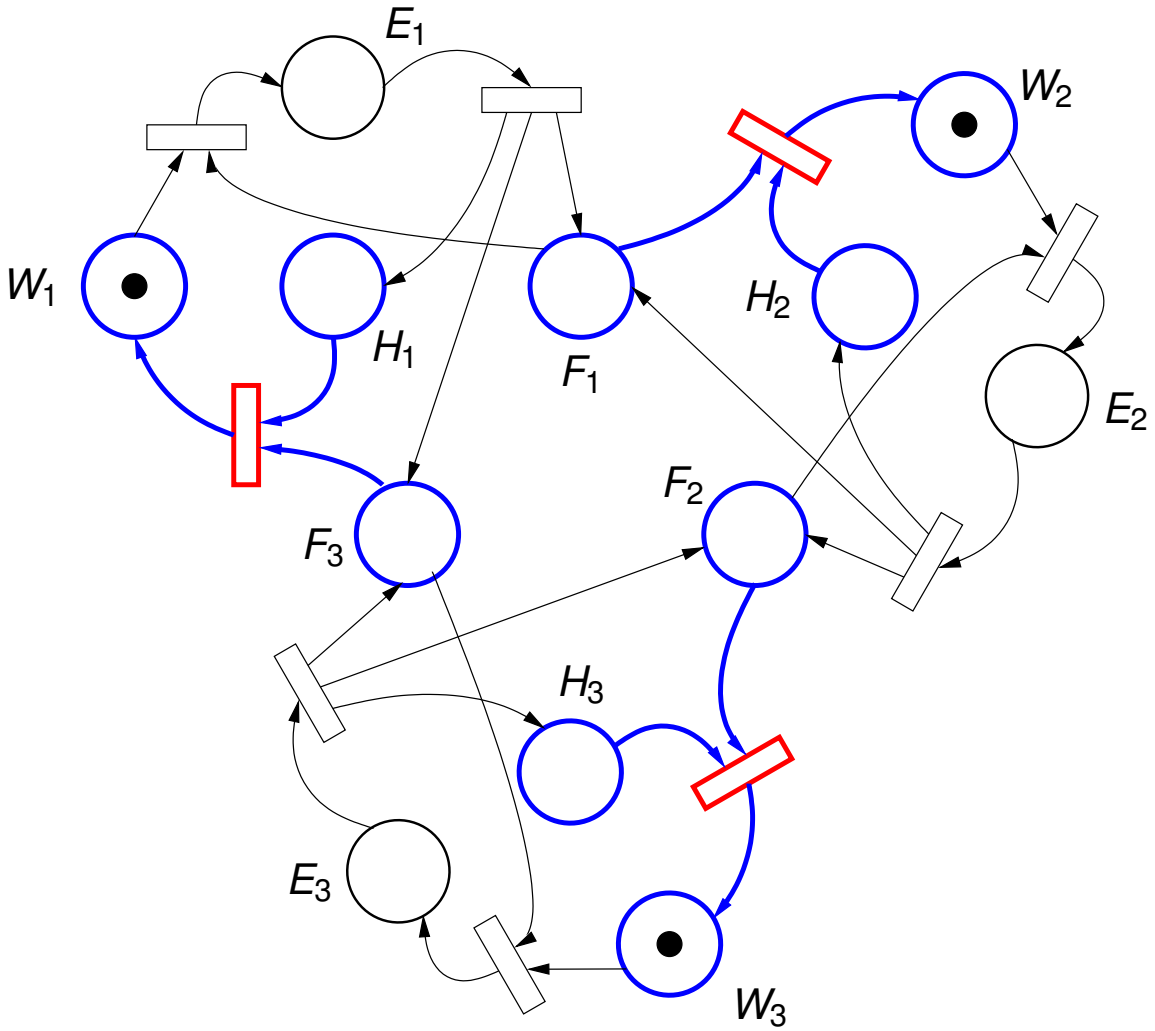
- Gibt es Möglichkeit der Verklemmung (Deadlock)?

Beispiel: Dining Philosophers



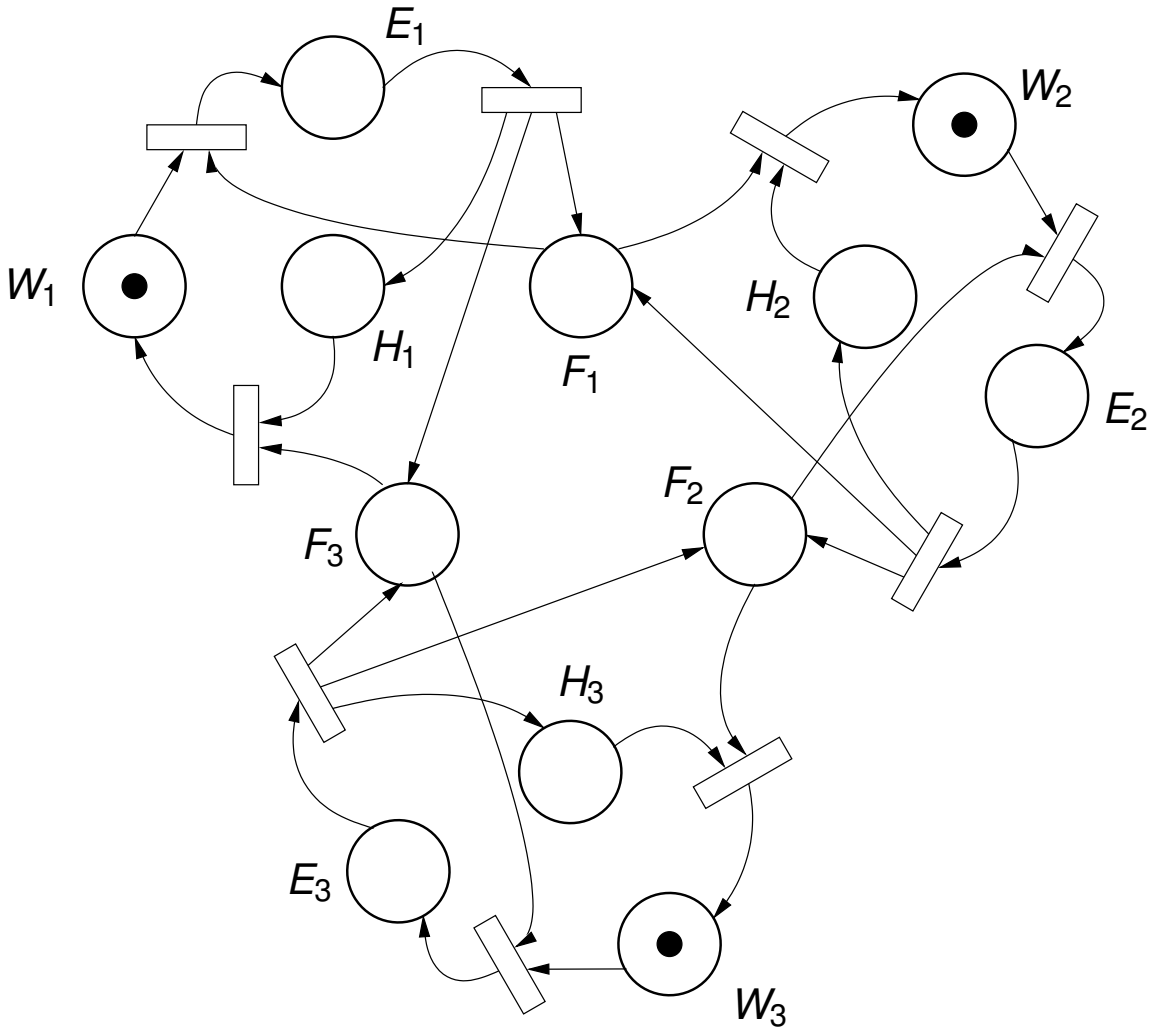
- Ja!

Beispiel: Dining Philosophers



- Ja!

Beispiel: Dining Philosophers



- Ja! Nachdem alle nebenläufig ihre rechte Gabel nehmen.

Petrinetze: Überdeckungsgraph

Erinnerung: Ein Petrinetz ist unbeschränkt genau dann, wenn sein Erreichbarkeitsgraph unendlich groß ist.

Petrinetze: Überdeckungsgraph

Erinnerung: Ein Petrinetz ist unbeschränkt genau dann, wenn sein Erreichbarkeitsgraph unendlich groß ist.

Aber wie können wir feststellen, ob Unendlichkeit vorliegt?

Petrinetze: Überdeckungsgraph

Erinnerung: Ein Petrinetz ist unbeschränkt genau dann, wenn sein Erreichbarkeitsgraph unendlich groß ist.

Aber wie können wir feststellen, ob Unendlichkeit vorliegt?

Und gibt es in diesem Fall trotzdem noch eine (endliche, grafische) Darstellung, die „in gewisser Weise“ alle erreichbaren Markierungen repräsentiert?

Petrinetze: Überdeckungsgraph

Erinnerung: Ein Petrinetz ist unbeschränkt genau dann, wenn sein Erreichbarkeitsgraph unendlich groß ist.

Aber wie können wir feststellen, ob Unendlichkeit vorliegt?

Und gibt es in diesem Fall trotzdem noch eine (endliche, grafische) Darstellung, die „in gewisser Weise“ alle erreichbaren Markierungen repräsentiert?

Und an Hand derer wir vielleicht sogar Eigenschaften wie Lebendigkeit und Kausalitäten entscheiden können?

Petrinetze: Überdeckungsgraph

Erinnerung: Ein Petrinetz ist unbeschränkt genau dann, wenn sein Erreichbarkeitsgraph unendlich groß ist.

Aber wie können wir feststellen, ob Unendlichkeit vorliegt?

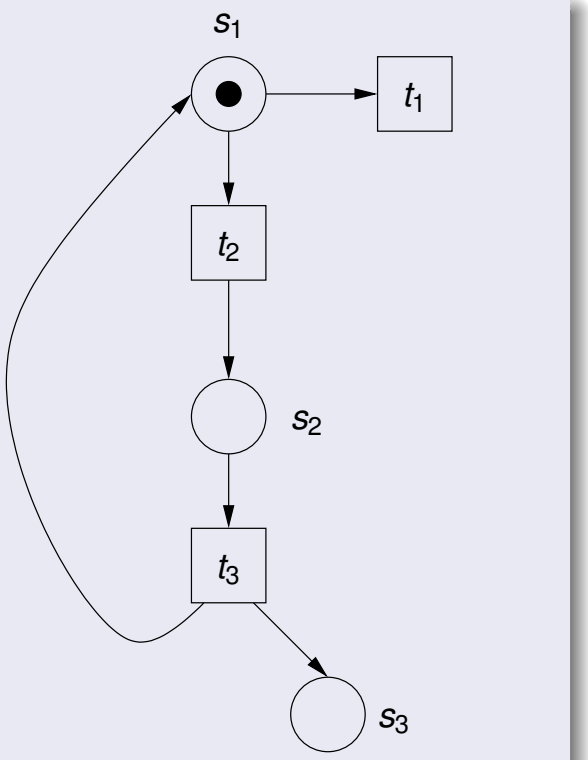
Und gibt es in diesem Fall trotzdem noch eine (endliche, grafische) Darstellung, die „in gewisser Weise“ alle erreichbaren Markierungen repräsentiert?

Und an Hand derer wir vielleicht sogar Eigenschaften wie Lebendigkeit und Kausalitäten entscheiden können?

$\rightsquigarrow$ Überdeckungsbaum $\rightsquigarrow$ Überdeckungsgraph

Petrinetze: Überdeckungsgraph

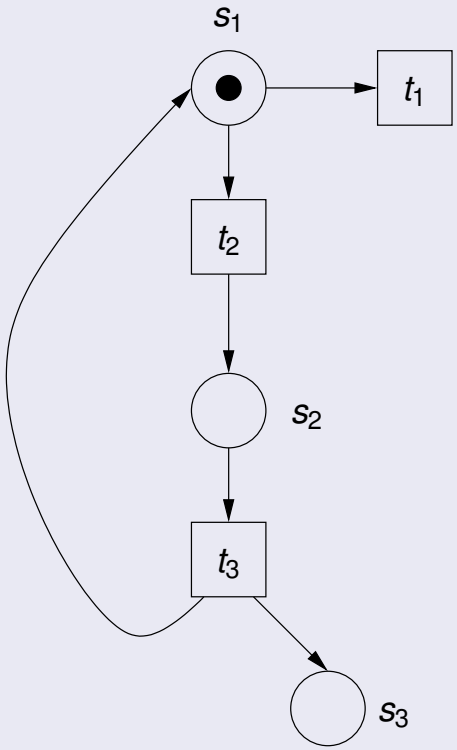
Beispiel eines unbeschränkten Petrinetzes:



Stellenordnung: s_1, s_2, s_3

Petrinetze: Überdeckungsgraph

Beispiel eines unbeschränkten Petrinetzes:



Stellenordnung: s_1, s_2, s_3

einige Schaltfolgen:

$\downarrow$

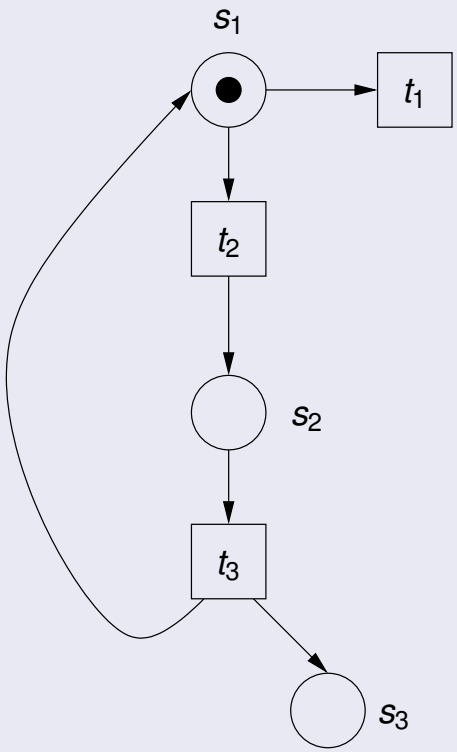
$(1,0,0)$

$\downarrow t_1$

$(0,0,0)$

Petrinetze: Überdeckungsgraph

Beispiel eines unbeschränkten Petrinetzes:



Stellenordnung: s_1, s_2, s_3

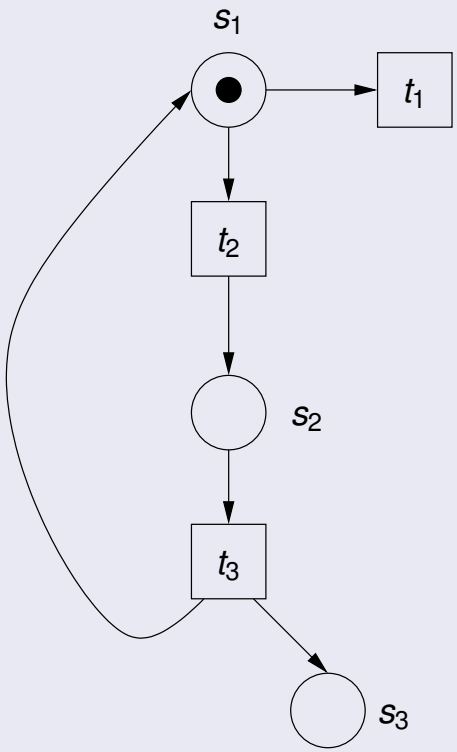
einige Schaltfolgen:

$$(1,0,0) \xrightarrow{t_1} (0,0,0)$$

$$(1,0,0) \xrightarrow{t_2} (0,1,0) \xrightarrow{t_3} (1,0,1) \xrightarrow{t_1} (0,0,1)$$

Petrinetze: Überdeckungsgraph

Beispiel eines unbeschränkten Petrinetzes:



Stellenordnung: s_1, s_2, s_3

einige Schaltfolgen:

$$(1,0,0) \xrightarrow{t_1} (0,0,0)$$

$$(1,0,0) \xrightarrow{t_2} (0,1,0) \xrightarrow{t_3} (1,0,1) \xrightarrow{t_1} (0,0,1)$$

$$(1,0,0) \xrightarrow{t_2} (0,1,0) \xrightarrow{t_3} (1,0,1) \xrightarrow{t_2} (0,1,1) \xrightarrow{t_3} (1,0,2) \xrightarrow{t_1} (0,0,2)$$

Petrinetze: Überdeckungsgraph

Beobachtungen:

- Das Verhalten von Petrinetzen ist „monoton“, das heißt, jede Schaltfolge ist auch dann noch möglich, wenn man der Ausgangsmarkierung (nicht unbedingt gleich der Anfangsmarkierung m_0) zusätzliche Marken hinzufügt.

Petrinetze: Überdeckungsgraph

Beobachtungen:

- Das Verhalten von Petrinetzen ist „monoton“, das heißt, jede Schaltfolge ist auch dann noch möglich, wenn man der Ausgangsmarkierung (nicht unbedingt gleich der Anfangsmarkierung m_0) zusätzliche Marken hinzufügt.
- Wenn zwei Markierungen m, m' existieren, so dass gilt:
  - m ist echt kleiner als m' (das heißt, $m \leq m'$ und $m \neq m'$, ab jetzt geschrieben als $m < m'$)

Petrinetze: Überdeckungsgraph

Beobachtungen:

- Das Verhalten von Petrinetzen ist „monoton“, das heißt, jede Schaltfolge ist auch dann noch möglich, wenn man der Ausgangsmarkierung (nicht unbedingt gleich der Anfangsmarkierung m_0) zusätzliche Marken hinzufügt.
- Wenn zwei Markierungen m, m' existieren, so dass gilt:
  - m ist echt kleiner als m' (das heißt, $m \leq m'$ und $m \neq m'$, ab jetzt geschrieben als $m < m'$) und
  - es gibt eine Schaltfolge von m zu m',

Petrinetze: Überdeckungsgraph

Beobachtungen:

- Das Verhalten von Petrinetzen ist „monoton“, das heißt, jede Schaltfolge ist auch dann noch möglich, wenn man der Ausgangsmarkierung (nicht unbedingt gleich der Anfangsmarkierung m_0) zusätzliche Marken hinzufügt.
- Wenn zwei Markierungen m, m' existieren, so dass gilt:
  - m ist echt kleiner als m' (das heißt, $m \leq m'$ und $m \neq m'$, ab jetzt geschrieben als $m < m'$) und
  - es gibt eine Schaltfolge von m zu m',

  dann kann man dieselbe Folge noch einmal von m' aus schalten und erhält eine wiederum echt größere Markierung m''.

Petrinetze: Überdeckungsgraph

Beobachtungen:

- Das Verhalten von Petrinetzen ist „monoton“, das heißt, jede Schaltfolge ist auch dann noch möglich, wenn man der Ausgangsmarkierung (nicht unbedingt gleich der Anfangsmarkierung m_0) zusätzliche Marken hinzufügt.
- Wenn zwei Markierungen m, m' existieren, so dass gilt:
  - m ist echt kleiner als m' (das heißt, $m \leq m'$ und $m \neq m'$, ab jetzt geschrieben als $m < m'$) und
  - es gibt eine Schaltfolge von m zu m',

  dann kann man dieselbe Folge noch einmal von m' aus schalten und erhält eine wiederum echt größere Markierung m''.

am Beispiel:

$$\rightarrow \underset{m}{(1,0,0)} \xrightarrow{t_2} \underset{<}{(0,1,0)} \xrightarrow{t_3} \underset{m'}{(1,0,1)} \xrightarrow{t_2} \underset{<}{(0,1,1)} \xrightarrow{t_3} \underset{m''}{(1,0,2)} \xrightarrow{t_1} (0,0,2)$$

Petrinetze: Überdeckungsgraph

Weitere Beobachtung:

- So einen Abschnitt kann man dann sogar immer weiter wiederholen, und die Markierung dadurch immer weiter wachsen lassen.

Petrinetze: Überdeckungsgraph

Weitere Beobachtung:

- So einen Abschnitt kann man dann sogar immer weiter wiederholen, und die Markierung dadurch immer weiter wachsen lassen.
- Man nennt dies auch „Pumpen“ der Folge.

Petrinetze: Überdeckungsgraph

Weitere Beobachtung:

- So einen Abschnitt kann man dann sogar immer weiter wiederholen, und die Markierung dadurch immer weiter wachsen lassen.
- Man nennt dies auch „Pumpen“ der Folge.

am Beispiel:

$$\longrightarrow (1,0,0) \xrightarrow{t_2} (0,1,0) \xrightarrow{t_3} (1,0,1) \xrightarrow{t_2} (0,1,1) \xrightarrow{t_3} (1,0,2)$$
$$\downarrow t_2$$
$$(0,1,2)$$
$$\downarrow t_3$$
$$(1,0,5) \xleftarrow{t_3} (0,1,4) \xleftarrow{t_2} (1,0,4) \xleftarrow{t_3} (0,1,3) \xleftarrow{t_2} (1,0,3)$$
$$\downarrow t_1$$
$$(0,0,5)$$

Petrinetze: Überdeckungsgraph

Dieses „immer weiter wachsen lassen“ kann man dadurch repräsentieren, dass man in den betroffenen Stellen eine spezielle ω-Markierung einführt, etwa:

$$\longrightarrow (1,0,0) \xrightarrow{t_2} (0,1,0) \xrightarrow{t_3} (1,0,\omega)$$

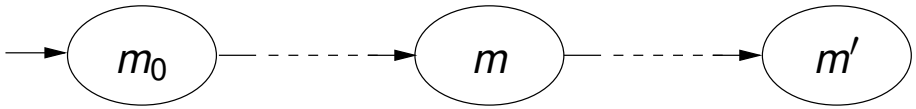
Petrinetze: Überdeckungsgraph

Dieses „immer weiter wachsen lassen“ kann man dadurch repräsentieren, dass man in den betroffenen Stellen eine spezielle ω-Markierung einführt, etwa:

$$\longrightarrow (1,0,0) \xrightarrow{t_2} (0,1,0) \xrightarrow{t_3} (1,0,\omega)$$

Formal:

Sobald eine neue Markierung m' hinzugefügt wird, führe für jede Vorgängermarkierung m mit $m < m'$ ...



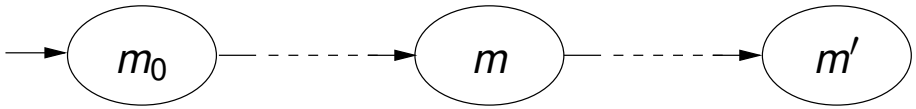
Petrinetze: Überdeckungsgraph

Dieses „immer weiter wachsen lassen“ kann man dadurch repräsentieren, dass man in den betroffenen Stellen eine spezielle ω-Markierung einführt, etwa:

$$\longrightarrow (1,0,0) \xrightarrow{t_2} (0,1,0) \xrightarrow{t_3} (1,0,\omega)$$

Formal:

Sobald eine neue Markierung m' hinzugefügt wird, führe für jede Vorgängermarkierung m mit $m < m'$ ...



... folgende Ersetzung auf m' durch:

- Für jedes $s \in S$ mit $m(s) < m'(s)$, setze $m'(s)$ auf ω.

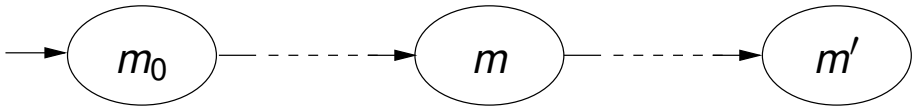
Petrinetze: Überdeckungsgraph

Dieses „immer weiter wachsen lassen“ kann man dadurch repräsentieren, dass man in den betroffenen Stellen eine spezielle ω-Markierung einführt, etwa:

$$\longrightarrow (1,0,0) \xrightarrow{t_2} (0,1,0) \xrightarrow{t_3} (1,0,\omega)$$

Formal:

Sobald eine neue Markierung m' hinzugefügt wird, führe für jede Vorgängermarkierung m mit $m < m'$ ...



... folgende Ersetzung auf m' durch:

- Für jedes $s \in S$ mit $m(s) < m'(s)$, setze $m'(s)$ auf ω.
- (Für alle anderen $s \in S$ gilt $m(s) = m'(s)$ und wir lassen $m'(s)$ unverändert.)

Petrinetze: Überdeckungsgraph

Bemerkungen:

- Die neu erzeugten besonderen ω-Markierungen ordnen einer oder mehreren Stellen sozusagen „unendlich“ viele Marken zu.

  Dies nimmt das wiederholte Schalten der Transitionsfolge von m zu m' vorweg, die nach und nach in den ω-Stellen beliebig viele Marken produzieren könnte.

Petrinetze: Überdeckungsgraph

Bemerkungen:

- Die neu erzeugten besonderen ω-Markierungen ordnen einer oder mehreren Stellen sozusagen „unendlich“ viele Marken zu.

  Dies nimmt das wiederholte Schalten der Transitionsfolge von m zu m' vorweg, die nach und nach in den ω-Stellen beliebig viele Marken produzieren könnte.

- Wir müssen auch für die ω-Markierungen, mit den konzeptionell beliebig vielen Marken auf bestimmten Stellen, ausdrücken können, wie jenseits der „gepumpten“ Teilfolge weiter geschaltet werden könnte, zum Beispiel:

$$\longrightarrow (1,0,0) \xrightarrow{t_2} (0,1,0) \xrightarrow{t_3} (1,0,\omega) \xrightarrow{t_1} \quad ?$$

Petrinetze: Überdeckungsgraph

Bemerkungen:

- Die neu erzeugten besonderen ω-Markierungen ordnen einer oder mehreren Stellen sozusagen „unendlich“ viele Marken zu.

  Dies nimmt das wiederholte Schalten der Transitionsfolge von m zu m' vorweg, die nach und nach in den ω-Stellen beliebig viele Marken produzieren könnte.

- Wir müssen auch für die ω-Markierungen, mit den konzeptionell beliebig vielen Marken auf bestimmten Stellen, ausdrücken können, wie jenseits der „gepumpten“ Teilfolge weiter geschaltet werden könnte, zum Beispiel:

$$\longrightarrow (1,0,0) \xrightarrow{t_2} (0,1,0) \xrightarrow{t_3} (1,0,\omega) \xrightarrow{t_1} \quad ?$$

- Dafür benötigen wir extra „Rechenregeln“ bezüglich ω.

Petrinetze: Überdeckungsgraph

Für alle $k \in \mathbb{N}_0$ legen wir fest: $\omega + k = \omega$ und $\omega - k = \omega$.

Petrinetze: Überdeckungsgraph

Für alle $k \in \mathbb{N}_0$ legen wir fest: $\omega + k = \omega$ und $\omega - k = \omega$.

Außerdem gilt uns ω als größer als jede natürliche Zahl, und $\omega \leq \omega$.

Petrinetze: Überdeckungsgraph

Für alle $k \in \mathbb{N}_0$ legen wir fest: $\omega + k = \omega$ und $\omega - k = \omega$.

Außerdem gilt uns ω als größer als jede natürliche Zahl, und $\omega \leq \omega$.

Damit haben wir jetzt zum Beispiel:

$$\longrightarrow (1,0,0) \xrightarrow{t_2} (0,1,0) \xrightarrow{t_3} (1,0,\omega) \xrightarrow{t_1} (0,0,\omega)$$

denn:

$$\begin{aligned} {}^\bullet t_1 &= (1,0,0) \\ t_1^\bullet &= (0,0,0) \\ (1,0,0) &\leq (1,0,\omega) \\ (1,0,\omega) \ominus (1,0,0) \oplus (0,0,0) &= (0,0,\omega) \end{aligned}$$

Petrinetze: Überdeckungsgraph

Außerdem aber auch:

$$\rightarrow (1,0,0) \xrightarrow{t_2} (0,1,0) \xrightarrow{t_3} (1,0,\omega) \xrightarrow{t_2} (0,1,\omega)$$

denn:

$$\begin{aligned} {}^\bullet t_2 &= (1,0,0) \\ t_2^\bullet &= (0,1,0) \\ (1,0,0) &\leq (1,0,\omega) \\ (1,0,\omega) \ominus (1,0,0) \oplus (0,1,0) &= (0,1,\omega) \end{aligned}$$

Petrinetze: Überdeckungsgraph

Außerdem aber auch:

$$\longrightarrow (1,0,0) \xrightarrow{t_2} (0,1,0) \xrightarrow{t_3} (1,0,\omega) \xrightarrow{t_2} (0,1,\omega) \xrightarrow{t_3} (1,0,\omega)$$

denn:

$$\begin{aligned} {}^\bullet t_2 &= (1,0,0) \\ t_2^\bullet &= (0,1,0) \\ (1,0,0) &\leq (1,0,\omega) \\ (1,0,\omega) \ominus (1,0,0) \oplus (0,1,0) &= (0,1,\omega) \end{aligned}$$

sowie:

$$\begin{aligned} {}^\bullet t_3 &= (0,1,0) \\ t_3^\bullet &= (1,0,1) \\ (0,1,0) &\leq (0,1,\omega) \\ (0,1,\omega) \ominus (0,1,0) \oplus (1,0,1) &= (1,0,\omega) \end{aligned}$$

Petrinetze: Überdeckungsgraph

Damit könnten wir jetzt wieder unendliche Pfade produzieren, etwa:

$$\longrightarrow (1,0,0) \xrightarrow{t_2} (0,1,0) \xrightarrow{t_3} (1,0,\omega) \xrightarrow{t_2} (0,1,\omega) \xrightarrow{t_3} (1,0,\omega)$$
$$\downarrow t_2$$
$$(0,1,\omega)$$
$$\downarrow t_3$$
$$(1,0,\omega) \xleftarrow{t_3} (0,1,\omega) \xleftarrow{t_2} (1,0,\omega) \xleftarrow{t_3} (0,1,\omega) \xleftarrow{t_2} (1,0,\omega)$$
$$\downarrow t_2$$
$$\vdots$$

Petrinetze: Überdeckungsgraph

Damit könnten wir jetzt wieder unendliche Pfade produzieren, etwa:

$$\longrightarrow (1,0,0) \xrightarrow{t_2} (0,1,0) \xrightarrow{t_3} (1,0,\omega) \xrightarrow{t_2} (0,1,\omega) \xrightarrow{t_3} (1,0,\omega)$$
$$\downarrow t_2$$
$$(0,1,\omega)$$
$$\downarrow t_3$$
$$(1,0,\omega) \xleftarrow{t_3} (0,1,\omega) \xleftarrow{t_2} (1,0,\omega) \xleftarrow{t_3} (0,1,\omega) \xleftarrow{t_2} (1,0,\omega)$$
$$\downarrow t_2$$
$$\vdots$$

Um das zu vermeiden, brechen wir ab, sobald sich eine Markierung exakt wiederholt.

Petrinetze: Überdeckungsgraph

Für das Beispiel haben wir jetzt insgesamt:

$$\longrightarrow (1,0,0) \xrightarrow{t_1} (0,0,0)$$

$$\longrightarrow (1,0,0) \xrightarrow{t_2} (0,1,0) \xrightarrow{t_3} (1,0,\omega) \xrightarrow{t_1} (0,0,\omega)$$

$$\longrightarrow (1,0,0) \xrightarrow{t_2} (0,1,0) \xrightarrow{t_3} (1,0,\omega) \xrightarrow{t_2} (0,1,\omega) \xrightarrow{t_3} (1,0,\omega)$$

Petrinetze: Überdeckungsgraph

Für das Beispiel haben wir jetzt insgesamt:

$$\longrightarrow (1,0,0) \xrightarrow{t_1} (0,0,0)$$

$$\longrightarrow (1,0,0) \xrightarrow{t_2} (0,1,0) \xrightarrow{t_3} (1,0,\omega) \xrightarrow{t_1} (0,0,\omega)$$

$$\longrightarrow (1,0,0) \xrightarrow{t_2} (0,1,0) \xrightarrow{t_3} (1,0,\omega) \xrightarrow{t_2} (0,1,\omega) \xrightarrow{t_3} (1,0,\omega)$$

(Man beachte, dass der zweite und dritte Pfad oben aus verschiedenen Gründen abbrechen.)

Petrinetze: Überdeckungsgraph

Für das Beispiel haben wir jetzt insgesamt:

$$\longrightarrow (1,0,0) \xrightarrow{t_1} (0,0,0)$$

$$\longrightarrow (1,0,0) \xrightarrow{t_2} (0,1,0) \xrightarrow{t_3} (1,0,\omega) \xrightarrow{t_1} (0,0,\omega)$$

$$\longrightarrow (1,0,0) \xrightarrow{t_2} (0,1,0) \xrightarrow{t_3} (1,0,\omega) \xrightarrow{t_2} (0,1,\omega) \xrightarrow{t_3} (1,0,\omega)$$

(Man beachte, dass der zweite und dritte Pfad oben aus verschiedenen Gründen abbrechen.)

Der Überdeckungsbaum stellt die Sammlung dieser Pfade kompakter, nämlich so weit wie möglich überlappend dar, indem gemeinsame Anfangsstücke nicht mehrfach repräsentiert werden.

Petrinetze: Überdeckungsgraph

Für das Beispiel haben wir jetzt insgesamt:

$$\longrightarrow (1,0,0) \xrightarrow{t_1} (0,0,0)$$

$$\longrightarrow (1,0,0) \xrightarrow{t_2} (0,1,0) \xrightarrow{t_3} (1,0,\omega) \xrightarrow{t_1} (0,0,\omega)$$

$$\longrightarrow (1,0,0) \xrightarrow{t_2} (0,1,0) \xrightarrow{t_3} (1,0,\omega) \xrightarrow{t_2} (0,1,\omega) \xrightarrow{t_3} (1,0,\omega)$$

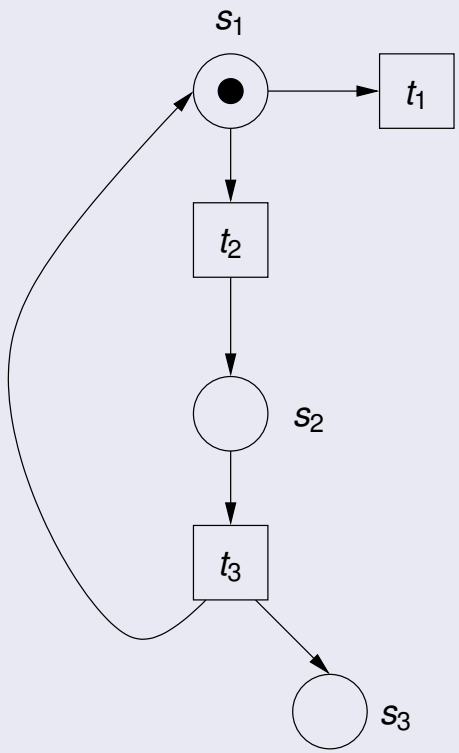
(Man beachte, dass der zweite und dritte Pfad oben aus verschiedenen Gründen abbrechen.)

Der Überdeckungsbaum stellt die Sammlung dieser Pfade kompakter, nämlich so weit wie möglich überlappend dar, indem gemeinsame Anfangsstücke nicht mehrfach repräsentiert werden.

Der englische Name für Überdeckungsbäume ist covering trees.

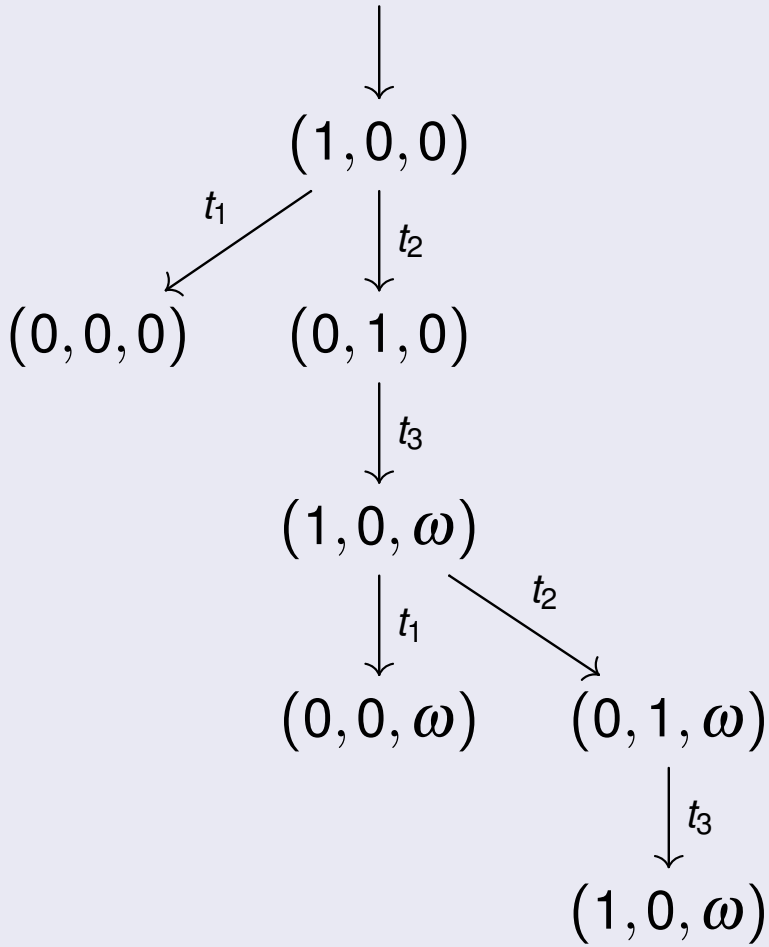
Petrinetze: Überdeckungsgraph

Beispiel:



Stellenordnung: s_1, s_2, s_3

Überdeckungsbaum:



Petrinetze: Überdeckungsgraph

Eigenschaften des Überdeckungsbaums (I)

- Die Konstruktion des Überdeckungsbaums terminiert immer nach endlich vielen Schritten.

Petrinetze: Überdeckungsgraph

Eigenschaften des Überdeckungsbaums (I)

- Die Konstruktion des Überdeckungsbaums terminiert immer nach endlich vielen Schritten.
- Irgendwelche ω-Markierungen treten genau dann auf, wenn das Petrinetz unbeschränkt ist.
  (Das heißt, der Überdeckungsbaum kann auch dazu verwendet werden, zu überprüfen, ob ein Petrinetz unbeschränkt ist.)

Petrinetze: Überdeckungsgraph

Eigenschaften des Überdeckungsbaums (I)

- Die Konstruktion des Überdeckungsbaums terminiert immer nach endlich vielen Schritten.
- Irgendwelche ω-Markierungen treten genau dann auf, wenn das Petrinetz unbeschränkt ist.
  (Das heißt, der Überdeckungsbaum kann auch dazu verwendet werden, zu überprüfen, ob ein Petrinetz unbeschränkt ist.)
- Auch etwa schwache Lebendigkeit sowie Kausalitäten lassen sich an Hand des Überdeckungsbaums entscheiden.

Petrinetze: Überdeckungsgraph

Eigenschaften des Überdeckungsbaums (II)

Sei N ein Petrinetz und B der dazugehörige Überdeckungsbaum. Dann gilt:

- Für jede erreichbare Markierung m von N gibt es einen Knoten m' in B mit $m \leq m'$.

Petrinetze: Überdeckungsgraph

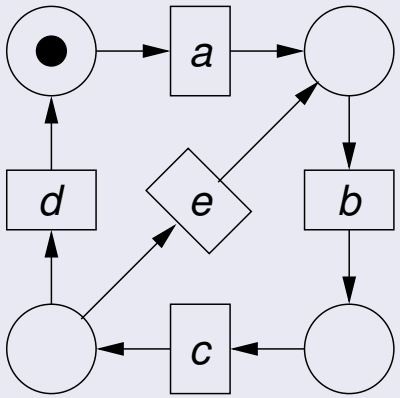
Eigenschaften des Überdeckungsbaums (II)

Sei N ein Petrinetz und B der dazugehörige Überdeckungsbaum. Dann gilt:

- Für jede erreichbare Markierung m von N gibt es einen Knoten m' in B mit $m \leq m'$.
- Für jeden Knoten m' in B und jedes $c \in \mathbb{N}_0$ gibt es eine erreichbare Markierung m von N, so dass für alle Stellen s gilt:
  - $m(s) = m'(s)$, falls $m'(s) \neq \omega$
  - $m(s) > c$, falls $m'(s) = \omega$.

Petrinetze: Überdeckungsgraph

Natürlich ist die Konstruktion eines Überdeckungsbaums auch für beschränkte Petrinetze möglich, etwa:



Stellenordnung wie in früherer Vorlesung

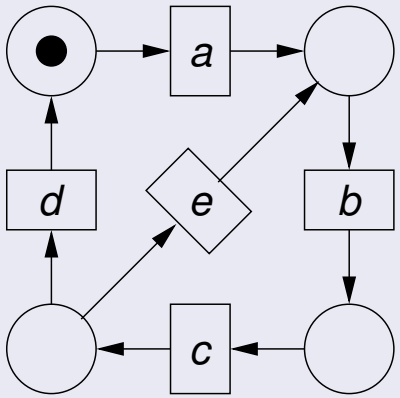
Überdeckungsbaum:

$\downarrow$

$(1, 0, 0, 0)$

Petrinetze: Überdeckungsgraph

Natürlich ist die Konstruktion eines Überdeckungsbaums auch für beschränkte Petrinetze möglich, etwa:



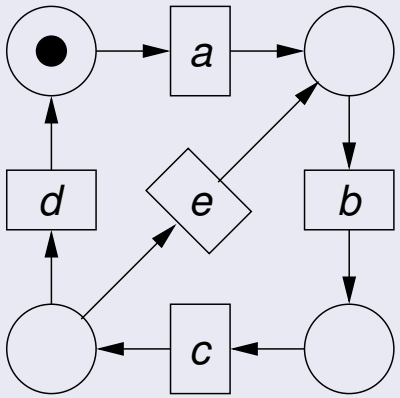
Stellenordnung wie in früherer Vorlesung

Überdeckungsbaum:

$$\downarrow$$
$$(1,0,0,0)$$
$$\downarrow a$$
$$(0,1,0,0)$$

Petrinetze: Überdeckungsgraph

Natürlich ist die Konstruktion eines Überdeckungsbaums auch für beschränkte Petrinetze möglich, etwa:



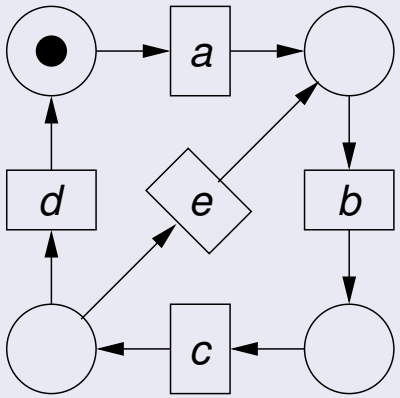
Stellenordnung wie in früherer Vorlesung

Überdeckungsbaum:

$$\downarrow$$
$$(1,0,0,0)$$
$$\downarrow a$$
$$(0,1,0,0)$$
$$\downarrow b$$
$$(0,0,0,1)$$

Petrinetze: Überdeckungsgraph

Natürlich ist die Konstruktion eines Überdeckungsbaums auch für beschränkte Petrinetze möglich, etwa:



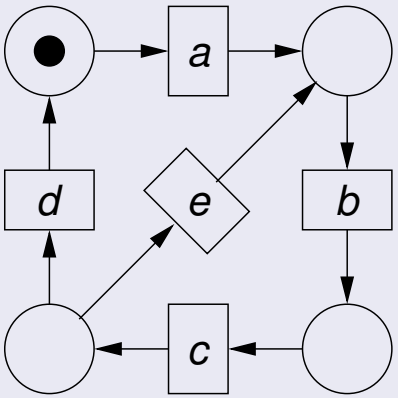
Stellenordnung wie in früherer Vorlesung

Überdeckungsbaum:

$$\downarrow$$
$$(1,0,0,0)$$
$$\downarrow a$$
$$(0,1,0,0)$$
$$\downarrow b$$
$$(0,0,0,1)$$
$$\downarrow c$$
$$(0,0,1,0)$$

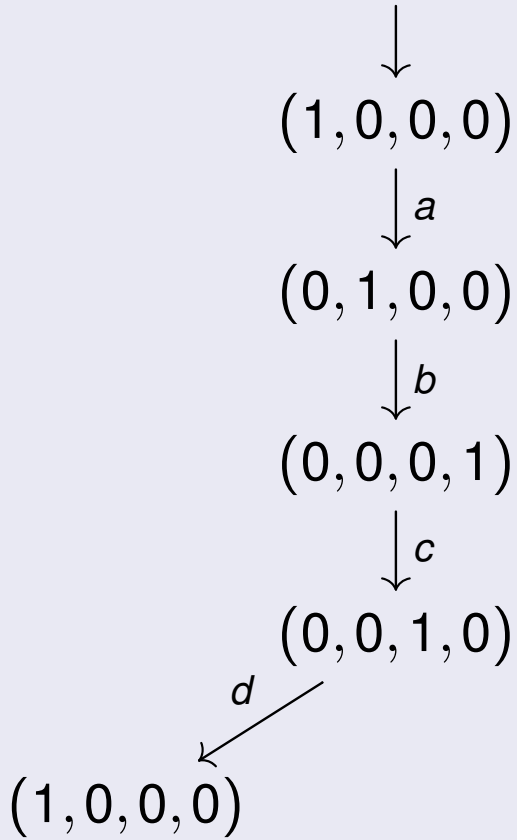
Petrinetze: Überdeckungsgraph

Natürlich ist die Konstruktion eines Überdeckungsbaums auch für beschränkte Petrinetze möglich, etwa:



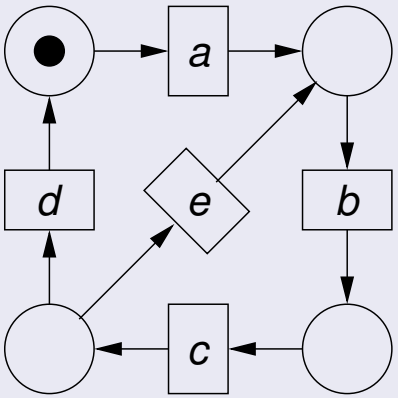
Stellenordnung wie in früherer Vorlesung

Überdeckungsbaum:



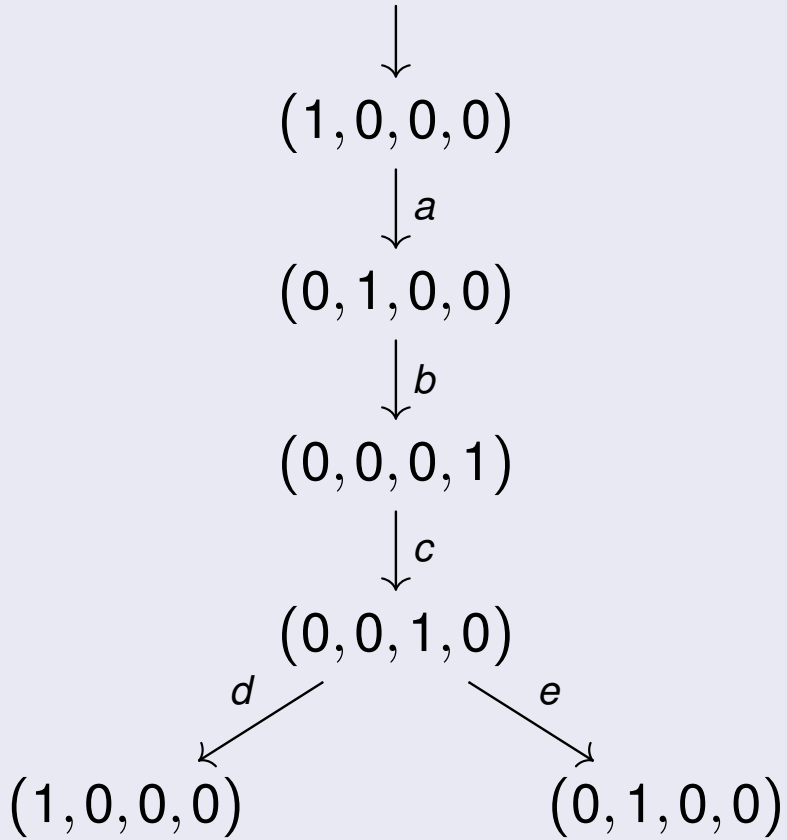
Petrinetze: Überdeckungsgraph

Natürlich ist die Konstruktion eines Überdeckungsbaums auch für beschränkte Petrinetze möglich, etwa:



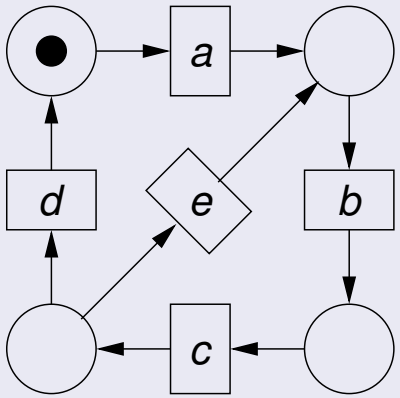
Stellenordnung wie in früherer Vorlesung

Überdeckungsbaum:



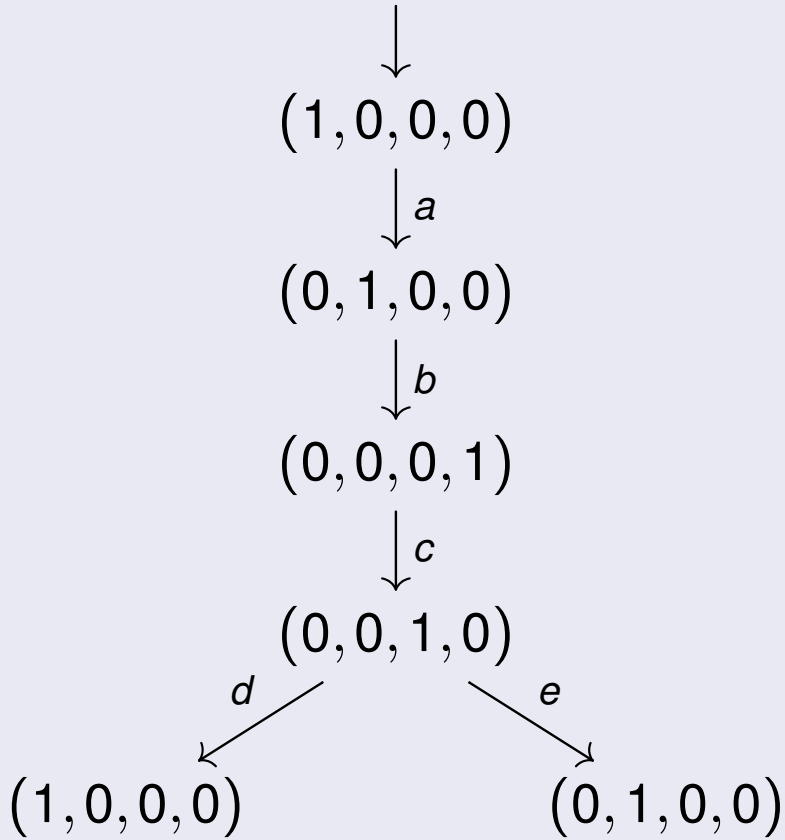
Petrinetze: Überdeckungsgraph

Natürlich ist die Konstruktion eines Überdeckungsbaums auch für beschränkte Petrinetze möglich, etwa:



Stellenordnung wie in früherer Vorlesung

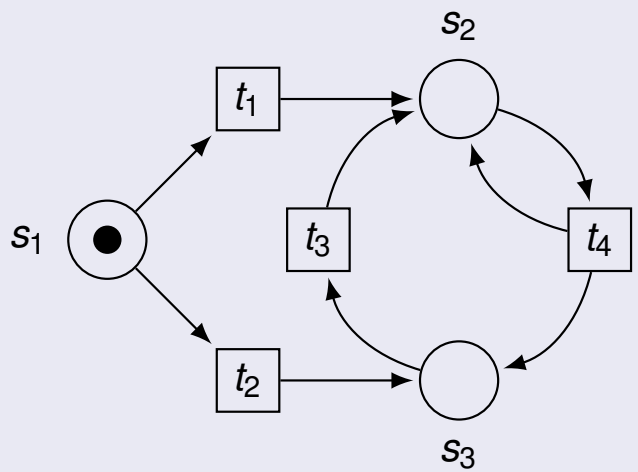
Überdeckungsbaum:



Allerdings ist in solchen Fällen der Erreichbarkeitsgraph klar attraktiver.

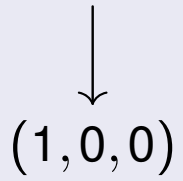
Petrinetze: Überdeckungsgraph

Und manchmal wird der Überdeckungsbaum schlicht unhandlich:



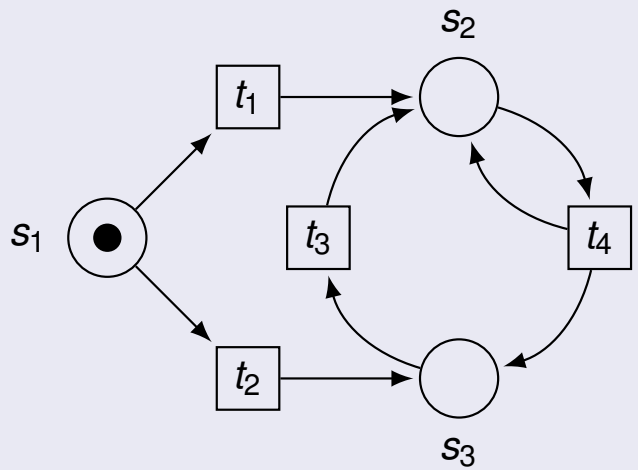
Stellenordnung: s_1, s_2, s_3

Überdeckungsbaum:



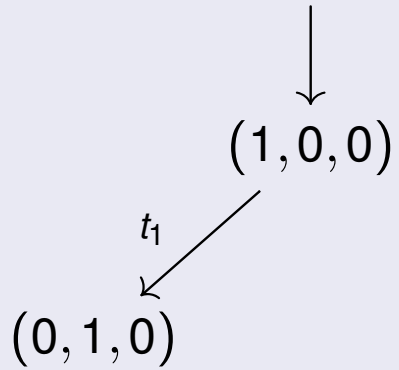
Petrinetze: Überdeckungsgraph

Und manchmal wird der Überdeckungsbaum schlicht unhandlich:



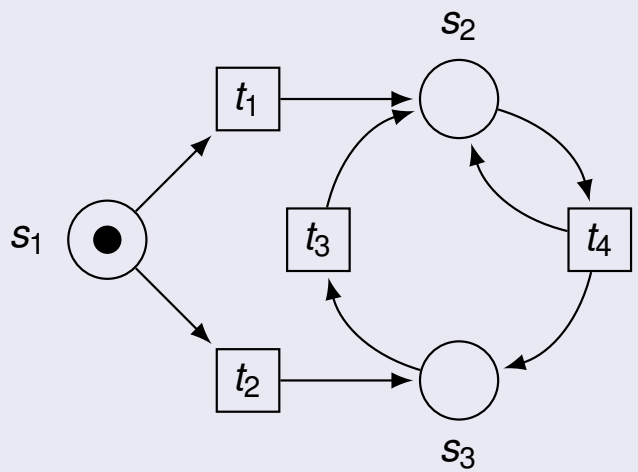
Stellenordnung: s_1, s_2, s_3

Überdeckungsbaum:



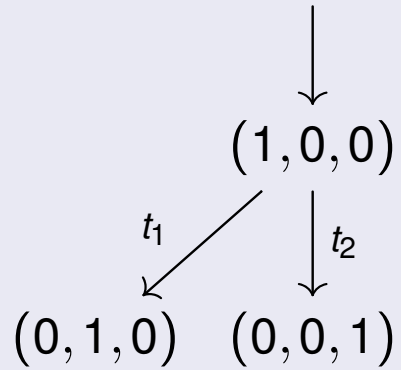
Petrinetze: Überdeckungsgraph

Und manchmal wird der Überdeckungsbaum schlicht unhandlich:



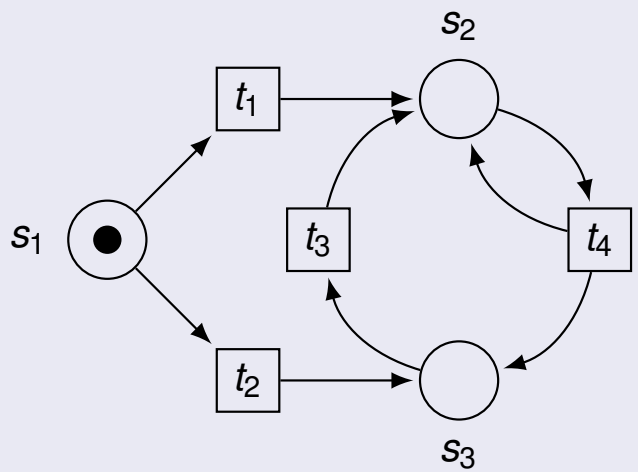
Stellenordnung: s_1, s_2, s_3

Überdeckungsbaum:



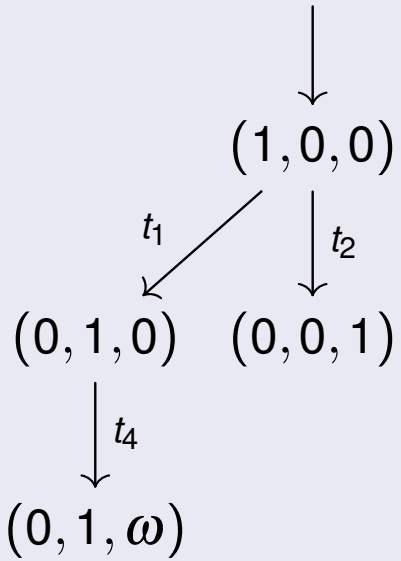
Petrinetze: Überdeckungsgraph

Und manchmal wird der Überdeckungsbaum schlicht unhandlich:



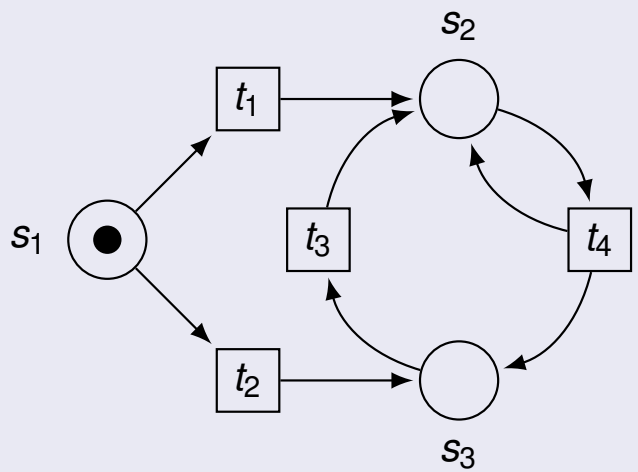
Stellenordnung: s_1, s_2, s_3

Überdeckungsbaum:



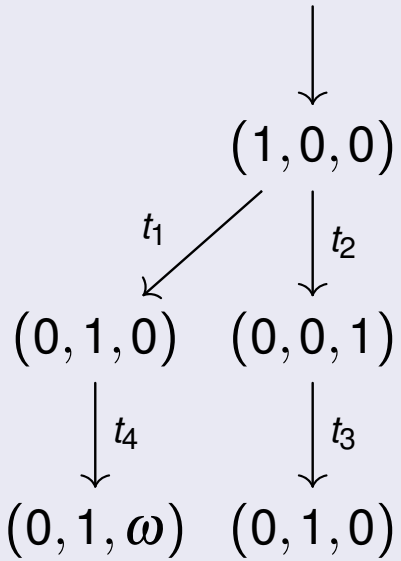
Petrinetze: Überdeckungsgraph

Und manchmal wird der Überdeckungsbaum schlicht unhandlich:



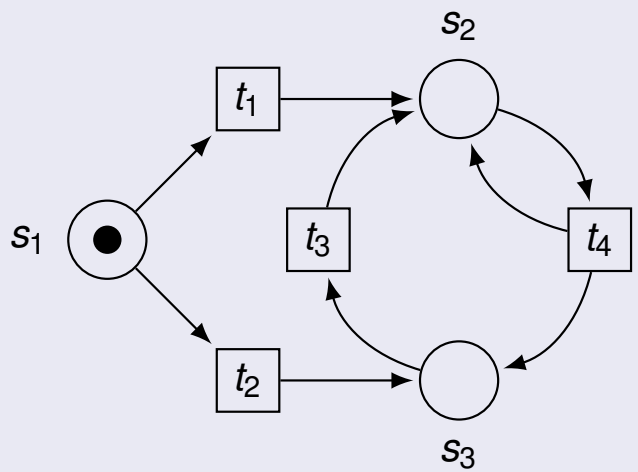
Stellenordnung: s_1, s_2, s_3

Überdeckungsbaum:



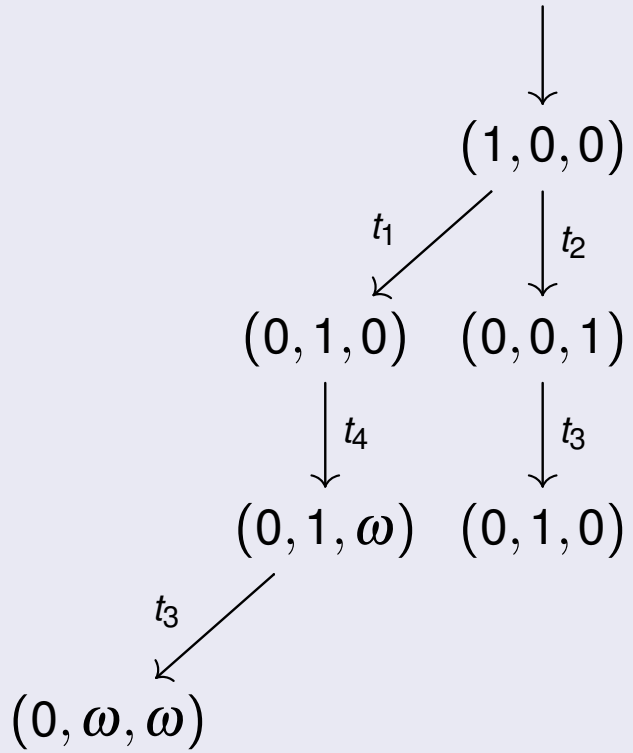
Petrinetze: Überdeckungsgraph

Und manchmal wird der Überdeckungsbaum schlicht unhandlich:



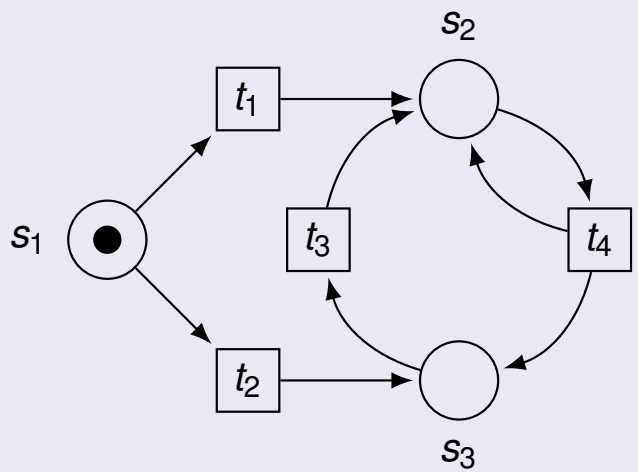
Stellenordnung: s_1, s_2, s_3

Überdeckungsbaum:



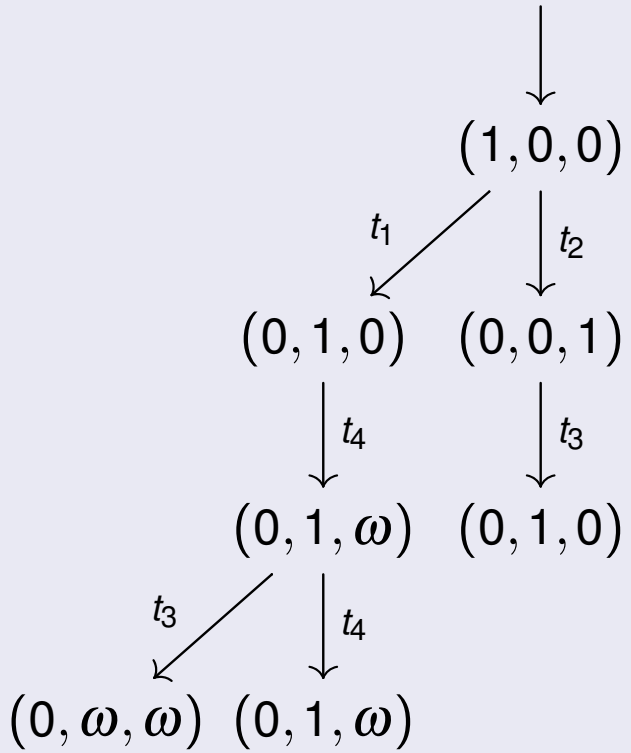
Petrinetze: Überdeckungsgraph

Und manchmal wird der Überdeckungsbaum schlicht unhandlich:



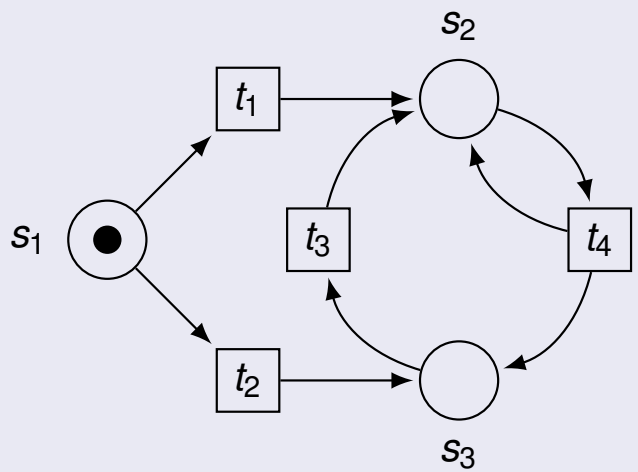
Stellenordnung: s_1, s_2, s_3

Überdeckungsbaum:



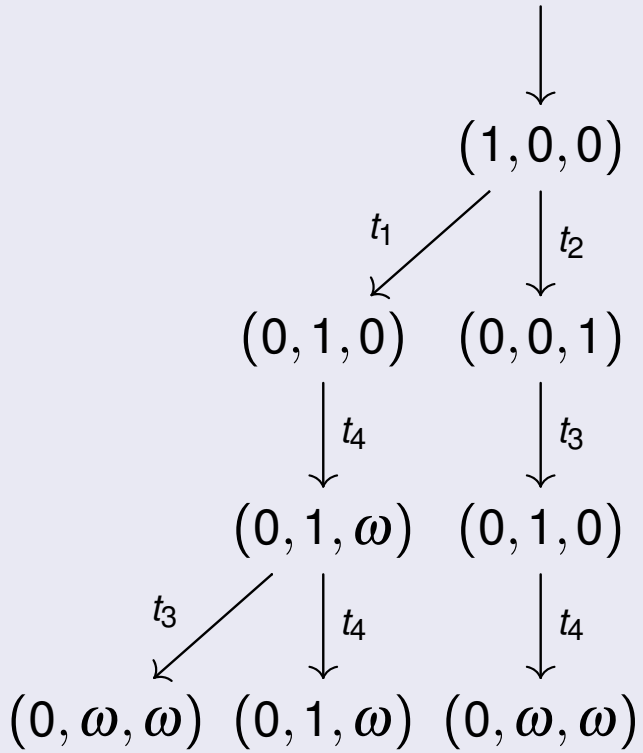
Petrinetze: Überdeckungsgraph

Und manchmal wird der Überdeckungsbaum schlicht unhandlich:



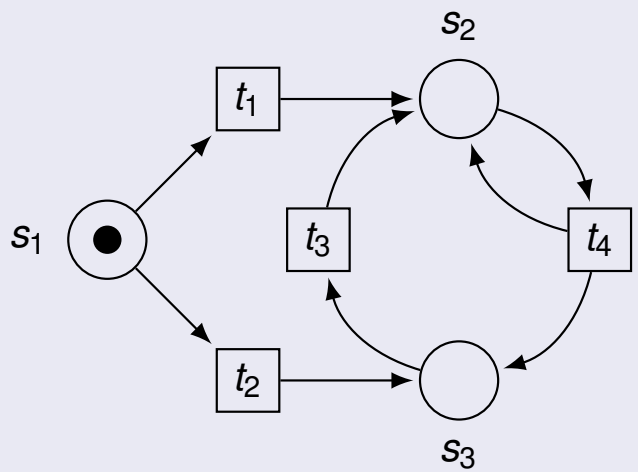
Stellenordnung: s_1, s_2, s_3

Überdeckungsbaum:



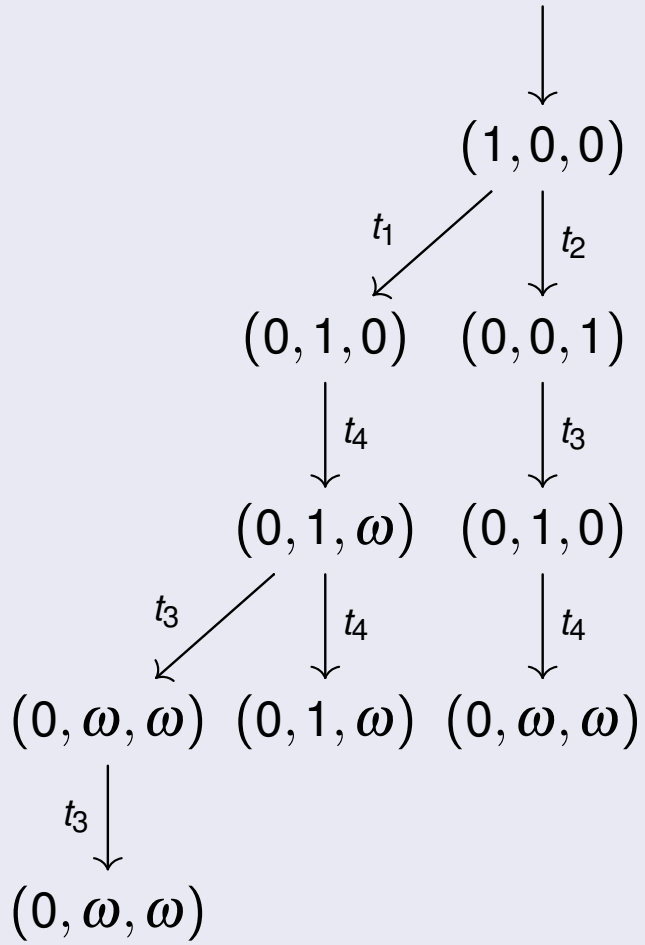
Petrinetze: Überdeckungsgraph

Und manchmal wird der Überdeckungsbaum schlicht unhandlich:



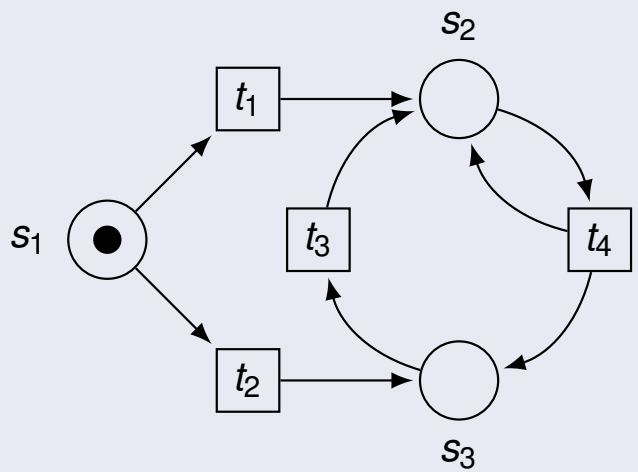
Stellenordnung: s_1, s_2, s_3

Überdeckungsbaum:



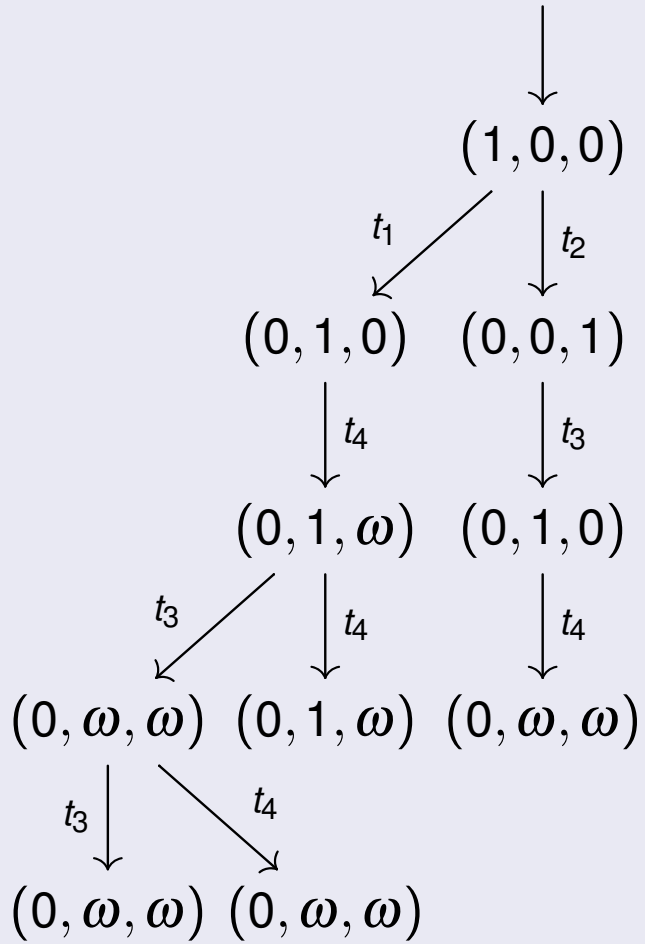
Petrinetze: Überdeckungsgraph

Und manchmal wird der Überdeckungsbaum schlicht unhandlich:



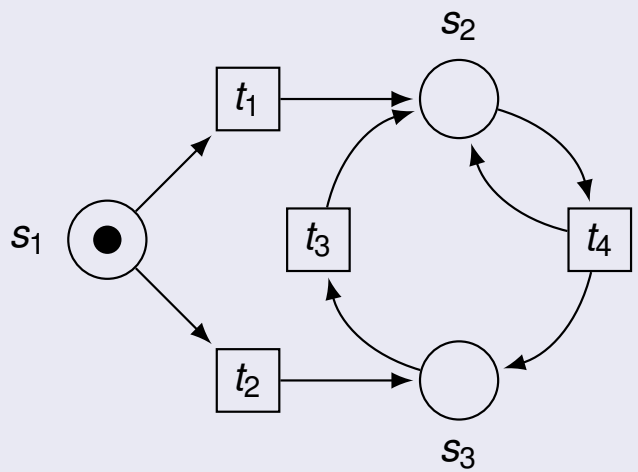
Stellenordnung: s_1, s_2, s_3

Überdeckungsbaum:



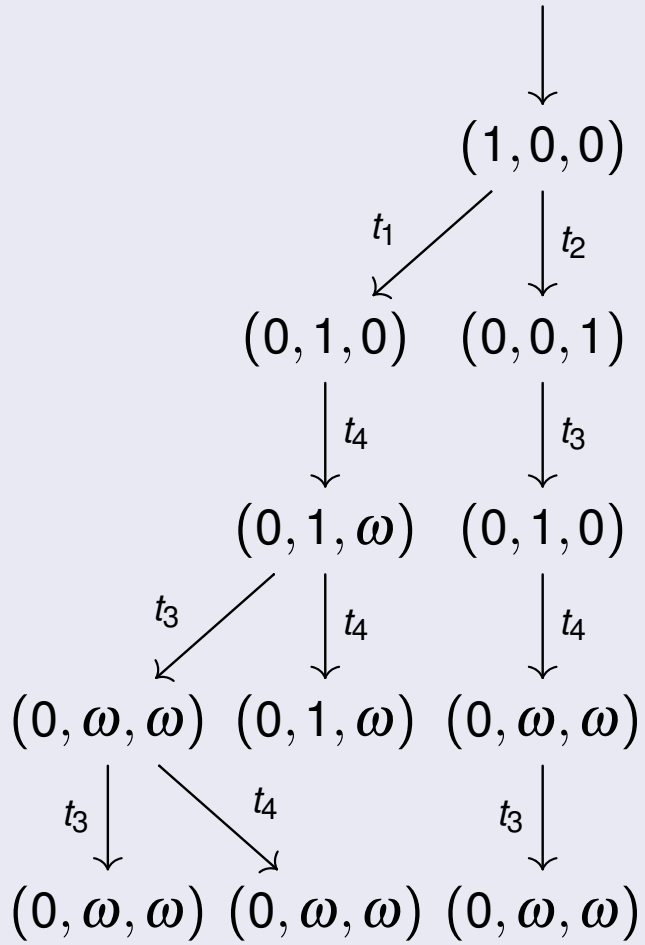
Petrinetze: Überdeckungsgraph

Und manchmal wird der Überdeckungsbaum schlicht unhandlich:



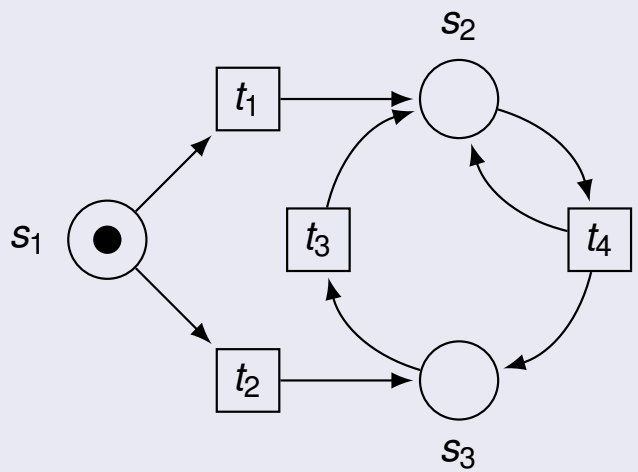
Stellenordnung: s_1, s_2, s_3

Überdeckungsbaum:



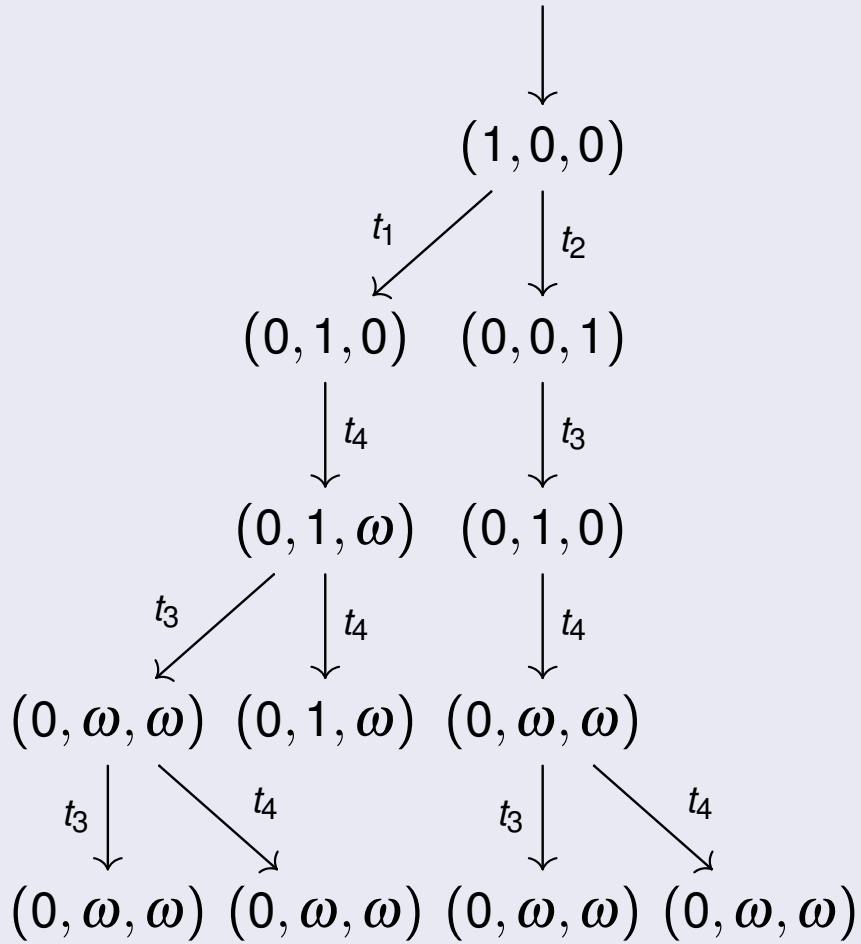
Petrinetze: Überdeckungsgraph

Und manchmal wird der Überdeckungsbaum schlicht unhandlich:



Stellenordnung: s_1, s_2, s_3

Überdeckungsbaum:



Petrinetze: Überdeckungsgraph

Gibt es ein Konstrukt, das die Vorteile von Überdeckungsbaum und Erreichbarkeitsgraph vereint?

Petrinetze: Überdeckungsgraph

Gibt es ein Konstrukt, das die Vorteile von Überdeckungsbaum und Erreichbarkeitsgraph vereint?

Ja, einen Überdeckungsgraph (englisch: covering graph).

Petrinetze: Überdeckungsgraph

Gibt es ein Konstrukt, das die Vorteile von Überdeckungsbaum und Erreichbarkeitsgraph vereint?

Ja, einen Überdeckungsgraph (englisch: covering graph).

Grundsätzliche Idee:

- Konstruktion wie Überdeckungsbaum
- Aber niemals Erzeugung zweier Knoten mit gleicher Markierung
- Stattdessen Pfeile zu früher erzeugten Knoten

Petrinetze: Überdeckungsgraph

Gibt es ein Konstrukt, das die Vorteile von Überdeckungsbaum und Erreichbarkeitsgraph vereint?

Ja, einen Überdeckungsgraph (englisch: covering graph).

Grundsätzliche Idee:

- Konstruktion wie Überdeckungsbaum
- Aber niemals Erzeugung zweier Knoten mit gleicher Markierung
- Stattdessen Pfeile zu früher erzeugten Knoten

Feststellung: Für beschränkte Petrinetze sind Erreichbarkeitsgraph und Überdeckungsgraph das Gleiche.

Petrinetze: Überdeckungsgraph

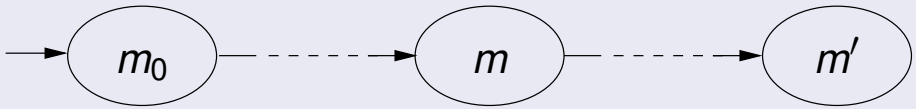
Konstruktion eines Überdeckungsgraphen – alternativ beschrieben

- Führe zunächst wie gewohnt die Konstruktion des Erreichbarkeitsgraphen aus.

Petrinetze: Überdeckungsgraph

Konstruktion eines Überdeckungsgraphen – alternativ beschrieben

- Führe zunächst wie gewohnt die Konstruktion des Erreichbarkeitsgraphen aus.
- Sobald eine Markierung m' erzeugt wurde, führe für jede Markierung $m < m'$ irgendwo zwischen m_0 und m' ...



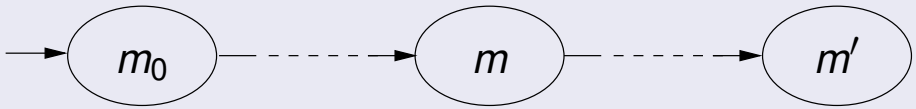
... folgende Ersetzung auf m' durch:

- Für jedes $s \in S$ mit $m(s) < m'(s)$, setze $m'(s)$ auf ω.

Petrinetze: Überdeckungsgraph

Konstruktion eines Überdeckungsgraphen – alternativ beschrieben

- Führe zunächst wie gewohnt die Konstruktion des Erreichbarkeitsgraphen aus.
- Sobald eine Markierung m' erzeugt wurde, führe für jede Markierung $m < m'$ irgendwo zwischen m_0 und m' ...



... folgende Ersetzung auf m' durch:

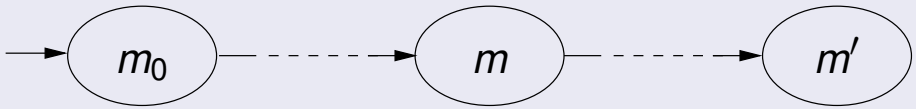
- Für jedes $s \in S$ mit $m(s) < m'(s)$, setze $m'(s)$ auf ω.

Sollte sich danach herausstellen, dass das neue m' (welches auch unverändert das ursprüngliche sein könnte) schon im Graph vorkommt, verwende diesen vorhandenen Knoten.

Petrinetze: Überdeckungsgraph

Konstruktion eines Überdeckungsgraphen – alternativ beschrieben

- Führe zunächst wie gewohnt die Konstruktion des Erreichbarkeitsgraphen aus.
- Sobald eine Markierung m' erzeugt wurde, führe für jede Markierung $m < m'$ irgendwo zwischen m_0 und m' ...



... folgende Ersetzung auf m' durch:

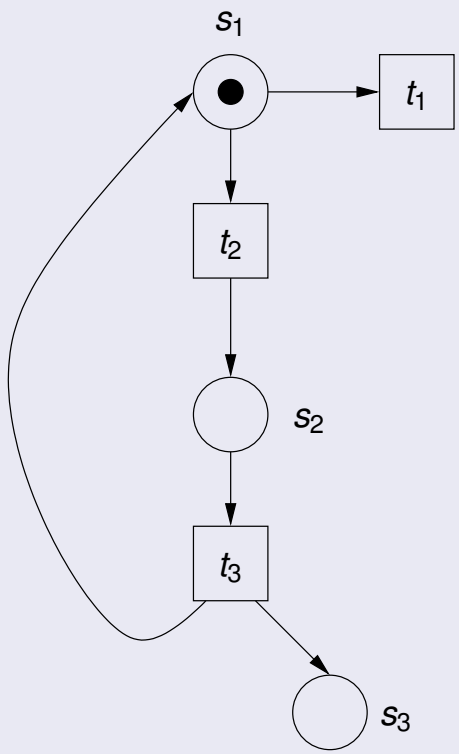
  - Für jedes $s \in S$ mit $m(s) < m'(s)$, setze $m'(s)$ auf ω.

Sollte sich danach herausstellen, dass das neue m' (welches auch unverändert das ursprüngliche sein könnte) schon im Graph vorkommt, verwende diesen vorhandenen Knoten.

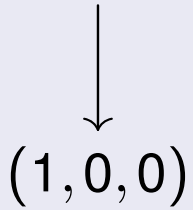
- Mache mit der Konstruktion weiter, bis keine Markierungen mehr hinzugefügt werden können.

Petrinetze: Überdeckungsgraph

Beispiel:



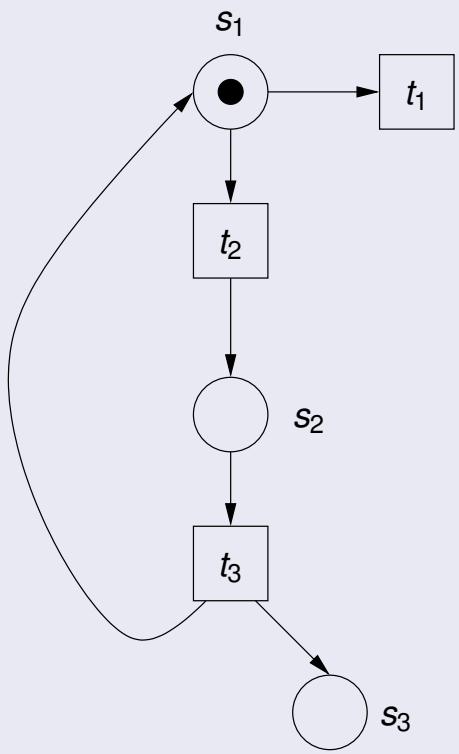
Stellenordnung: s_1, s_2, s_3



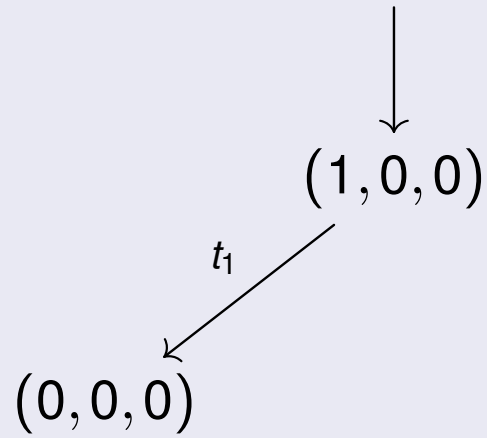
$(1, 0, 0)$

Petrinetze: Überdeckungsgraph

Beispiel:

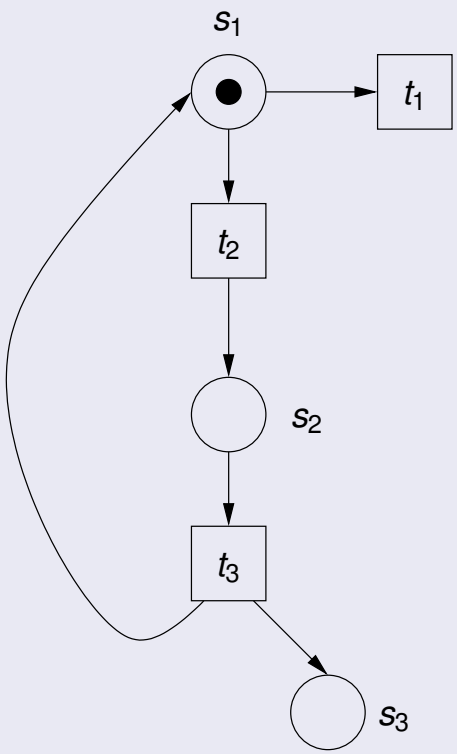


Stellenordnung: s_1, s_2, s_3

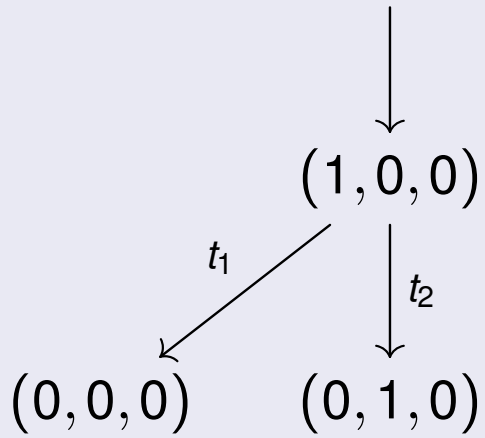


Petrinetze: Überdeckungsgraph

Beispiel:

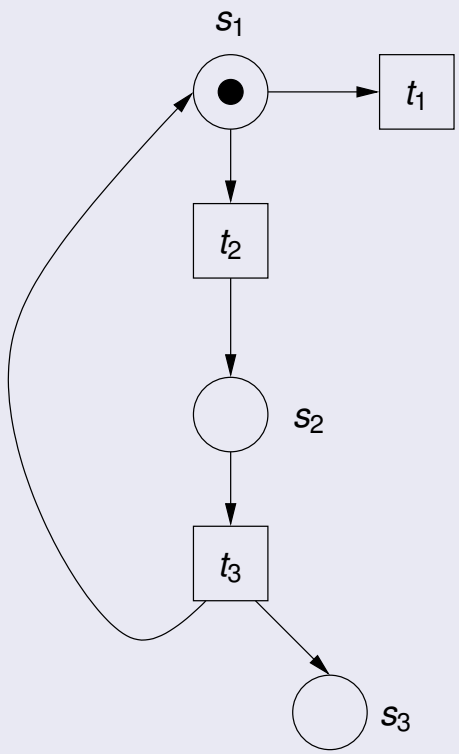


Stellenordnung: s_1, s_2, s_3

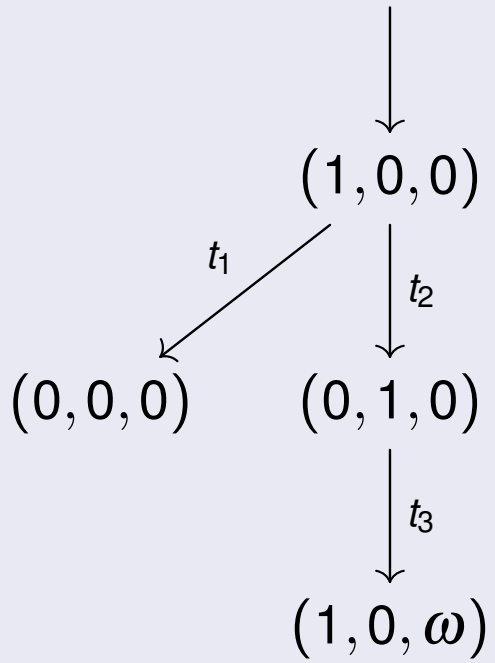


Petrinetze: Überdeckungsgraph

Beispiel:

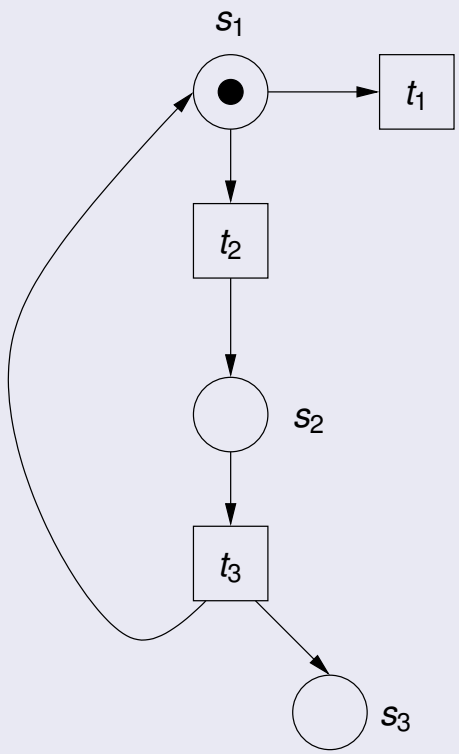


Stellenordnung: s_1, s_2, s_3

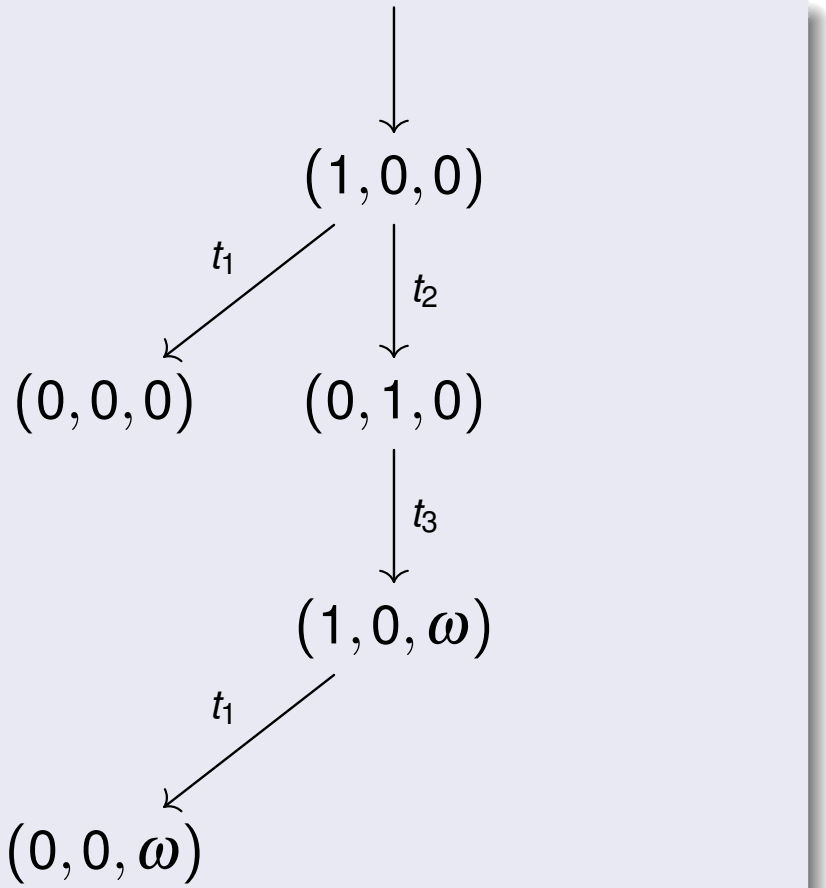


Petrinetze: Überdeckungsgraph

Beispiel:

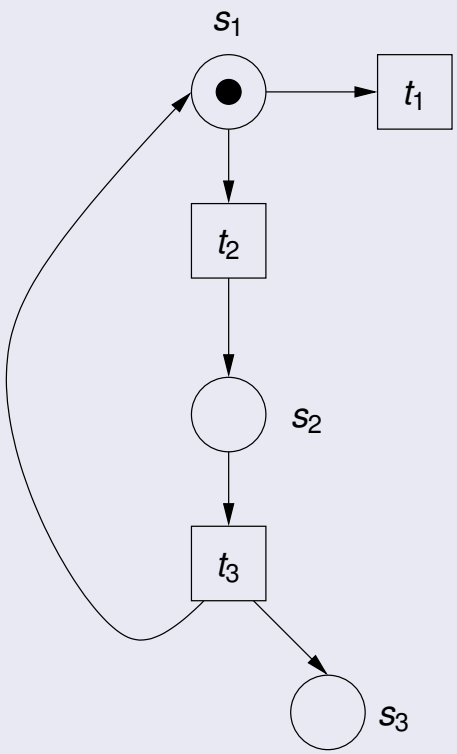


Stellenordnung: s_1, s_2, s_3

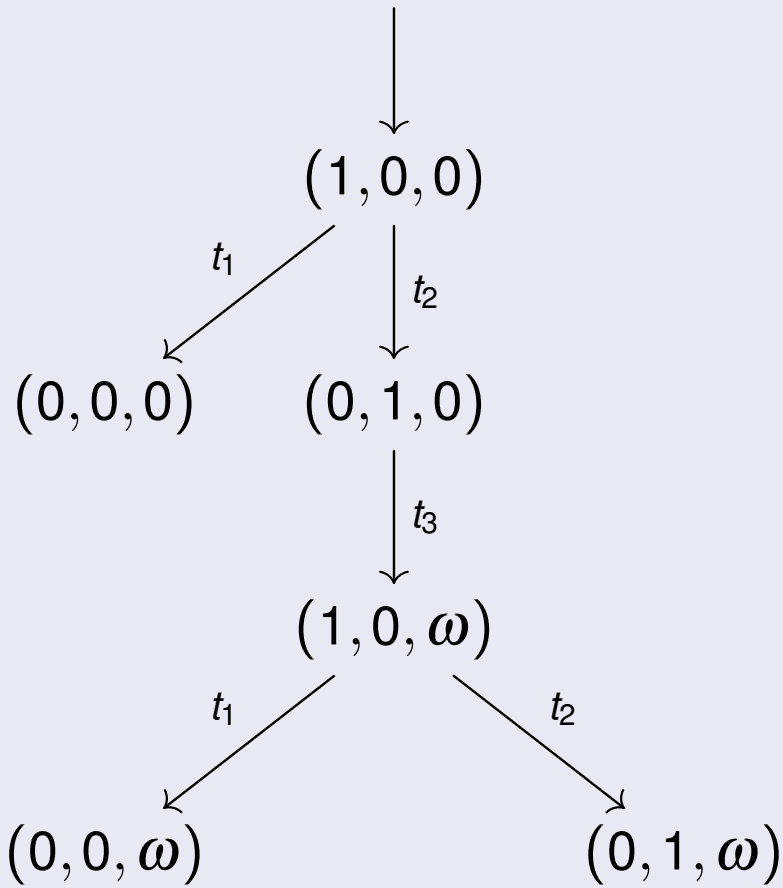


Petrinetze: Überdeckungsgraph

Beispiel:

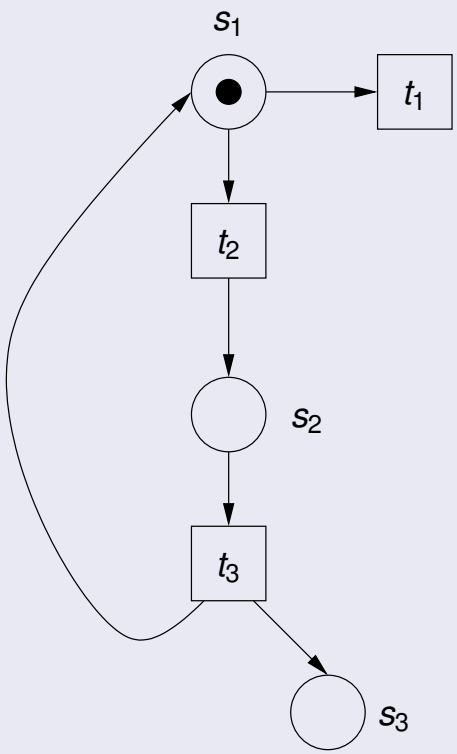


Stellenordnung: s_1, s_2, s_3

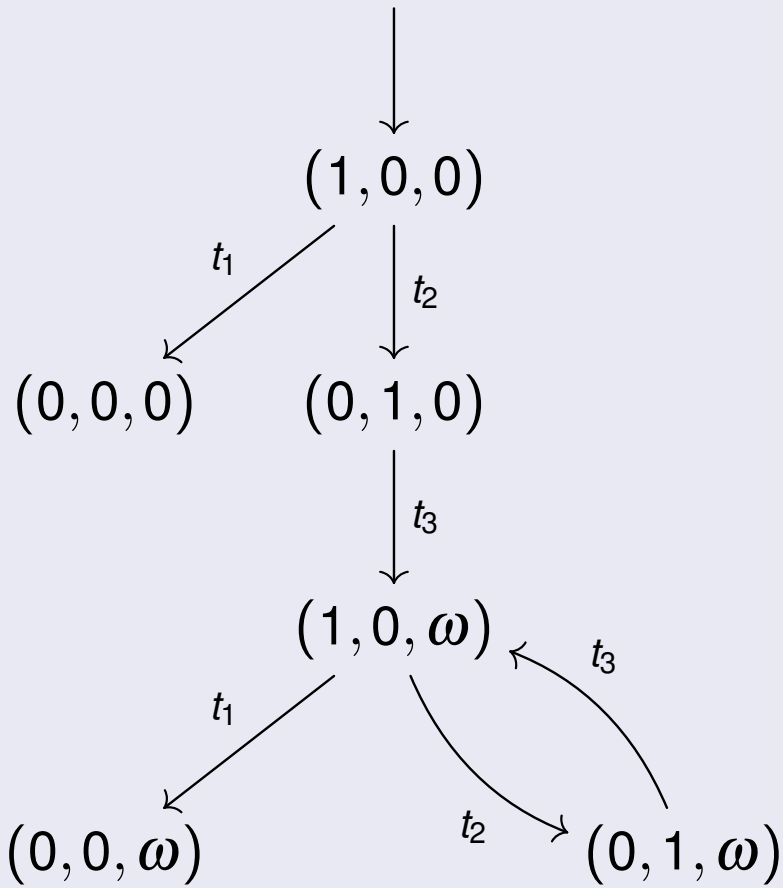


Petrinetze: Überdeckungsgraph

Beispiel:



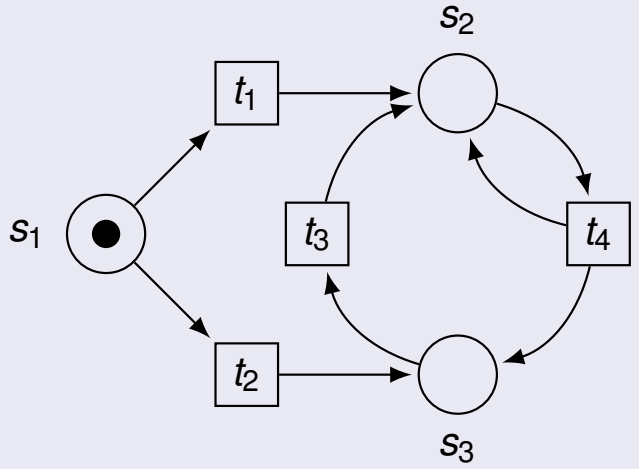
Stellenordnung: s_1, s_2, s_3



Petrinetze: Überdeckungsgraph

Ein Petrinetz kann mehrere Überdeckungsgraphen haben.

Hier für ein Beispiel, für das wir auch schon den Überdeckungsbaum gesehen haben:



Stellenordnung: s_1, s_2, s_3

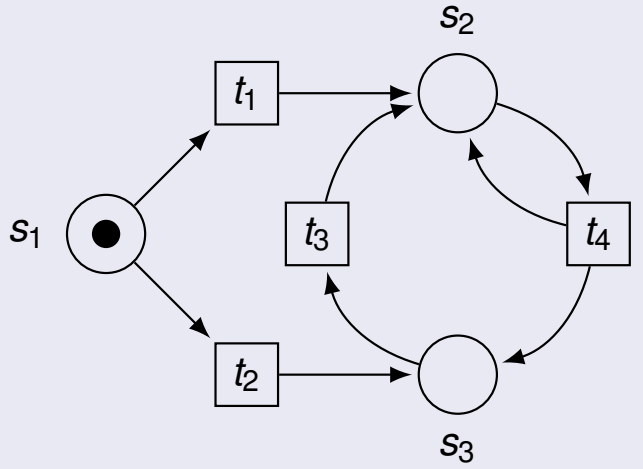
$\longrightarrow (1,0,0)$

$\longrightarrow (1,0,0)$

Petrinetze: Überdeckungsgraph

Ein Petrinetz kann mehrere Überdeckungsgraphen haben.

Hier für ein Beispiel, für das wir auch schon den Überdeckungsbaum gesehen haben:



Stellenordnung: s_1, s_2, s_3

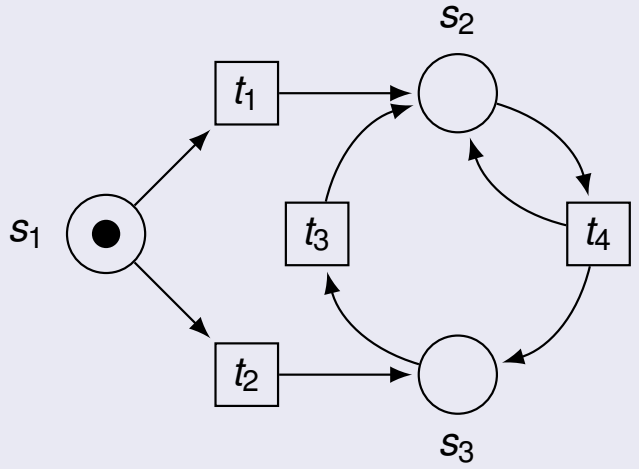
$$\longrightarrow (1,0,0) \xrightarrow{t_1} (0,1,0)$$

$$\longrightarrow (1,0,0) \xrightarrow{t_1} (0,1,0)$$

Petrinetze: Überdeckungsgraph

Ein Petrinetz kann mehrere Überdeckungsgraphen haben.

Hier für ein Beispiel, für das wir auch schon den Überdeckungsbaum gesehen haben:



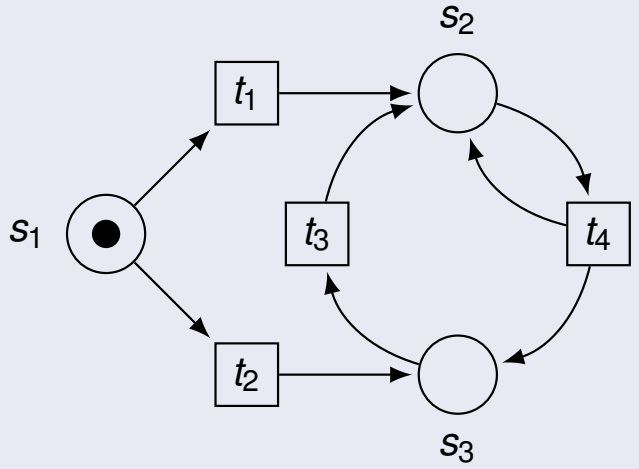
Stellenordnung: s_1, s_2, s_3

$$\longrightarrow (1,0,0) \xrightarrow{t_1} (0,1,0)$$
$$\qquad\qquad\quad\; \xrightarrow{t_2} (0,0,1)$$
$$\longrightarrow (1,0,0) \xrightarrow{t_1} (0,1,0)$$
$$\qquad\qquad\quad\; \xrightarrow{t_2} (0,0,1)$$

Petrinetze: Überdeckungsgraph

Ein Petrinetz kann mehrere Überdeckungsgraphen haben.

Hier für ein Beispiel, für das wir auch schon den Überdeckungsbaum gesehen haben:



Stellenordnung: s_1, s_2, s_3

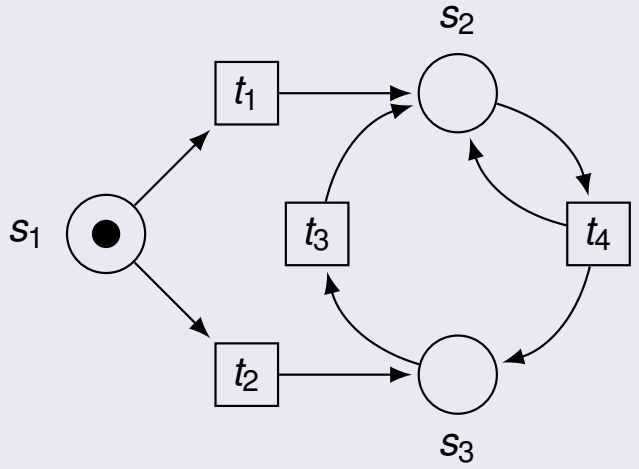
$$\longrightarrow (1,0,0) \xrightarrow{t_1} (0,1,0) \xrightarrow[t_4]{} (0,1,\omega)$$
$$\qquad\qquad\quad\ \ \searrow^{t_2} (0,0,1)$$

$$\longrightarrow (1,0,0) \xrightarrow{t_1} (0,1,0)$$
$$\qquad\qquad\quad\ \ \searrow^{t_2} (0,0,1)$$

Petrinetze: Überdeckungsgraph

Ein Petrinetz kann mehrere Überdeckungsgraphen haben.

Hier für ein Beispiel, für das wir auch schon den Überdeckungsbaum gesehen haben:



Stellenordnung: s_1, s_2, s_3

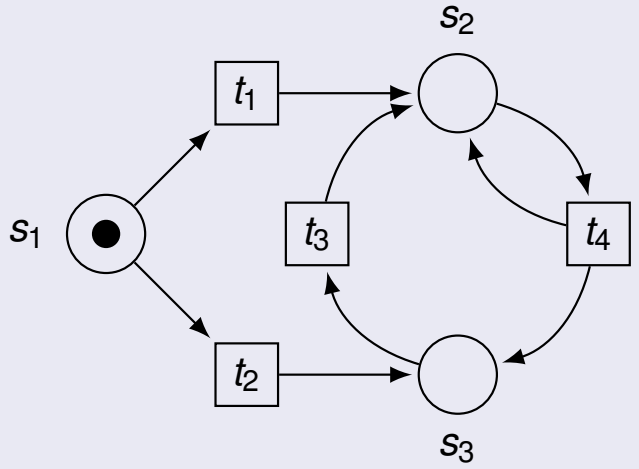
$$\longrightarrow (1,0,0) \xrightarrow{t_1} (0,1,0) \xrightarrow[t_4]{} (0,1,\omega)$$
$$\qquad\qquad\quad \xrightarrow{t_2} (0,0,1)$$

$$\longrightarrow (1,0,0) \xrightarrow{t_1} (0,1,0) \quad \uparrow t_3$$
$$\qquad\qquad\quad \xrightarrow{t_2} (0,0,1)$$

Petrinetze: Überdeckungsgraph

Ein Petrinetz kann mehrere Überdeckungsgraphen haben.

Hier für ein Beispiel, für das wir auch schon den Überdeckungsbaum gesehen haben:



Stellenordnung: s_1, s_2, s_3

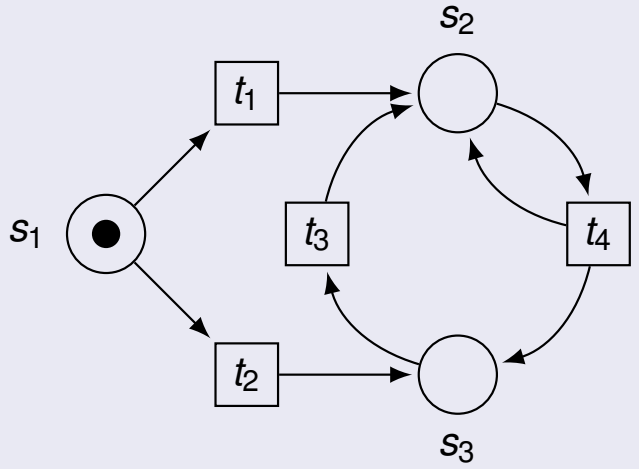
$$\longrightarrow (1,0,0) \xrightarrow{t_1} (0,1,0) \xrightarrow[t_4]{} (0,1,\omega)$$
$$\qquad\qquad\qquad\qquad\qquad\quad \uparrow t_3$$
$$\longrightarrow (1,0,0) \xrightarrow{t_2} (0,0,1)$$

$$\longrightarrow (1,0,0) \xrightarrow{t_1} (0,1,0)$$
$$\qquad\qquad\qquad\qquad\quad \uparrow t_3$$
$$\longrightarrow (1,0,0) \xrightarrow{t_2} (0,0,1)$$

Petrinetze: Überdeckungsgraph

Ein Petrinetz kann mehrere Überdeckungsgraphen haben.

Hier für ein Beispiel, für das wir auch schon den Überdeckungsbaum gesehen haben:



Stellenordnung: s_1, s_2, s_3

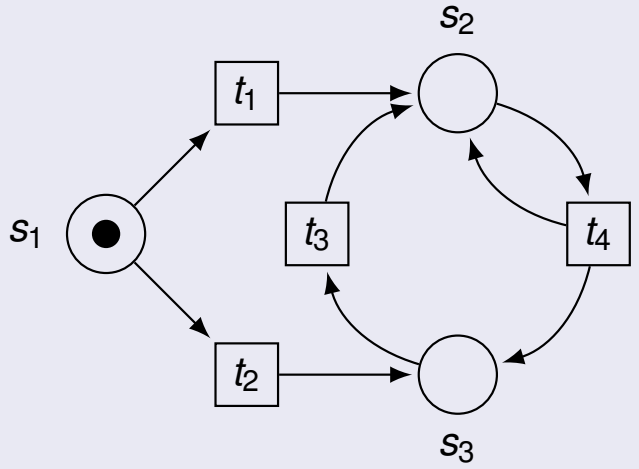
$$\longrightarrow (1,0,0) \xrightarrow{t_1} (0,1,0) \xrightarrow{t_4} (0,1,\omega)$$
$$\qquad\qquad\qquad\qquad\qquad\qquad\quad \uparrow t_3$$
$$\longrightarrow (1,0,0) \xrightarrow{t_2} (0,0,1)$$

$$\longrightarrow (1,0,0) \xrightarrow{t_1} (0,1,0) \xrightarrow{t_4} (0,\omega,\omega)$$
$$\qquad\qquad\qquad\qquad\qquad\qquad\quad \uparrow t_3$$
$$\longrightarrow (1,0,0) \xrightarrow{t_2} (0,0,1)$$

Petrinetze: Überdeckungsgraph

Ein Petrinetz kann mehrere Überdeckungsgraphen haben.

Hier für ein Beispiel, für das wir auch schon den Überdeckungsbaum gesehen haben:



Stellenordnung: s_1, s_2, s_3

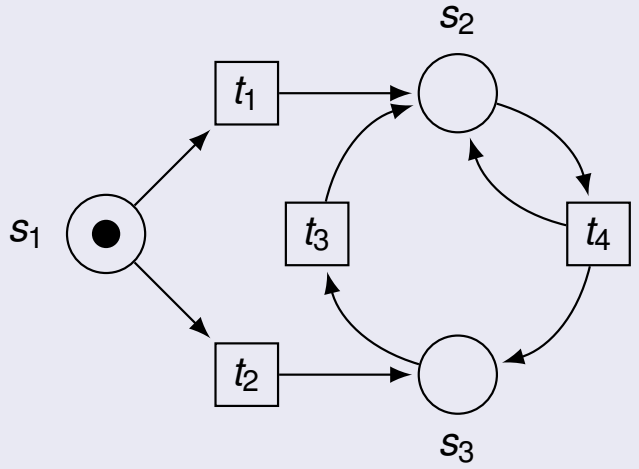
$$\longrightarrow (1,0,0) \xrightarrow{t_1} (0,1,0) \xrightarrow{t_4} (0,1,\omega) \xrightarrow{t_3} (0,\omega,\omega)$$
$$\longrightarrow (1,0,0) \xrightarrow{t_2} (0,0,1) \uparrow t_3$$

$$\longrightarrow (1,0,0) \xrightarrow{t_1} (0,1,0) \xrightarrow{t_4} (0,\omega,\omega)$$
$$\longrightarrow (1,0,0) \xrightarrow{t_2} (0,0,1) \uparrow t_3$$

Petrinetze: Überdeckungsgraph

Ein Petrinetz kann mehrere Überdeckungsgraphen haben.

Hier für ein Beispiel, für das wir auch schon den Überdeckungsbaum gesehen haben:



Stellenordnung: s_1, s_2, s_3

$$\longrightarrow (1,0,0) \xrightarrow{t_1} (0,1,0) \xrightarrow{t_4} (0,1,\omega) \xrightarrow{t_3} (0,\omega,\omega)$$

$$(1,0,0) \xrightarrow{t_2} (0,0,1) \xrightarrow{t_3} (0,1,0) \qquad (0,1,\omega) \xrightarrow{t_4} (0,1,\omega)$$

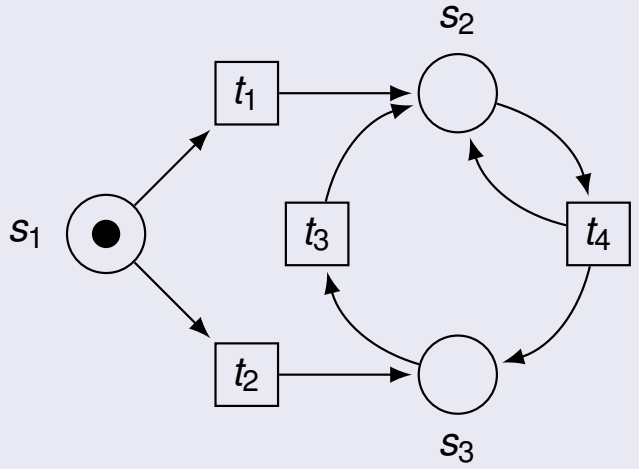
$$\longrightarrow (1,0,0) \xrightarrow{t_1} (0,1,0) \xrightarrow{t_4} (0,\omega,\omega)$$

$$(1,0,0) \xrightarrow{t_2} (0,0,1) \xrightarrow{t_3} (0,1,0)$$

Petrinetze: Überdeckungsgraph

Ein Petrinetz kann mehrere Überdeckungsgraphen haben.

Hier für ein Beispiel, für das wir auch schon den Überdeckungsbaum gesehen haben:



Stellenordnung: s_1, s_2, s_3

$$\longrightarrow (1,0,0) \xrightarrow{t_1} (0,1,0) \xrightarrow{t_4} (0,1,\omega) \xrightarrow{t_3} (0,\omega,\omega)$$

$$(0,1,0) \uparrow t_3 \qquad t_4 \circlearrowright \qquad t_3 \circlearrowright$$

$$\longrightarrow (1,0,0) \xrightarrow{t_2} (0,0,1)$$

$$\longrightarrow (1,0,0) \xrightarrow{t_1} (0,1,0) \xrightarrow{t_4} (0,\omega,\omega)$$

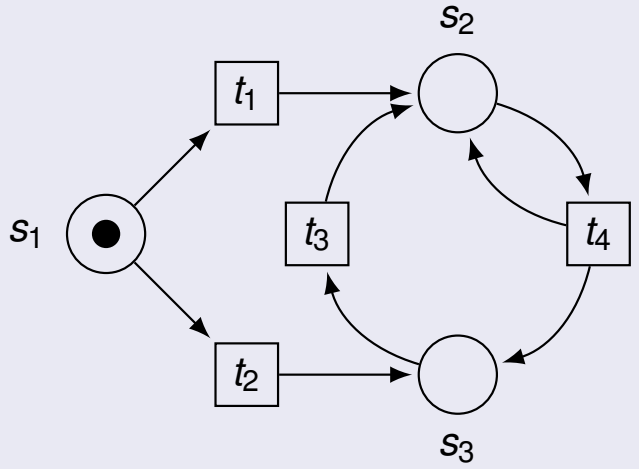
$$(0,1,0) \uparrow t_3$$

$$\longrightarrow (1,0,0) \xrightarrow{t_2} (0,0,1)$$

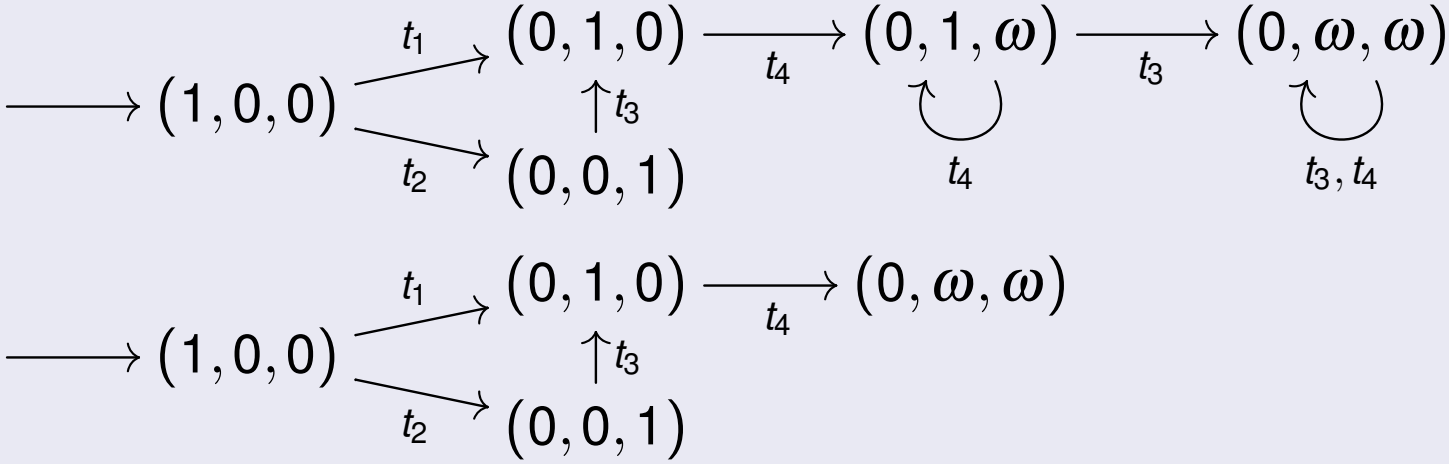
Petrinetze: Überdeckungsgraph

Ein Petrinetz kann mehrere Überdeckungsgraphen haben.

Hier für ein Beispiel, für das wir auch schon den Überdeckungsbaum gesehen haben:



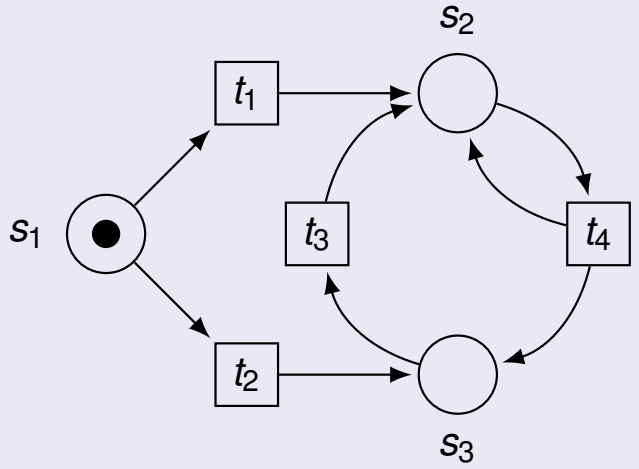
Stellenordnung: s_1, s_2, s_3



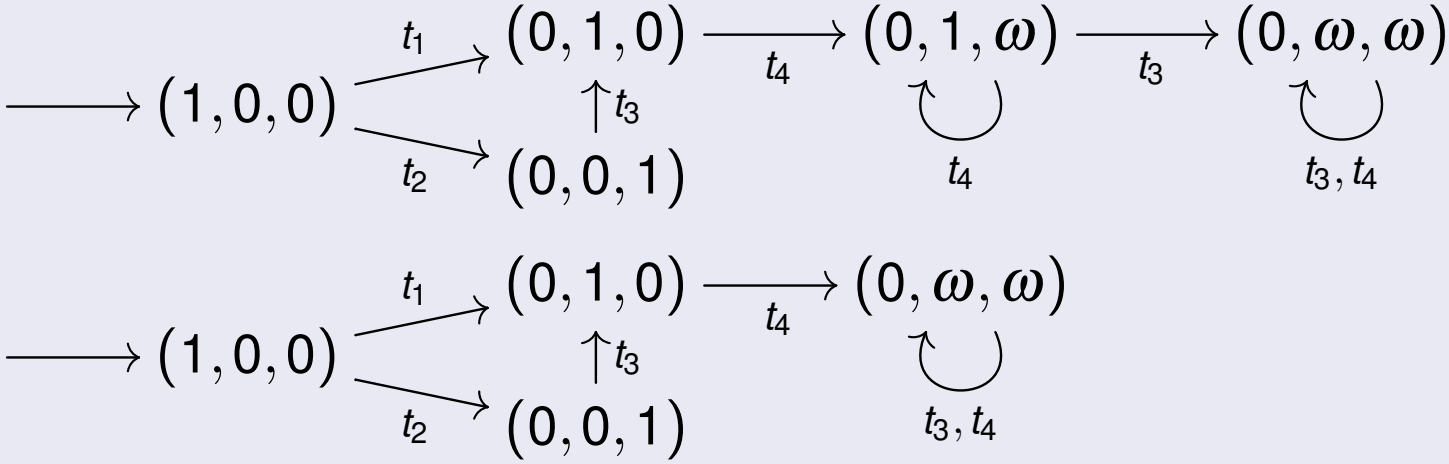
Petrinetze: Überdeckungsgraph

Ein Petrinetz kann mehrere Überdeckungsgraphen haben.

Hier für ein Beispiel, für das wir auch schon den Überdeckungsbaum gesehen haben:



Stellenordnung: s_1, s_2, s_3



Petrinetze: Überdeckungsgraph

Überdeckungsgraphen haben analoge nützliche Eigenschaften wie Überdeckungsbäume.

Petrinetze: Überdeckungsgraph

Überdeckungsgraphen haben analoge nützliche Eigenschaften wie Überdeckungsbäume.

Also insbesondere Termination der Erzeugung, die Möglichkeit zum Test auf Unbeschränktheit und auf schwache (aber nicht auf starke) Lebendigkeit eines Petrinetzes, und auf Kausalitäten,

Petrinetze: Überdeckungsgraph

Überdeckungsgraphen haben analoge nützliche Eigenschaften wie Überdeckungsbäume.

Also insbesondere Termination der Erzeugung, die Möglichkeit zum Test auf Unbeschränktheit und auf schwache (aber nicht auf starke) Lebendigkeit eines Petrinetzes, und auf Kausalitäten, sowie:

- Für jede erreichbare Markierung m eines Petrinetzes gibt es einen Knoten m' im Überdeckungsgraph mit $m \leq m'$.
- Für jeden Knoten m' im Überdeckungsgraph und jedes $c \in \mathbb{N}_0$ gibt es eine erreichbare Markierung m des Petrinetzes, so dass für alle Stellen s gilt:
  - $m(s) = m'(s)$, falls $m'(s) \neq \omega$
  - $m(s) > c$, falls $m'(s) = \omega$.

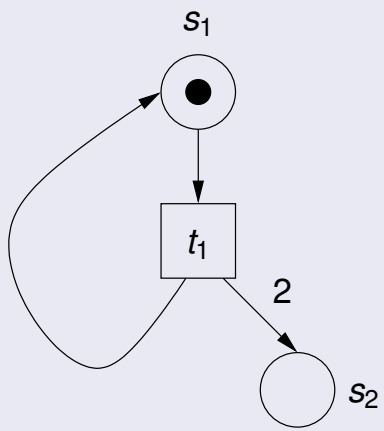
Petrinetze: Überdeckungsgraph

Frage: Ist für eine ω-Markierung m' im Überdeckungsgraph vielleicht sogar jede Markierung m des Petrinetzes mit $m \leq m'$ erreichbar?

Petrinetze: Überdeckungsgraph

Frage: Ist für eine ω-Markierung m' im Überdeckungsgraph vielleicht sogar jede Markierung m des Petrinetzes mit $m \leq m'$ erreichbar?

Nein! Gegenbeispiel:

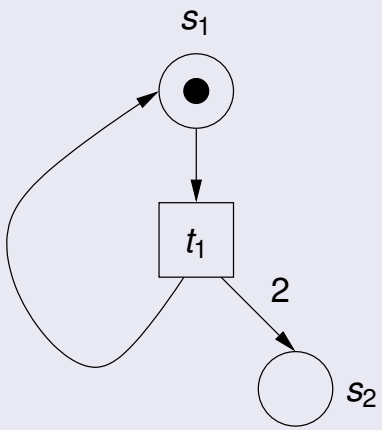


Stellenordnung: s_1, s_2

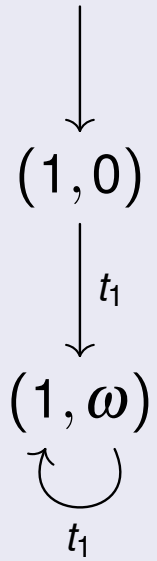
Petrinetze: Überdeckungsgraph

Frage: Ist für eine ω-Markierung m' im Überdeckungsgraph vielleicht sogar jede Markierung m des Petrinetzes mit $m \leq m'$ erreichbar?

Nein! Gegenbeispiel:



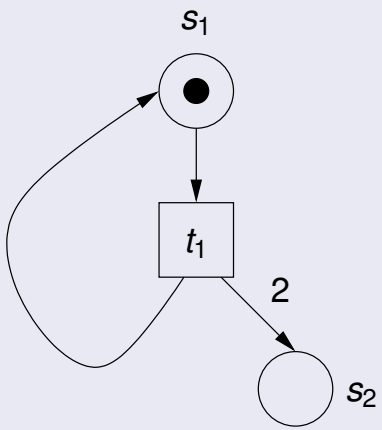
Stellenordnung: s_1, s_2



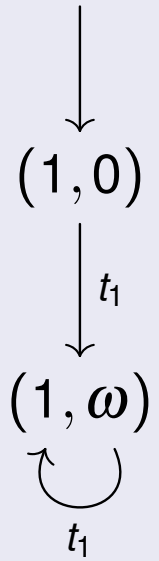
Petrinetze: Überdeckungsgraph

Frage: Ist für eine ω-Markierung m' im Überdeckungsgraph vielleicht sogar jede Markierung m des Petrinetzes mit $m \leq m'$ erreichbar?

Nein! Gegenbeispiel:



Stellenordnung: s_1, s_2



Die Markierung $(1,1)$ ist kleiner als $(1,\omega)$, ist aber in dem Petrinetz nicht erreichbar.

Petrinetze: Erreichbarkeitsproblem

Entscheidbarkeit des Erreichbarkeitsproblems

Es gibt ein Verfahren, das für ein gegebenes (unbeschränktes) Petrinetz N und eine Markierung m entscheidet, ob m in N erreichbar ist. (Mayr, 1984)

- Dieses Verfahren ist jedoch extrem aufwändig und in der Praxis derzeit nicht einsetzbar.

Petrinetze: Erreichbarkeitsproblem

Entscheidbarkeit des Erreichbarkeitsproblems

Es gibt ein Verfahren, das für ein gegebenes (unbeschränktes) Petrinetz N und eine Markierung m entscheidet, ob m in N erreichbar ist. (Mayr, 1984)

- Dieses Verfahren ist jedoch extrem aufwändig und in der Praxis derzeit nicht einsetzbar.
- Die obige Aussage bedeutet jedoch auch, dass Petrinetze nicht zu den mächtigsten Berechnungsmodellen gehören. Es gibt nämlich Berechnungsmodelle, für die das Erreichbarkeitsproblem nicht entscheidbar ist.

Petrinetze: Erreichbarkeitsproblem

Entscheidbarkeit des Erreichbarkeitsproblems

Es gibt ein Verfahren, das für ein gegebenes (unbeschränktes) Petrinetz N und eine Markierung m entscheidet, ob m in N erreichbar ist. (Mayr, 1984)

- Dieses Verfahren ist jedoch extrem aufwändig und in der Praxis derzeit nicht einsetzbar.
- Die obige Aussage bedeutet jedoch auch, dass Petrinetze nicht zu den mächtigsten Berechnungsmodellen gehören. Es gibt nämlich Berechnungsmodelle, für die das Erreichbarkeitsproblem nicht entscheidbar ist.

  Anders ausgedrückt: Petrinetze sind nicht Turing-mächtig ($\rightsquigarrow$ Vorlesung „Berechenbarkeit und Komplexität“).

Petrinetze: Erreichbarkeitsproblem

Entscheidbarkeit des Erreichbarkeitsproblems

Es gibt ein Verfahren, das für ein gegebenes (unbeschränktes) Petrinetz N und eine Markierung m entscheidet, ob m in N erreichbar ist. (Mayr, 1984)

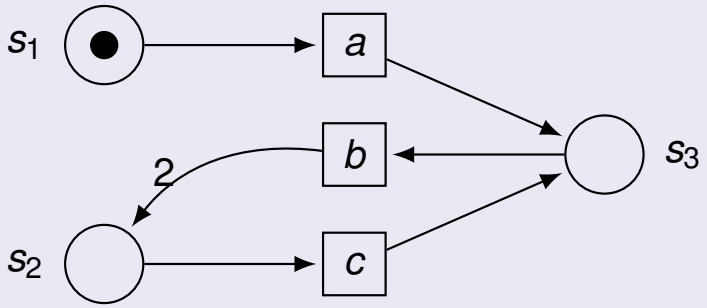
- Dieses Verfahren ist jedoch extrem aufwändig und in der Praxis derzeit nicht einsetzbar.
- Die obige Aussage bedeutet jedoch auch, dass Petrinetze nicht zu den mächtigsten Berechnungsmodellen gehören. Es gibt nämlich Berechnungsmodelle, für die das Erreichbarkeitsproblem nicht entscheidbar ist.

  Anders ausgedrückt: Petrinetze sind nicht Turing-mächtig ($\rightsquigarrow$ Vorlesung „Berechenbarkeit und Komplexität“).

  Das liegt vor allem daran, dass Petrinetze keine Nulltests folgender Form erlauben: „Die Transition t kann nur feuern, wenn in der Stelle s keine Marken liegen.“

Petrinetze: Überdeckungsgraph – Wiederholung

Ein anderes Beispiel mit Überdeckungsbaum:



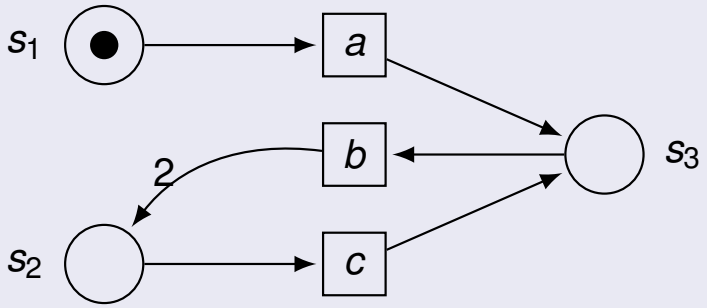
Stellenordnung: s_1, s_2, s_3

$\downarrow$

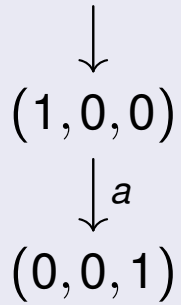
$(1, 0, 0)$

Petrinetze: Überdeckungsgraph – Wiederholung

Ein anderes Beispiel mit Überdeckungsbaum:

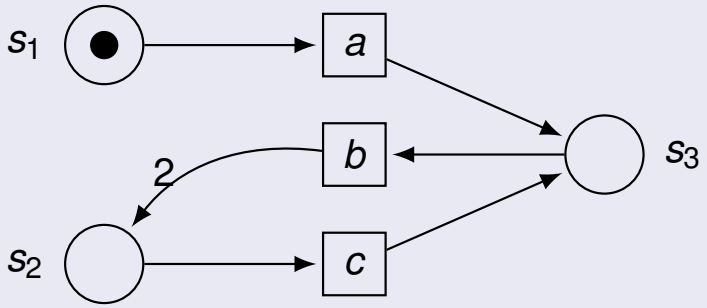


Stellenordnung: s_1, s_2, s_3



Petrinetze: Überdeckungsgraph – Wiederholung

Ein anderes Beispiel mit Überdeckungsbaum:

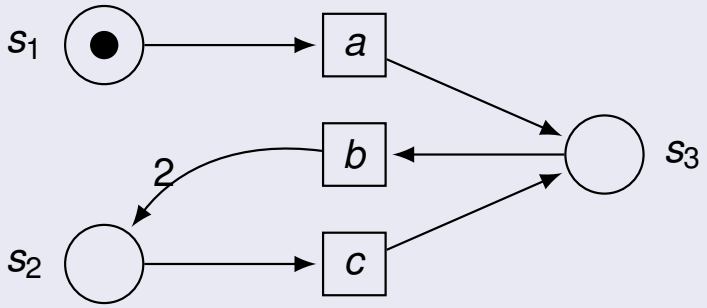


Stellenordnung: s_1, s_2, s_3

$$\downarrow$$
$$(1,0,0)$$
$$\downarrow a$$
$$(0,0,1)$$
$$\downarrow b$$
$$(0,2,0)$$

Petrinetze: Überdeckungsgraph – Wiederholung

Ein anderes Beispiel mit Überdeckungsbaum:

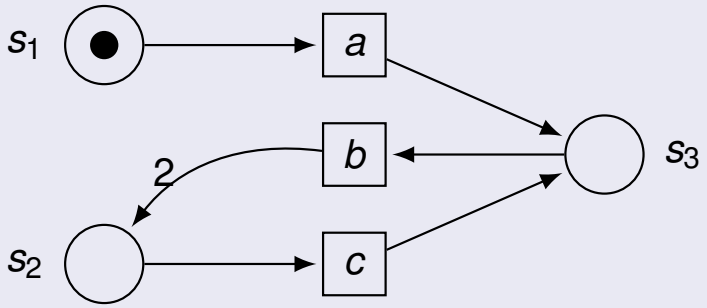


Stellenordnung: s_1, s_2, s_3

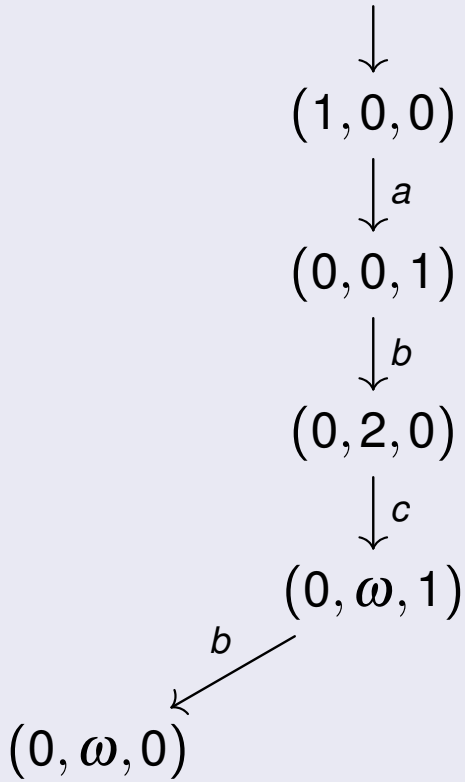
$$\downarrow$$
$$(1,0,0)$$
$$\downarrow a$$
$$(0,0,1)$$
$$\downarrow b$$
$$(0,2,0)$$
$$\downarrow c$$
$$(0,\omega,1)$$

Petrinetze: Überdeckungsgraph – Wiederholung

Ein anderes Beispiel mit Überdeckungsbaum:

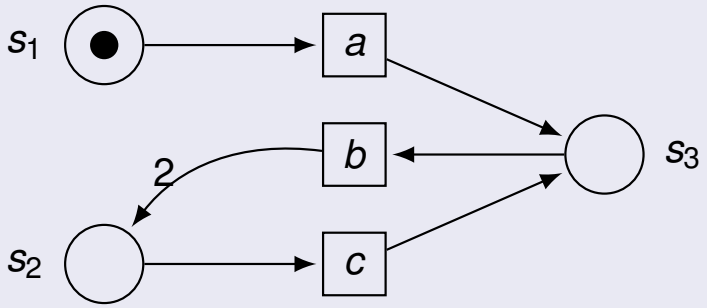


Stellenordnung: s_1, s_2, s_3

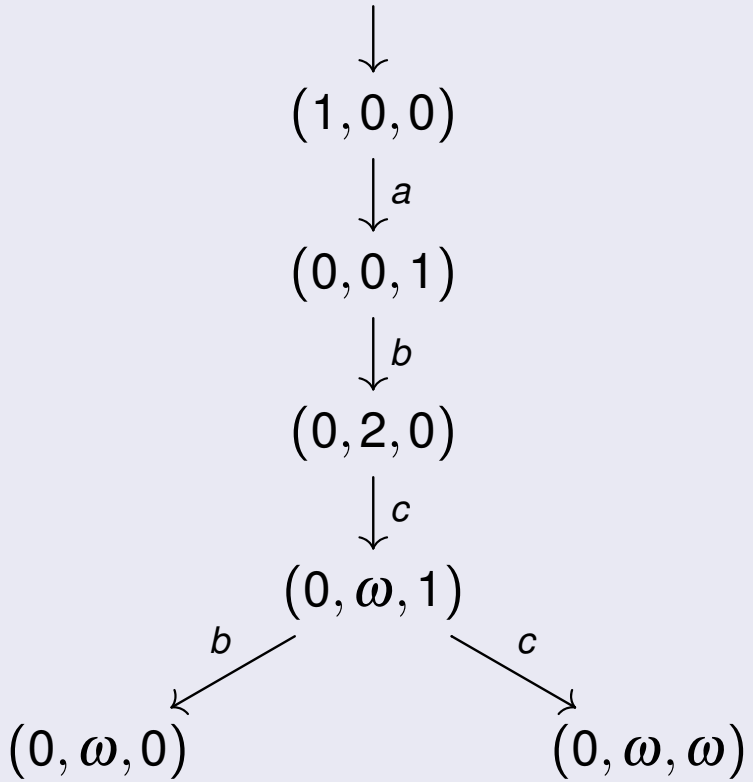


Petrinetze: Überdeckungsgraph – Wiederholung

Ein anderes Beispiel mit Überdeckungsbaum:

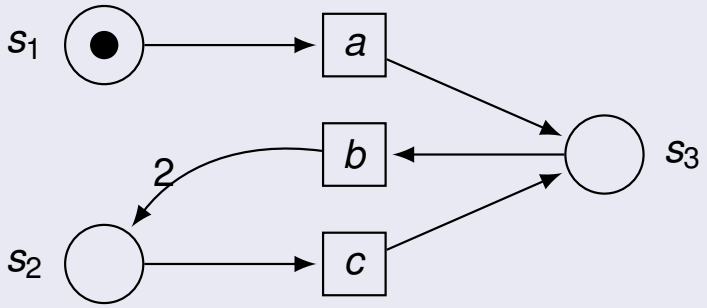


Stellenordnung: s_1, s_2, s_3

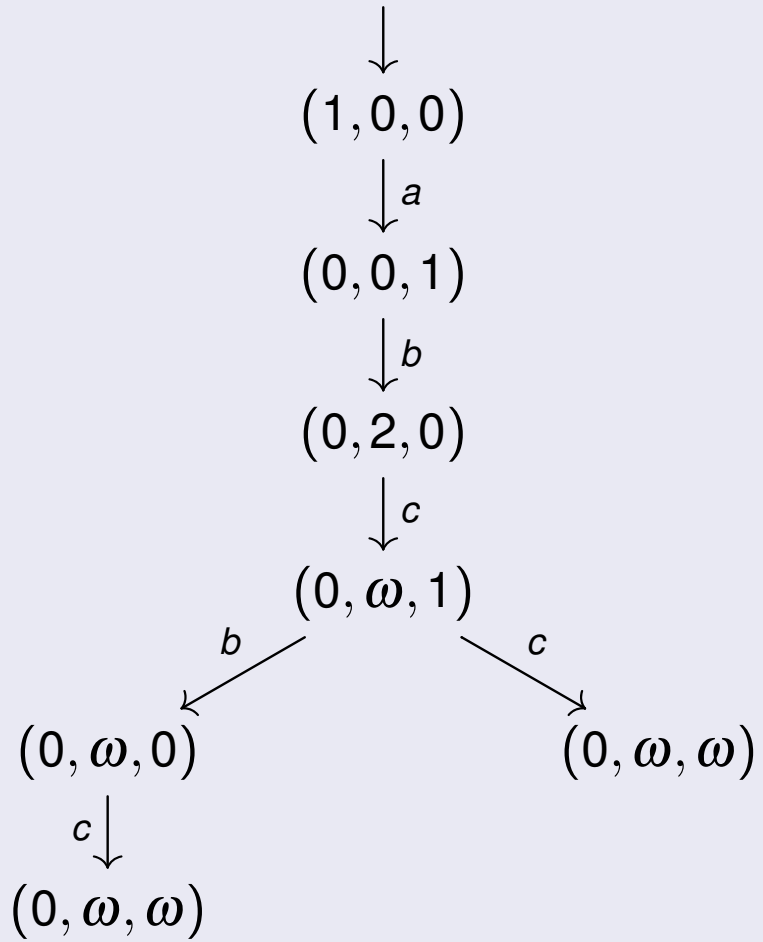


Petrinetze: Überdeckungsgraph – Wiederholung

Ein anderes Beispiel mit Überdeckungsbaum:

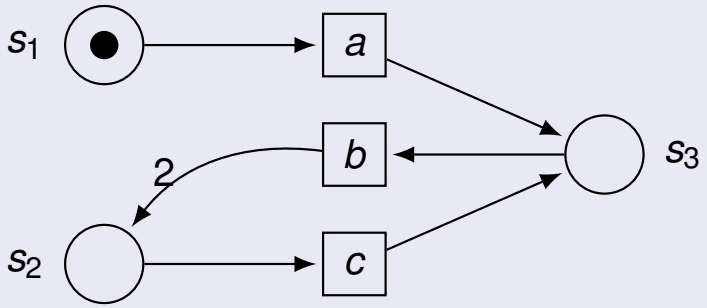


Stellenordnung: s_1, s_2, s_3

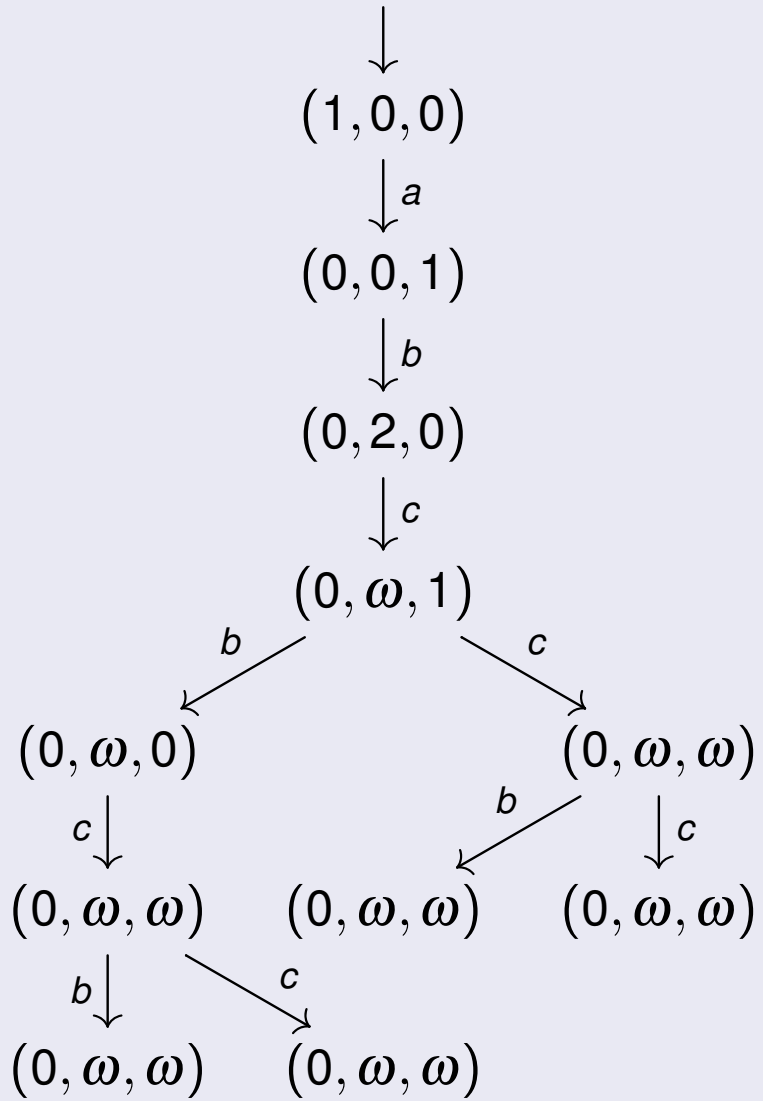


Petrinetze: Überdeckungsgraph – Wiederholung

Ein anderes Beispiel mit Überdeckungsbaum:

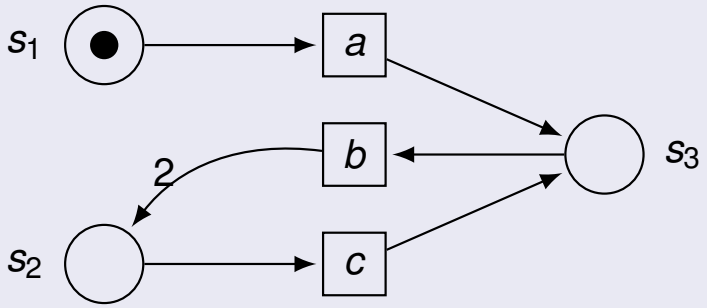


Stellenordnung: s_1, s_2, s_3



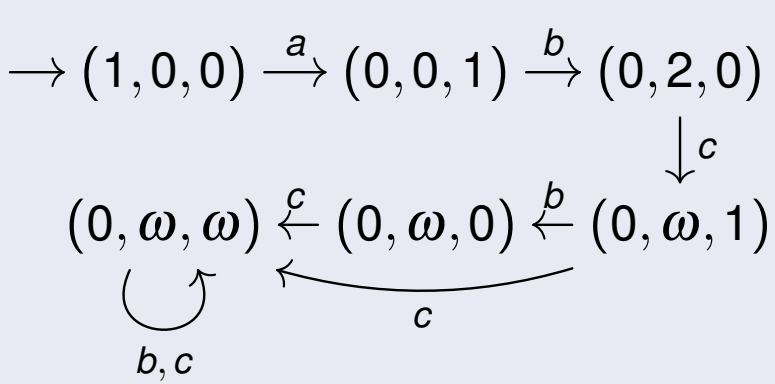
Petrinetze: Überdeckungsgraph – Wiederholung

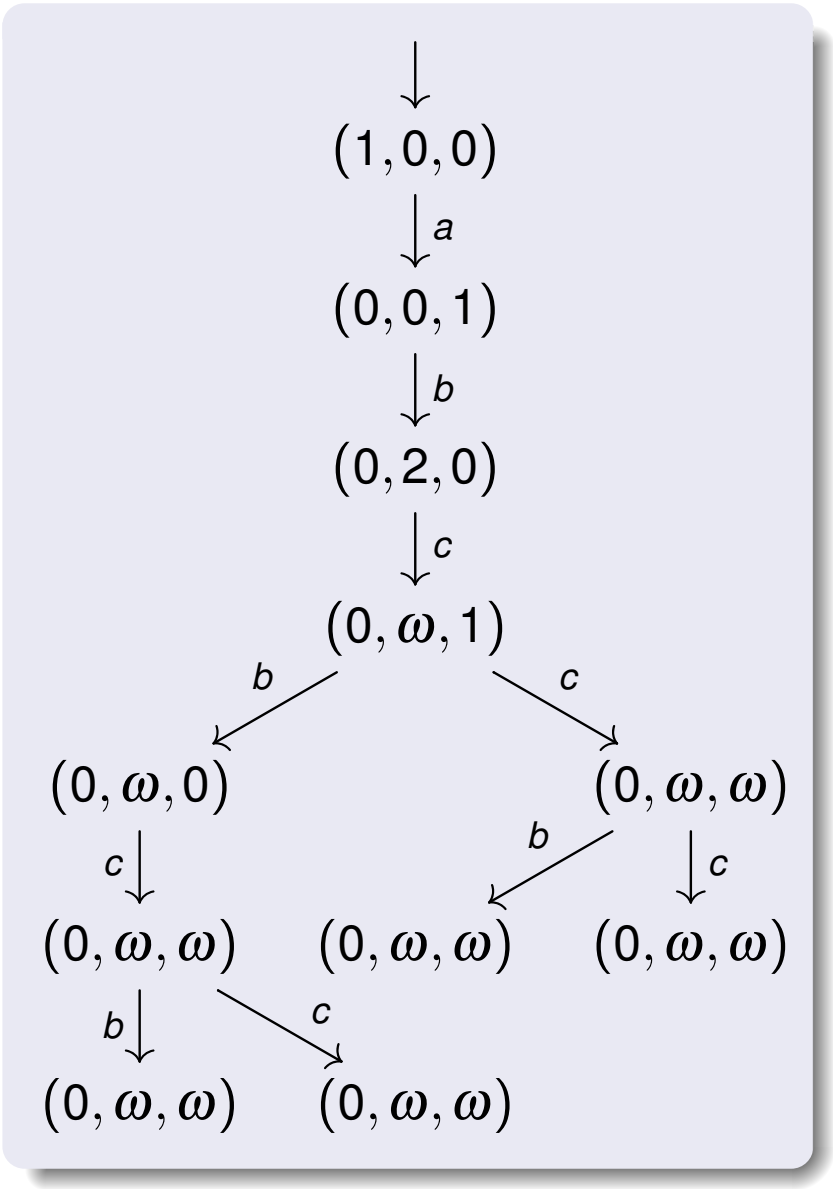
Ein anderes Beispiel mit Überdeckungsbaum:



Stellenordnung: s_1, s_2, s_3

und Überdeckungsgraph:





Petrinetze: Fallstudien (Wechselseitiger Ausschluss)

Wir betrachten zum Abschluss des Petrinetz-Teils der Vorlesung nun noch einige „Fallstudien“, also Beispiele, an denen typische Szenarien und spezielle Modellierungs-„Muster“ nochmals deutlich werden . . .

Petrinetze: Fallstudien (Wechselseitiger Ausschluss)

Wir betrachten zum Abschluss des Petrinetz-Teils der Vorlesung nun noch einige „Fallstudien“, also Beispiele, an denen typische Szenarien und spezielle Modellierungs-„Muster“ nochmals deutlich werden . . .

Zunächst behandeln wir das Konzept des wechselseitigen Ausschlusses (engl. mutual exclusion).

- Wir betrachten zwei Akteure, die jeweils einen kritischen Bereich haben.

Petrinetze: Fallstudien (Wechselseitiger Ausschluss)

Wir betrachten zum Abschluss des Petrinetz-Teils der Vorlesung nun noch einige „Fallstudien“, also Beispiele, an denen typische Szenarien und spezielle Modellierungs-„Muster“ nochmals deutlich werden . . .

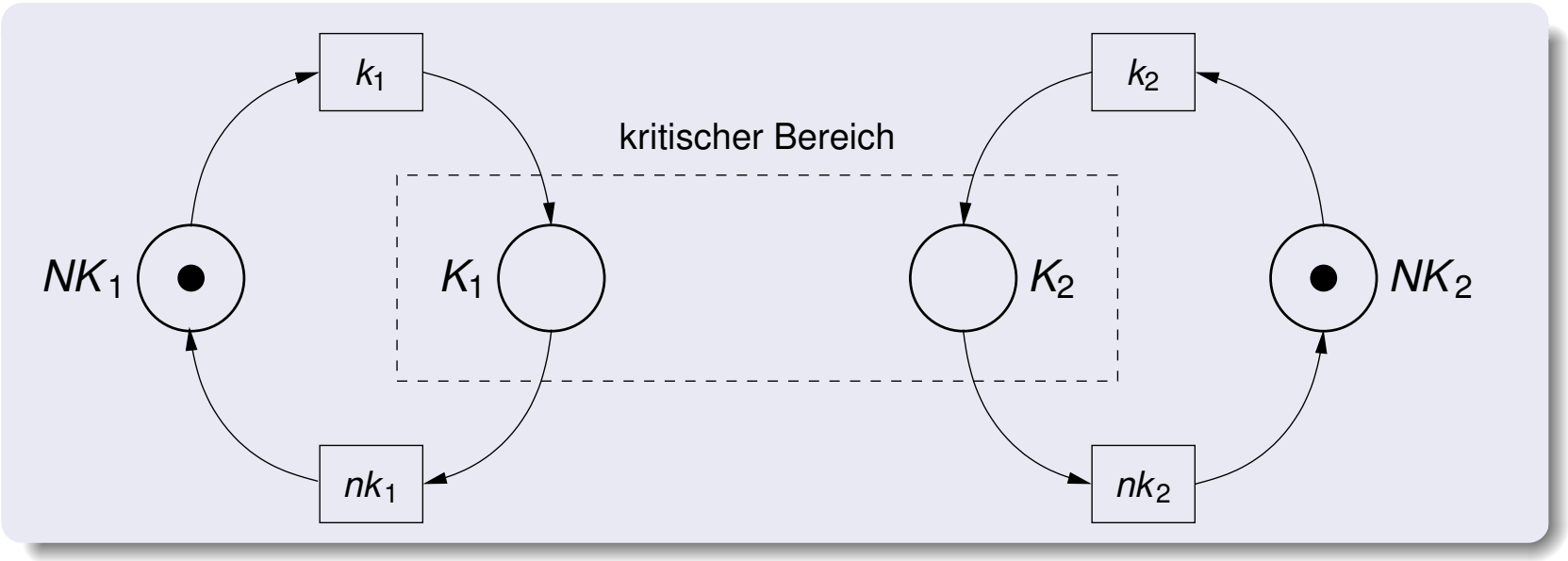
Zunächst behandeln wir das Konzept des wechselseitigen Ausschlusses (engl. mutual exclusion).

- Wir betrachten zwei Akteure, die jeweils einen kritischen Bereich haben.
- Beide Akteure dürfen nicht gleichzeitig in ihren kritischen Bereich kommen, da sie sich dort gegenseitig behindern und unerwünschtes Verhalten auslösen würden (z.B. indem beide Akteure in dieselbe Datei schreiben).

  Es darf sich also immer höchstens ein Akteur im kritischen Bereich befinden.

Petrinetze: Fallstudien (Wechselseitiger Ausschluss)

Ursprüngliches System:

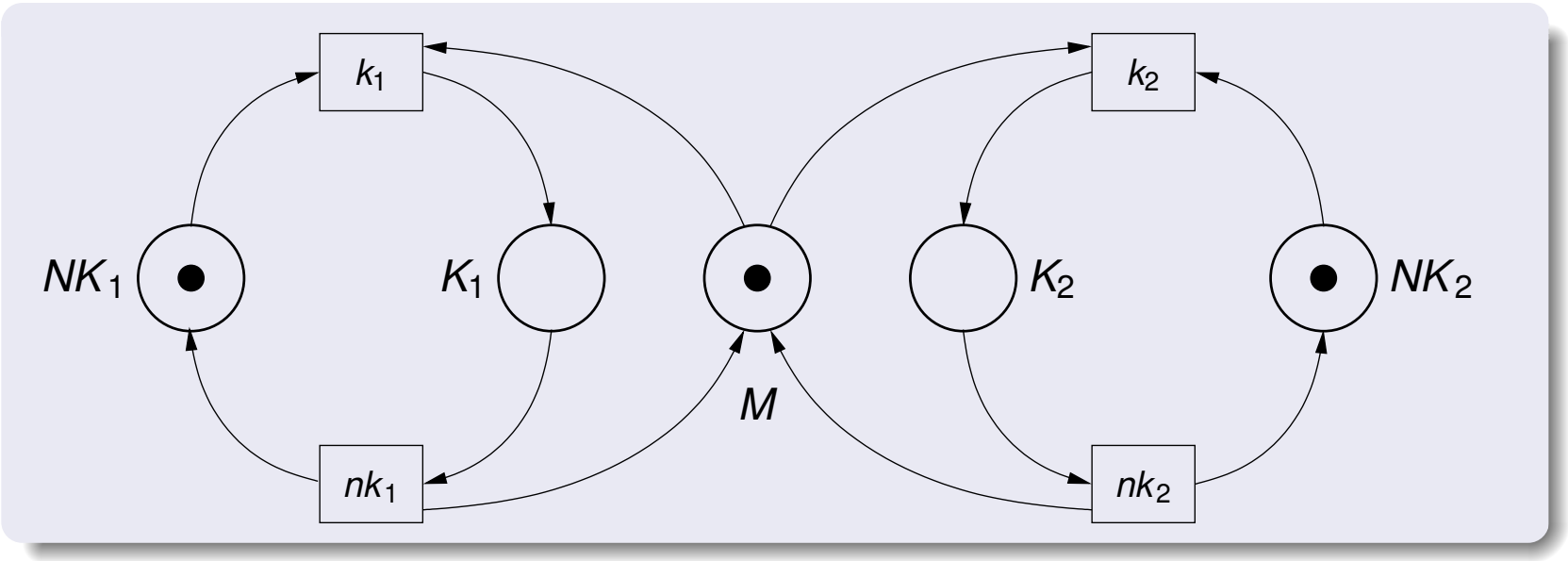


Bedeutung der Stellen:

K_1: kritischer Bereich Akteur 1
NK_1: nicht-kritischer Bereich Akteur 1
K_2: krititischer Bereich Akteur 2
NK_2: nicht-kritischer Bereich Akteur 2

Petrinetze: Fallstudien (Wechselseitiger Ausschluss)

Erweitertes System mit Synchronisation:



Bedeutung der Stellen:

- K_1: kritischer Bereich Akteur 1
- NK_1: nicht-kritischer Bereich Akteur 1
- K_2: krititischer Bereich Akteur 2
- NK_2: nicht-kritischer Bereich Akteur 2
- M: Hilfsstelle, sogenannter Mutex

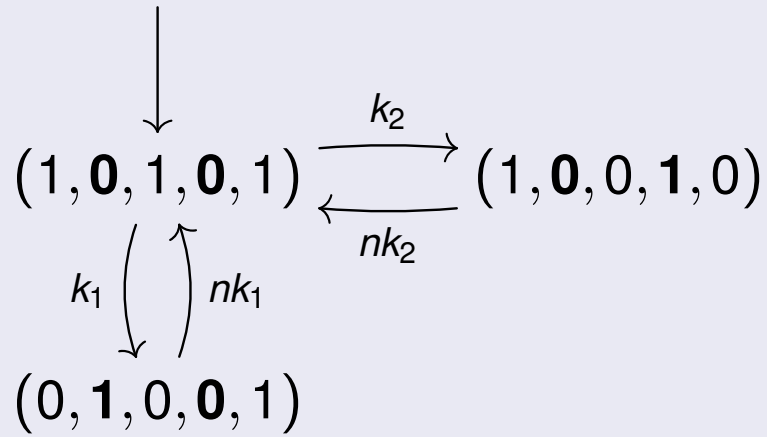
Petrinetze: Fallstudien (Wechselseitiger Ausschluss)

Wir möchten zeigen, dass in den Stellen K_1, K_2 niemals gleichzeitig Marken liegen.

Petrinetze: Fallstudien (Wechselseitiger Ausschluss)

Wir möchten zeigen, dass in den Stellen K_1, K_2 niemals gleichzeitig Marken liegen.

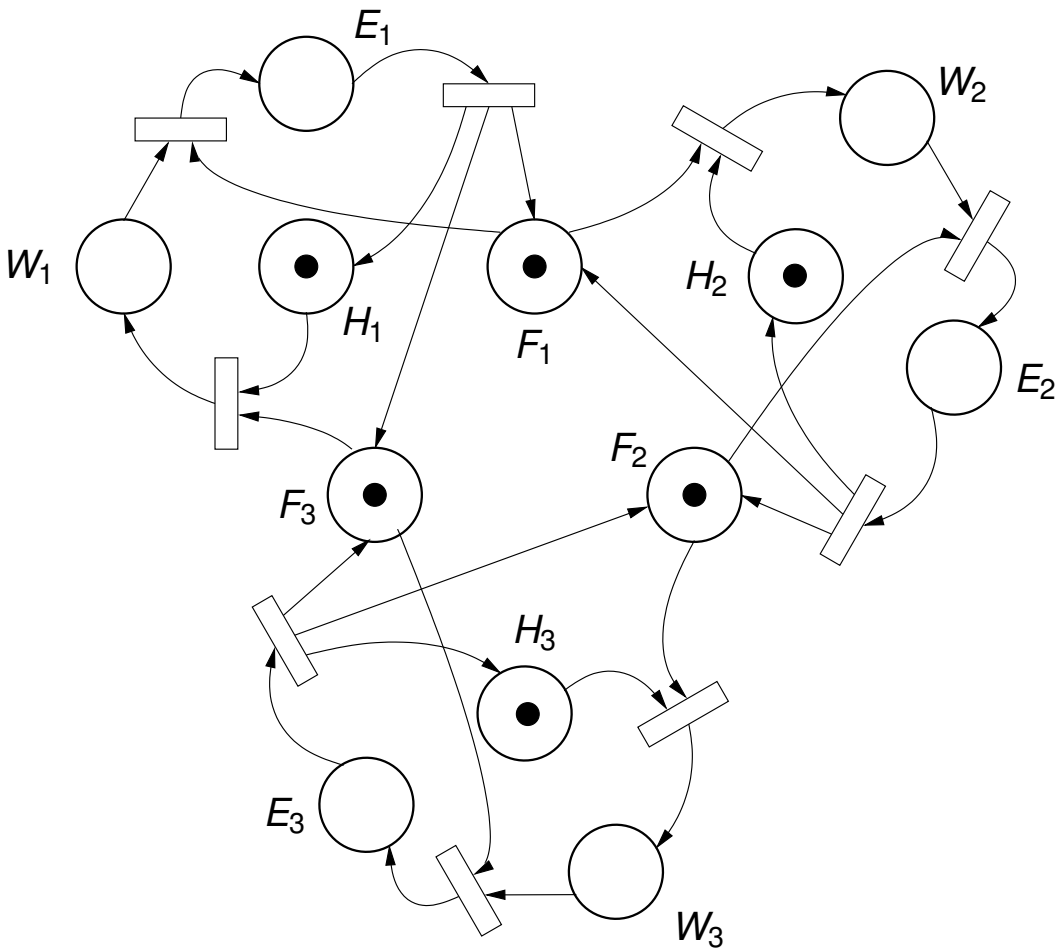
Erreichbarkeitsgraph:

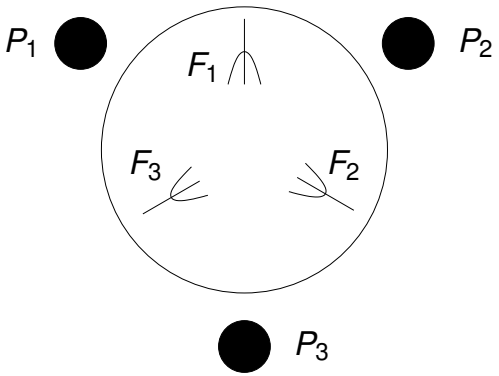


Stellenordnung: NK_1, K_1, M, K_2, NK_2

Petrinetze: Fallstudien (Speisende Philosophen)

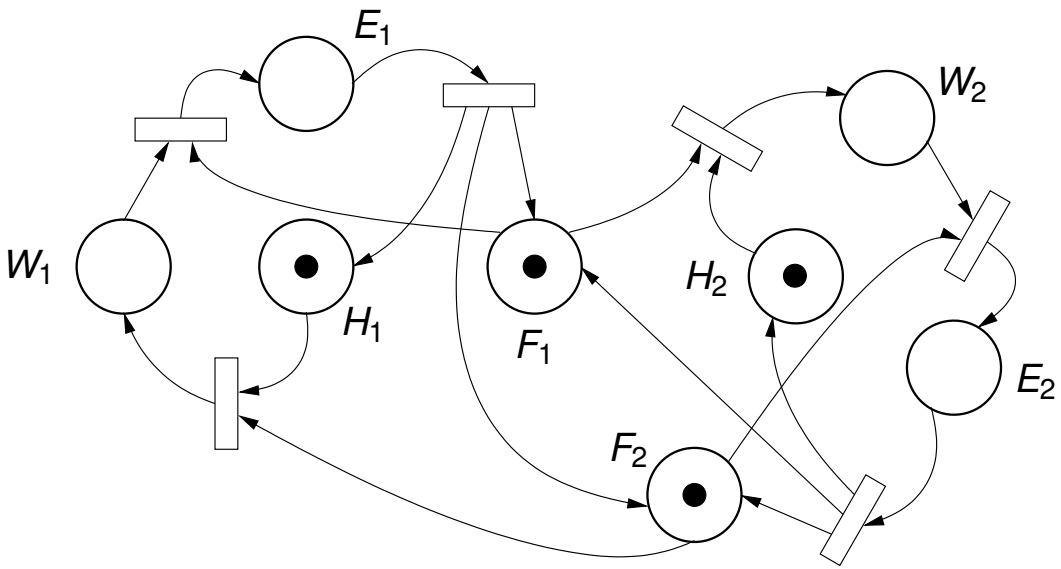
Wir kommen wieder auf die speisenden Philosophen zurück,

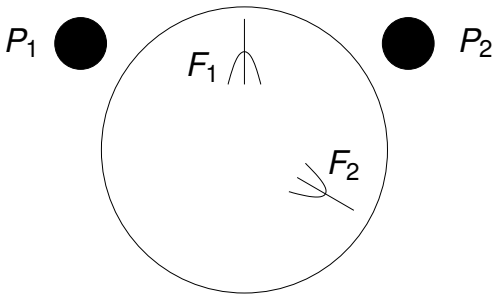




Petrinetze: Fallstudien (Speisende Philosophen)

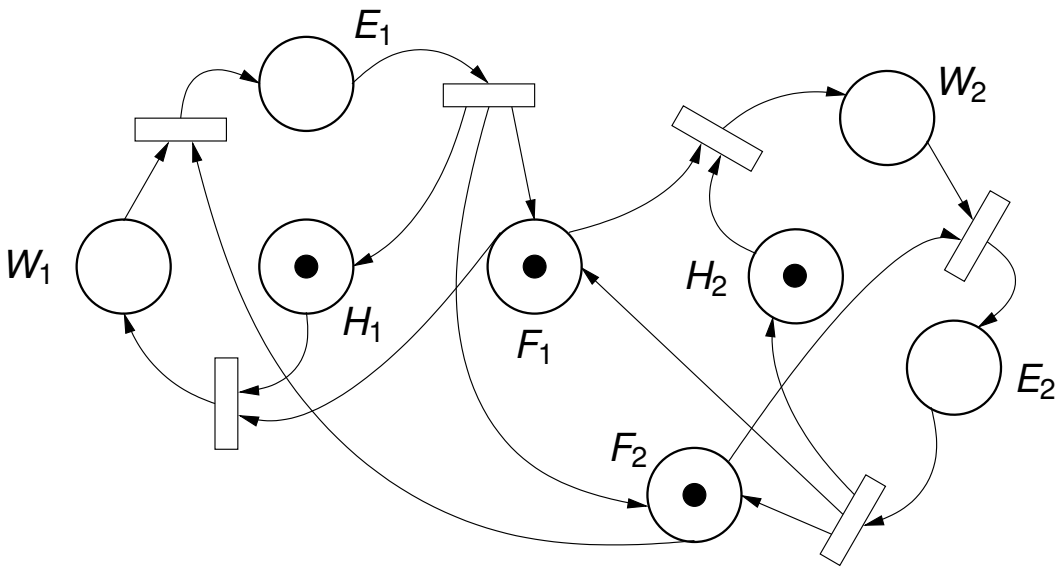
Wir kommen wieder auf die speisenden Philosophen zurück, aber schicken den dritten Philosophen nach Hause:



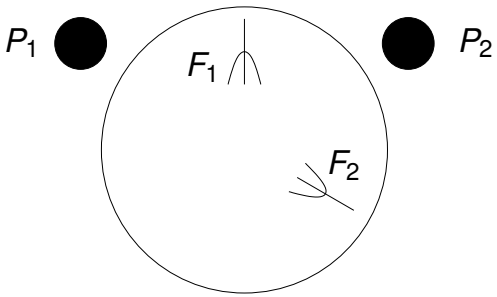


Petrinetze: Fallstudien (Speisende Philosophen)

Wir kommen wieder auf die speisenden Philosophen zurück, aber schicken den dritten Philosophen nach Hause:



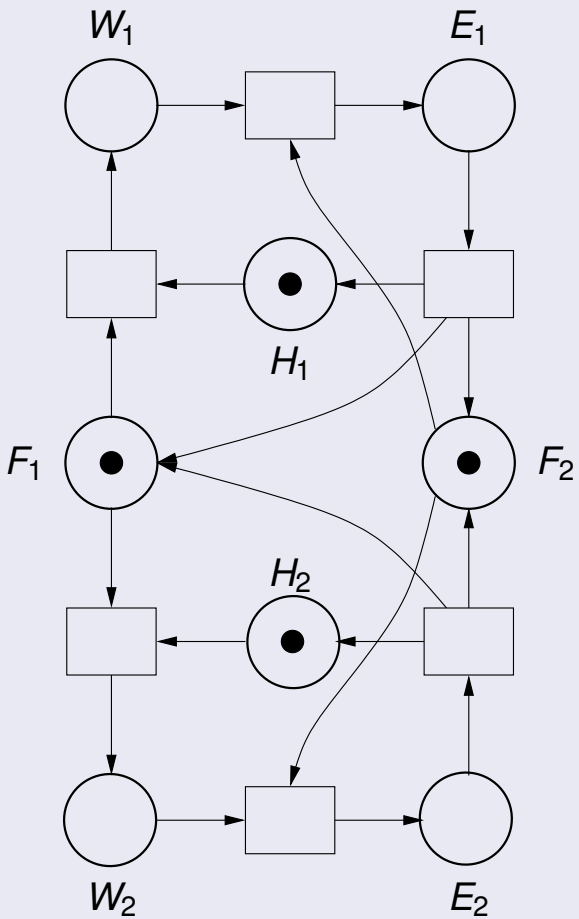
Außerdem machen wir den ersten Philosophen zum Linkshänder (er nimmt die linke Gabel zuerst).



Petrinetze: Fallstudien (Speisende Philosophen)

Anders dargestellt:

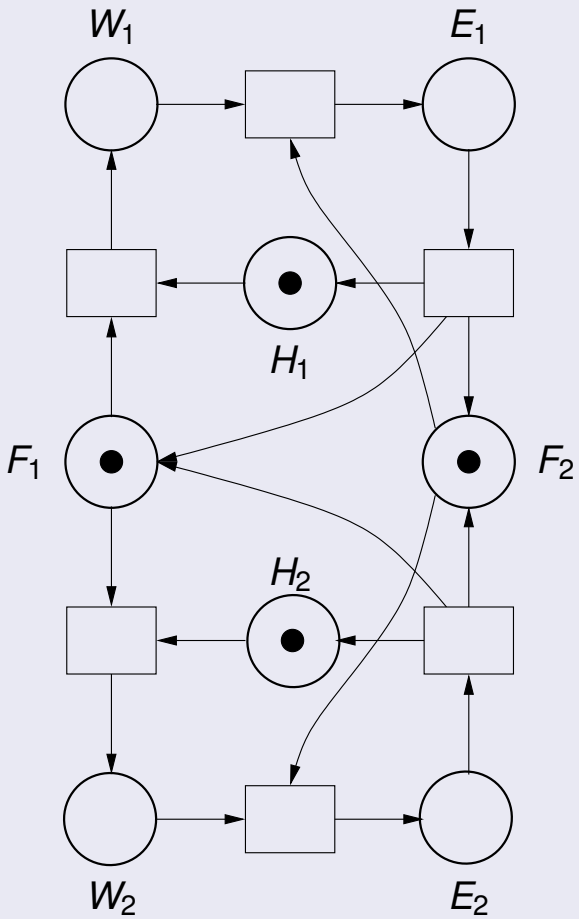
Links- und rechtshändige Philosophen



Petrinetze: Fallstudien (Speisende Philosophen)

Anders dargestellt:

Links- und rechtshändige Philosophen



Verklemmungsfrei durch Synchronisation!

Petrinetze: Fallstudien (Kapazitätsbegrenzung)

Ein anderes Erfordernis, das wir immer wieder mal ausdrücken wollten (z.B. beim Keksautomat), war die Begrenzung der Kapazität einzelner Stellen im Petrinetz.

Petrinetze: Fallstudien (Kapazitätsbegrenzung)

Ein anderes Erfordernis, das wir immer wieder mal ausdrücken wollten (z.B. beim Keksautomat), war die Begrenzung der Kapazität einzelner Stellen im Petrinetz.

Eine Möglichkeit zum Umgang damit ist die Einführung einer speziellen Art von Petrinetzen:

Petrinetz mit Kapazitäten (Definition)

Ein Petrinetz mit Kapazitäten besteht aus einem (herkömmlichen) Petrinetz, mit Stellenmenge S, und einer Kapazitätsfunktion $k : S \to \mathbb{N}_0$.

Petrinetze: Fallstudien (Kapazitätsbegrenzung)

Ein anderes Erfordernis, das wir immer wieder mal ausdrücken wollten (z.B. beim Keksautomat), war die Begrenzung der Kapazität einzelner Stellen im Petrinetz.

Eine Möglichkeit zum Umgang damit ist die Einführung einer speziellen Art von Petrinetzen:

Petrinetz mit Kapazitäten (Definition)

Ein Petrinetz mit Kapazitäten besteht aus einem (herkömmlichen) Petrinetz, mit Stellenmenge S, und einer Kapazitätsfunktion $k : S \to \mathbb{N}_0$. Für die Anfangsmarkierung m_0 muss gelten: $m_0 \leq k$.

Petrinetze: Fallstudien (Kapazitätsbegrenzung)

Ein anderes Erfordernis, das wir immer wieder mal ausdrücken wollten (z.B. beim Keksautomat), war die Begrenzung der Kapazität einzelner Stellen im Petrinetz.

Eine Möglichkeit zum Umgang damit ist die Einführung einer speziellen Art von Petrinetzen:

Petrinetz mit Kapazitäten (Definition)

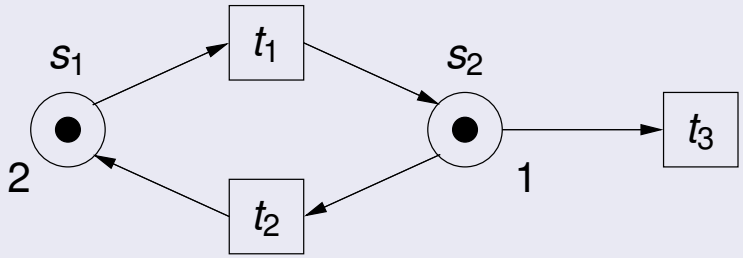
Ein Petrinetz mit Kapazitäten besteht aus einem (herkömmlichen) Petrinetz, mit Stellenmenge S, und einer Kapazitätsfunktion $k : S \to \mathbb{N}_0$. Für die Anfangsmarkierung m_0 muss gelten: $m_0 \leq k$.

Intuition: Jede Stelle s darf höchstens $k(s)$ Marken enthalten.

In der grafischen Darstellung werden die Kapazitäten an die Stellen geschrieben.

Petrinetze: Fallstudien (Kapazitätsbegrenzung)

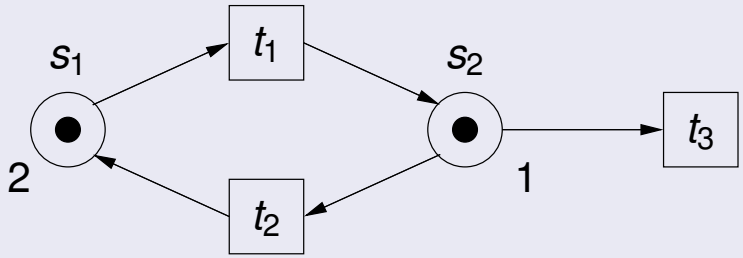
Beispielnetz mit Kapazitäten



$k(s_1) = 2, k(s_2) = 1$

Petrinetze: Fallstudien (Kapazitätsbegrenzung)

Beispielnetz mit Kapazitäten



$k(s_1) = 2, k(s_2) = 1$

Natürlich muss die Dynamik angepasst werden:

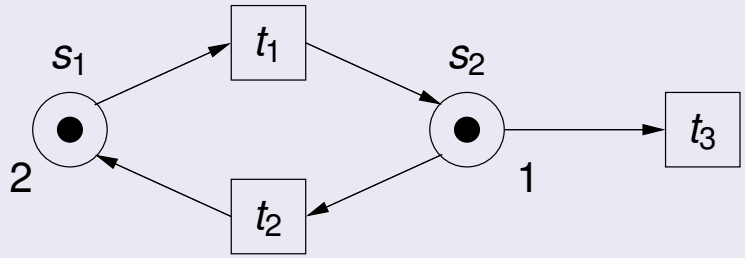
Aktivierung/Schalten bei Petrinetzen mit Kapazitäten (Definition)

Eine Transition t ist für eine Markierung m aktiviert, wenn gilt:

1. ${}^\bullet t \leq m$
2. und $m \ominus {}^\bullet t \oplus t^\bullet \leq k$.

Petrinetze: Fallstudien (Kapazitätsbegrenzung)

Beispielnetz mit Kapazitäten



$k(s_1) = 2, k(s_2) = 1$

Natürlich muss die Dynamik angepasst werden:

Aktivierung/Schalten bei Petrinetzen mit Kapazitäten (Definition)

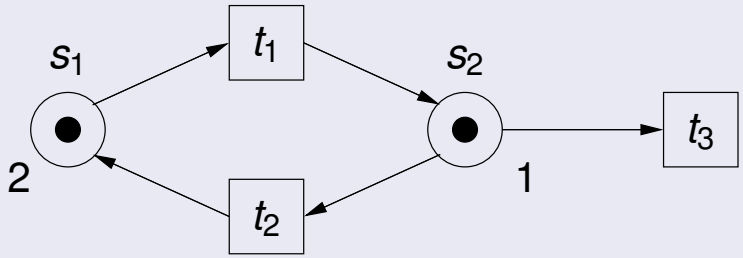
Eine Transition t ist für eine Markierung m aktiviert, wenn gilt:

1. ${}^\bullet t \leq m$
2. und $m \ominus {}^\bullet t \oplus t^\bullet \leq k$.

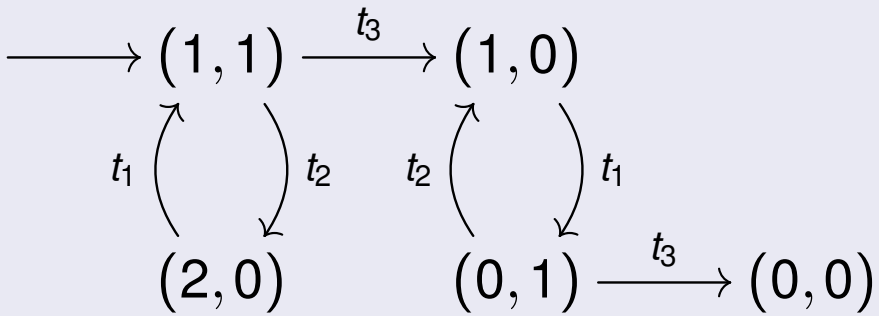
Das heißt, eine Transition darf nur dann schalten, wenn dadurch die Kapazitäten nicht überschritten werden.

Petrinetze: Fallstudien (Kapazitätsbegrenzung)

Beispielnetz mit Kapazitäten



Erreichbarkeitsgraph



Stellenordnung: s_1, s_2

Insbesondere: Für die Anfangsmarkierung $(1,1)$ ist die Transition t_1 nicht aktiviert.

Petrinetze: Fallstudien (Kapazitätsbegrenzung)

Es geht jedoch auch ohne Einführung einer neuen Petrinetz-Art!

Petrinetze: Fallstudien (Kapazitätsbegrenzung)

Es geht jedoch auch ohne Einführung einer neuen Petrinetz-Art!

Umwandlung eines Petrinetzes mit Kapazitäten in eines ohne

1. Füge zu jeder Stelle s eine sogenannte Komplementstelle $\overline{s}$ hinzu. In der neuen Anfangsmarkierung enthält $\overline{s}$ genau $k(s) - m_0(s)$ Marken.

   Idee: Die Summe der Marken in der Stelle und der Komplementstelle ergibt immer die gewünschte Kapazität.

Petrinetze: Fallstudien (Kapazitätsbegrenzung)

Es geht jedoch auch ohne Einführung einer neuen Petrinetz-Art!

Umwandlung eines Petrinetzes mit Kapazitäten in eines ohne

1. Füge zu jeder Stelle s eine sogenannte Komplementstelle $\overline{s}$ hinzu. In der neuen Anfangsmarkierung enthält $\overline{s}$ genau $k(s) - m_0(s)$ Marken.

   Idee: Die Summe der Marken in der Stelle und der Komplementstelle ergibt immer die gewünschte Kapazität.

2. Falls eine Transition t insgesamt Marken aus einer Stelle s herausnimmt, $n = t^\bullet(s) - {}^\bullet t(s) < 0$, füge eine Kante von t nach $\overline{s}$ mit Gewicht $-n$ ein.

Petrinetze: Fallstudien (Kapazitätsbegrenzung)

Es geht jedoch auch ohne Einführung einer neuen Petrinetz-Art!

Umwandlung eines Petrinetzes mit Kapazitäten in eines ohne

1. Füge zu jeder Stelle s eine sogenannte Komplementstelle $\overline{s}$ hinzu. In der neuen Anfangsmarkierung enthält $\overline{s}$ genau $k(s) - m_0(s)$ Marken.

   Idee: Die Summe der Marken in der Stelle und der Komplementstelle ergibt immer die gewünschte Kapazität.

2. Falls eine Transition t insgesamt Marken aus einer Stelle s herausnimmt, $n = t^\bullet(s) - {}^\bullet t(s) < 0$, füge eine Kante von t nach $\overline{s}$ mit Gewicht $-n$ ein.

3. Falls eine Transition t insgesamt Marken in eine Stelle s hineinlegt, $n = t^\bullet(s) - {}^\bullet t(s) > 0$, füge eine Kante von $\overline{s}$ nach t mit Gewicht n ein.

Petrinetze: Fallstudien (Kapazitätsbegrenzung)

Mit dieser Konstruktion ist für jede erreichbare Markierung m sichergestellt, dass:

1. $m(s) + m(\overline{s}) = k(s)$ für jedes Paar $s, \overline{s}$;

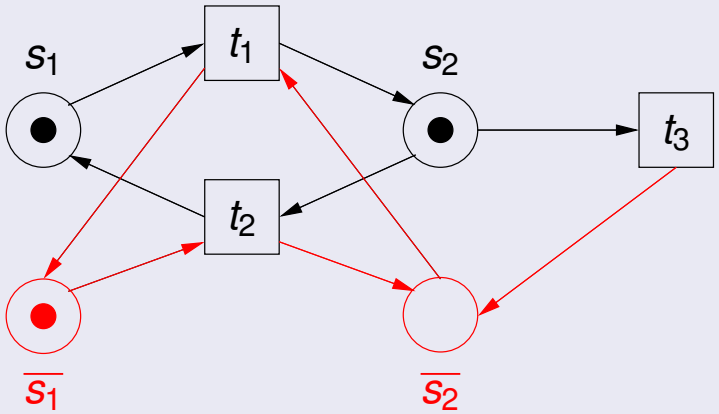
Petrinetze: Fallstudien (Kapazitätsbegrenzung)

Mit dieser Konstruktion ist für jede erreichbare Markierung m sichergestellt, dass:

1. $m(s) + m(\overline{s}) = k(s)$ für jedes Paar $s, \overline{s}$;
2. eine Transition t nur schaltbar ist, wenn die Kapazitäten der Stellen in der Nachbedingung noch nicht ausgeschöpft sind. Das wird dadurch überprüft, dass die benötigten Restkapazitäten über die Komplementstellen abgefragt werden.

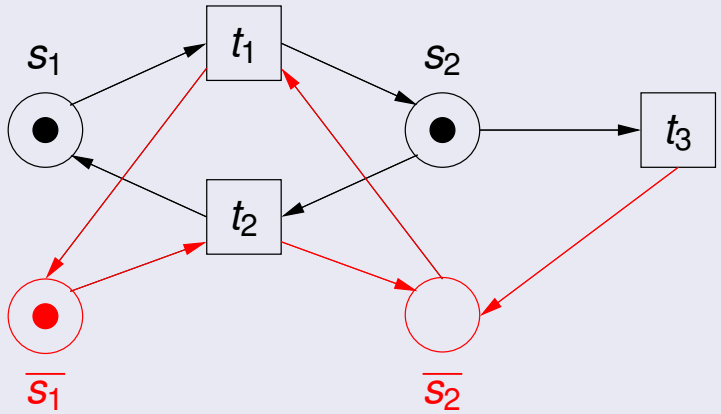
Petrinetze: Fallstudien (Kapazitätsbegrenzung)

Umgewandeltes Beispielnetz

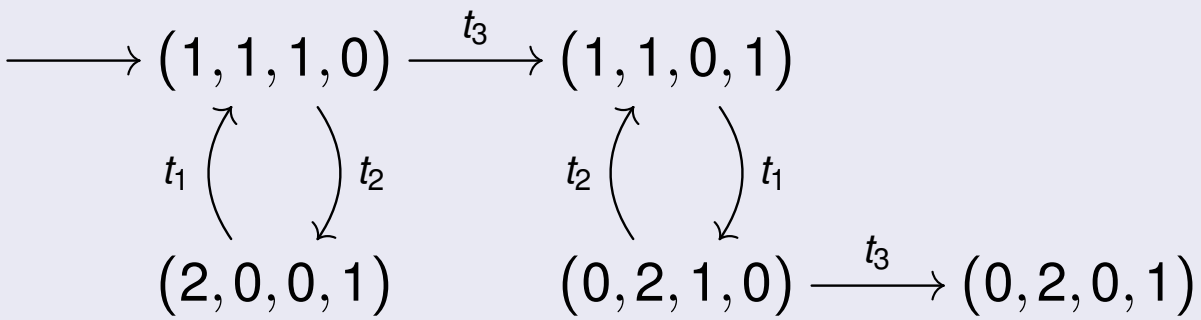


Petrinetze: Fallstudien (Kapazitätsbegrenzung)

Umgewandeltes Beispielnetz



Erreichbarkeitsgraph



Stellenordnung: $s_1, \overline{s_1}, s_2, \overline{s_2}$

Ausblick: Weitere Arten von Petrinetzen

Neben den soeben konzeptionell betrachteten Petrinetzen mit Kapazitäten gibt es noch weitere Arten/Erweiterungen von Petrinetzen.

Ausblick: Weitere Arten von Petrinetzen

Neben den soeben konzeptionell betrachteten Petrinetzen mit Kapazitäten gibt es noch weitere Arten/Erweiterungen von Petrinetzen.

Zum Beispiel Attributierte Petrinetze, auch spezieller, bekannt unter Namen wie:

- Petrinetze mit individuellen Marken
- Prädikat-Transitions-Petrinetze
- engl. coloured Petri nets

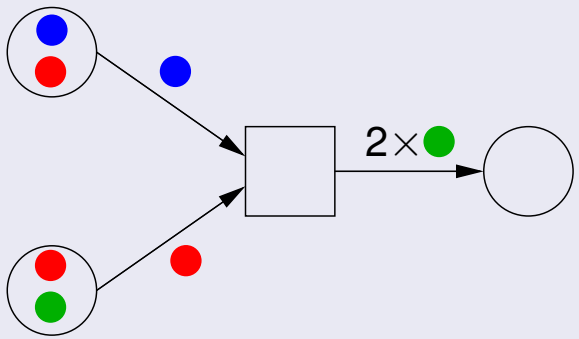
Ausblick: Weitere Arten von Petrinetzen

Dabei haben die Marken Farben. Die Transitionen geben an, Marken welcher Farbe entnommen und erzeugt werden sollen.

Ausblick: Weitere Arten von Petrinetzen

Dabei haben die Marken Farben. Die Transitionen geben an, Marken welcher Farbe entnommen und erzeugt werden sollen.

Beispielsweise entnimmt folgende Transition eine blaue und eine rote Marke und erzeugt zwei grüne Marken.



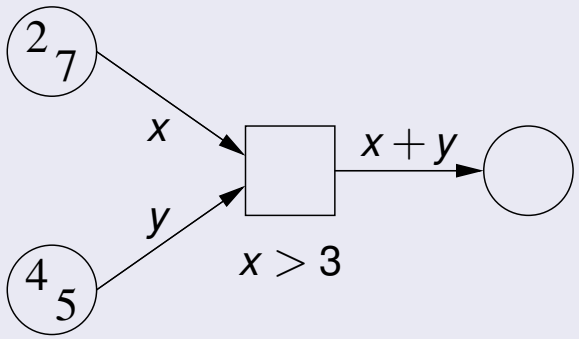
Ausblick: Weitere Arten von Petrinetzen

Die „Farben“ können dabei auch Elemente eines bestimmten Datentyps sein (z.B. Zahlen). Die Transitionen werden dann symbolisch annotiert und an ihnen können auch Bedingungen stehen.

Ausblick: Weitere Arten von Petrinetzen

Die „Farben“ können dabei auch Elemente eines bestimmten Datentyps sein (z.B. Zahlen). Die Transitionen werden dann symbolisch annotiert und an ihnen können auch Bedingungen stehen.

Beispielsweise hat folgendes Petrinetz natürliche Zahlen als „Farben“. Die Transition entnimmt der ersten Stelle eine Zahl x und der zweiten Stelle eine Zahl y. In die Stelle der Nachbedingung wird dann die Zahl $x + y$ gelegt. Die Transition darf nur schalten wenn $x > 3$.



von Petrinetzen zu UML

Bisher haben wir uns mit Zustandsübergangsdiagrammen und Petrinetzen (sowie deren Erreichbarkeitsgraphen und verwandten Konzepten) beschäftigt.

von Petrinetzen zu UML

Bisher haben wir uns mit Zustandsübergangsdiagrammen und Petrinetzen (sowie deren Erreichbarkeitsgraphen und verwandten Konzepten) beschäftigt.

Dabei handelte es sich um Modellierung von

- dynamischen Aspekten,

von Petrinetzen zu UML

Bisher haben wir uns mit Zustandsübergangsdiagrammen und Petrinetzen (sowie deren Erreichbarkeitsgraphen und verwandten Konzepten) beschäftigt.

Dabei handelte es sich um Modellierung von

- dynamischen Aspekten,
- in qualitativer und quantitativer Hinsicht,

von Petrinetzen zu UML

Bisher haben wir uns mit Zustandsübergangsdiagrammen und Petrinetzen (sowie deren Erreichbarkeitsgraphen und verwandten Konzepten) beschäftigt.

Dabei handelte es sich um Modellierung von

- dynamischen Aspekten,
- in qualitativer und quantitativer Hinsicht,
- einer breiten Klasse von Systemen,

von Petrinetzen zu UML

Bisher haben wir uns mit Zustandsübergangsdiagrammen und Petrinetzen (sowie deren Erreichbarkeitsgraphen und verwandten Konzepten) beschäftigt.

Dabei handelte es sich um Modellierung von

- dynamischen Aspekten,
- in qualitativer und quantitativer Hinsicht,
- einer breiten Klasse von Systemen,
- unter „white box“-Sicht,

von Petrinetzen zu UML

Bisher haben wir uns mit Zustandsübergangsdiagrammen und Petrinetzen (sowie deren Erreichbarkeitsgraphen und verwandten Konzepten) beschäftigt.

Dabei handelte es sich um Modellierung von

- dynamischen Aspekten,
- in qualitativer und quantitativer Hinsicht,
- einer breiten Klasse von Systemen,
- unter „white box“-Sicht,
- mit formaler Syntax und Semantik.

von Petrinetzen zu UML

Bisher haben wir uns mit Zustandsübergangsdiagrammen und Petrinetzen (sowie deren Erreichbarkeitsgraphen und verwandten Konzepten) beschäftigt.

Dabei handelte es sich um Modellierung von

- dynamischen Aspekten,
- in qualitativer und quantitativer Hinsicht,
- einer breiten Klasse von Systemen,
- unter „white box“-Sicht,
- mit formaler Syntax und Semantik.

Bei UML handelt es sich um eine ganze Familie von Modellierungsmitteln, daher wird einiges anders, teils insbesondere allgemeiner.

von Petrinetzen zu UML

UML beinhaltet Mittel zur

- auch vor allem visuell-grafischer statt textuell-mathematischer Modellierung

von Petrinetzen zu UML

UML beinhaltet Mittel zur

- auch vor allem visuell-grafischer statt textuell-mathematischer Modellierung von
- sowohl statischen als auch dynamischen Aspekten,

von Petrinetzen zu UML

UML beinhaltet Mittel zur

- auch vor allem visuell-grafischer statt textuell-mathematischer Modellierung von
- sowohl statischen als auch dynamischen Aspekten,
- in qualitativer und quantitativer Hinsicht,

von Petrinetzen zu UML

UML beinhaltet Mittel zur

- auch vor allem visuell-grafischer statt textuell-mathematischer Modellierung von
- sowohl statischen als auch dynamischen Aspekten,
- in qualitativer und quantitativer Hinsicht,
- von vor allem (objekt-orientierten) Software-Systemen,

von Petrinetzen zu UML

UML beinhaltet Mittel zur

- auch vor allem visuell-grafischer statt textuell-mathematischer Modellierung von
- sowohl statischen als auch dynamischen Aspekten,
- in qualitativer und quantitativer Hinsicht,
- von vor allem (objekt-orientierten) Software-Systemen,
- unter „white box“- oder „black box“-Sicht,

von Petrinetzen zu UML

UML beinhaltet Mittel zur

- auch vor allem visuell-grafischer statt textuell-mathematischer Modellierung von
- sowohl statischen als auch dynamischen Aspekten,
- in qualitativer und quantitativer Hinsicht,
- von vor allem (objekt-orientierten) Software-Systemen,
- unter „white box“- oder „black box“-Sicht,
- mit bestenfalls semi-formaler (Syntax und) Semantik,

von Petrinetzen zu UML

UML beinhaltet Mittel zur

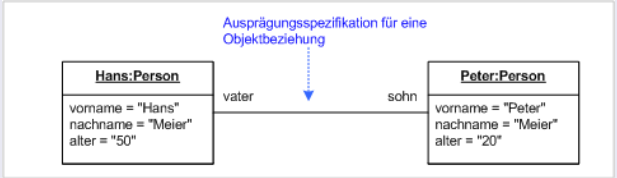
- auch vor allem visuell-grafischer statt textuell-mathematischer Modellierung von
- sowohl statischen als auch dynamischen Aspekten,
- in qualitativer und quantitativer Hinsicht,
- von vor allem (objekt-orientierten) Software-Systemen,
- unter „white box“- oder „black box“-Sicht,
- mit bestenfalls semi-formaler (Syntax und) Semantik,
- zur Verwendung bei Softwareentwicklung „im Großen“ mit strukturierten Entwicklungsprozessen.

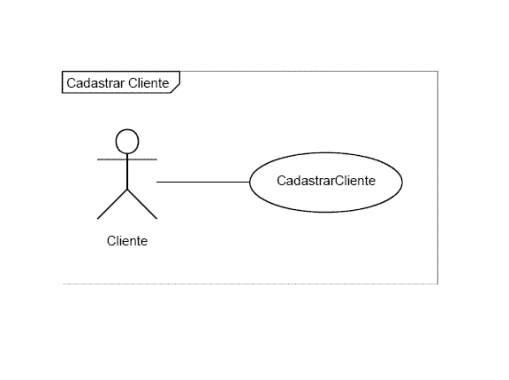
UML: Einführung und Objekt-Orientierung

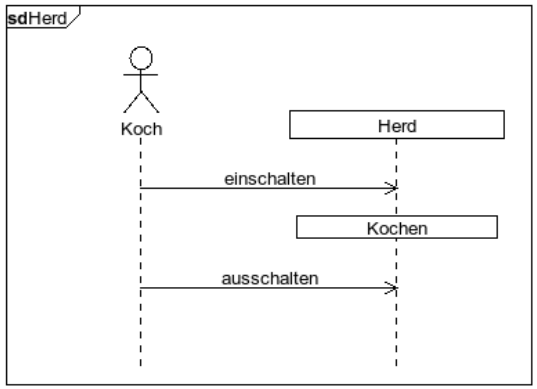
UML: Unified Modeling Language

UML = Unified Modeling Language

- Standard-Modellierungssprache für Software Engineering.
- Basiert auf objekt-orientierten Konzepten.
- Sehr umfangreich, enthält viele verschiedene Typen von Modellen.
- Entwickelt von Grady Booch, James Rumbaugh, Ivar Jacobson (1997).







UML: Einführung I

Einsatzgebiete von UML im Software-Engineering

- Visualisierung
- Spezifikation
- Konstruktion (z.B. zur Codegenerierung)
- Dokumentation

UML: Einführung II

Vokabular der UML (nach Booch/Rumbaugh/Jacobson):

Dinge

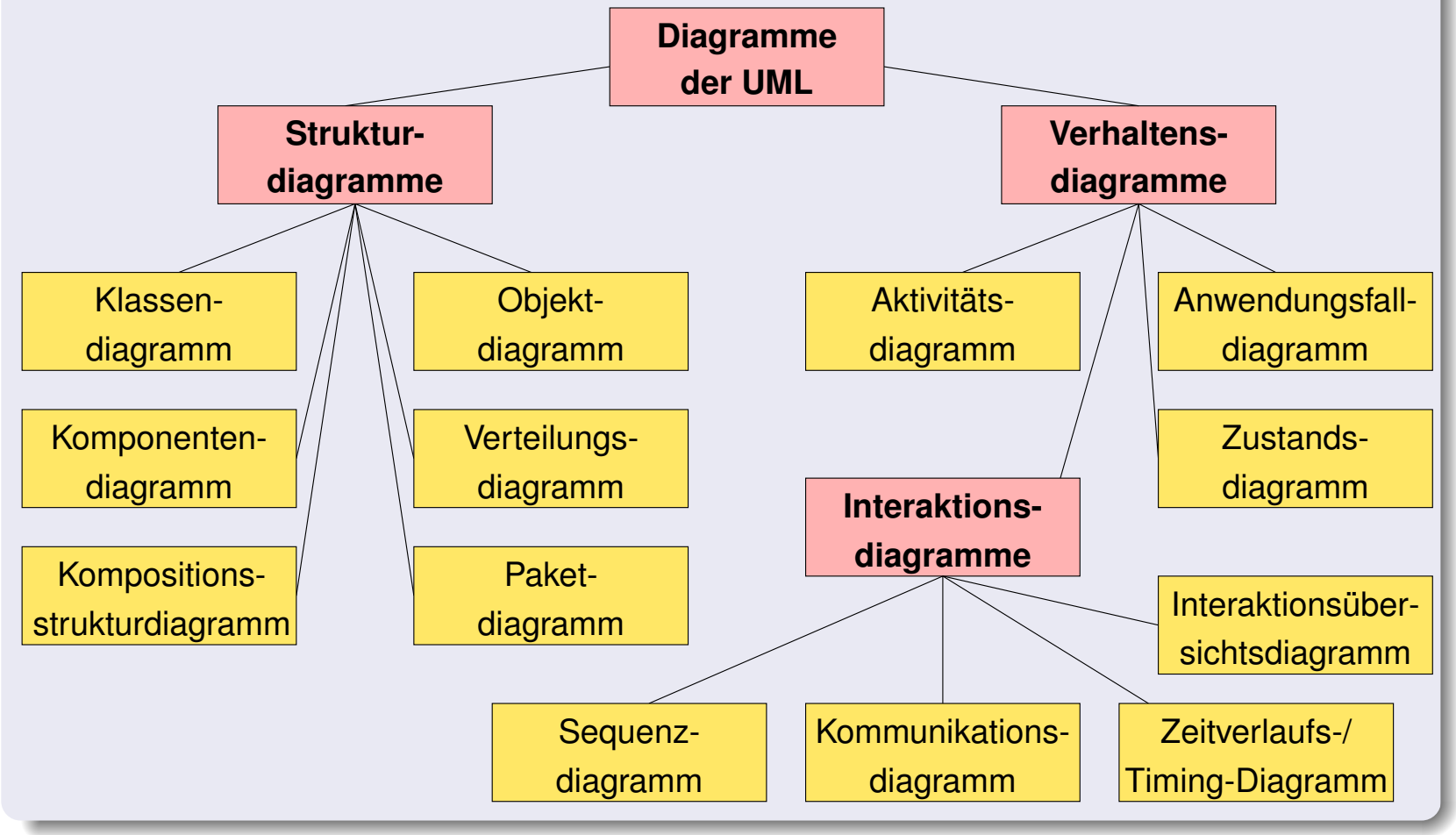
- Strukturen (structural things)
- Verhalten (behavioral things)
- Gruppen (grouping things)
- Annotationen (annotational things)

Beziehungen (relationships)

- Abhängigkeiten (dependencies)
- Assoziationen (associations)
- Generalisierungen (generalizations)
- Realisierungen (realizations)

UML: Einführung III

Diagramme



Wir werden im Folgenden einige dieser Begriffe mit Leben füllen.

UML und Objekt-Orientierung

Grundidee der Objekt-Orientierung

Vereinfacht besteht die Welt aus Objekten, die untereinander in Beziehungen stehen. Diese Sichtweise wird auch auf Modellierung und Softwareentwicklung übertragen.

UML und Objekt-Orientierung

Grundidee der Objekt-Orientierung

Vereinfacht besteht die Welt aus Objekten, die untereinander in Beziehungen stehen. Diese Sichtweise wird auch auf Modellierung und Softwareentwicklung übertragen.

Etwas genauer . . .

Daten (= Attribute) werden zusammen mit der Funktionalität (= Methoden) in Objekten organisiert bzw. gekapselt.

UML und Objekt-Orientierung

Grundidee der Objekt-Orientierung

Vereinfacht besteht die Welt aus Objekten, die untereinander in Beziehungen stehen. Diese Sichtweise wird auch auf Modellierung und Softwareentwicklung übertragen.

Etwas genauer . . .

Daten (= Attribute) werden zusammen mit der Funktionalität (= Methoden) in Objekten organisiert bzw. gekapselt.

Jedes Objekt ist in der Lage, Nachrichten (= Methodenaufrufe) zu empfangen, Daten zu verarbeiten und Nachrichten zu senden.

UML und Objekt-Orientierung

Grundidee der Objekt-Orientierung

Vereinfacht besteht die Welt aus Objekten, die untereinander in Beziehungen stehen. Diese Sichtweise wird auch auf Modellierung und Softwareentwicklung übertragen.

Etwas genauer . . .

Daten (= Attribute) werden zusammen mit der Funktionalität (= Methoden) in Objekten organisiert bzw. gekapselt.

Jedes Objekt ist in der Lage, Nachrichten (= Methodenaufrufe) zu empfangen, Daten zu verarbeiten und Nachrichten zu senden.

Diese Objekte, bzw. die Objekttypen, können – einmal realisiert – in verschiedenen Kontexten wiederverwendet werden.

UML und Objekt-Orientierung

Geschichte der Objekt-Orientierung

- Entwicklung von objekt-orientierten Programmiersprachen:
  - 60er Jahre: Simula (zur Beschreibung und Simulation von komplexen Mensch-Maschine-Interaktionen)
  - 80er Jahre: C++
  - 90er Jahre: Java
- Verbreitung von objekt-orientierten Entwurfsmethoden:
  - 70er Jahre: Entity-Relationship-Modell
  - 90er Jahre: Vorläufer von UML:
    OOSE (Object-Oriented Software Engineering),
    OMT (Object Modeling Technique)
  - Seit 1997: UML
  - Seit 2005: UML 2.0

UML und Objekt-Orientierung

Behauptete Vorteile der objekt-orientierten Modellierung und Programmierung:

- Leichte Wiederverwendbarkeit dadurch, dass Daten und Funktionalität zusammen verwaltet werden und es Konzepte zur Modifikation von Verhalten gibt (Stichwort: Vererbung).

UML und Objekt-Orientierung

Behauptete Vorteile der objekt-orientierten Modellierung und Programmierung:

- Leichte Wiederverwendbarkeit dadurch, dass Daten und Funktionalität zusammen verwaltet werden und es Konzepte zur Modifikation von Verhalten gibt (Stichwort: Vererbung).
- Verträglichkeit mit Nebenläufigkeit und Parallelität: Kontrollfluss kann nebenläufig in verschiedenen Objekten ablaufen und diese können durch Nachrichtenaustausch bzw. Methodenaufrufe miteinander kommunizieren.

UML und Objekt-Orientierung

Behauptete Vorteile der objekt-orientierten Modellierung und Programmierung:

- Leichte Wiederverwendbarkeit dadurch, dass Daten und Funktionalität zusammen verwaltet werden und es Konzepte zur Modifikation von Verhalten gibt (Stichwort: Vererbung).
- Verträglichkeit mit Nebenläufigkeit und Parallelität: Kontrollfluss kann nebenläufig in verschiedenen Objekten ablaufen und diese können durch Nachrichtenaustausch bzw. Methodenaufrufe miteinander kommunizieren.
- Nähe zur realen Welt: viele Dinge der realen Welt können als Objekte modelliert werden.

UML und Objekt-Orientierung

Ein Beispiel für die Modellierung von Objekten der realen Welt:

Fahrkartenautomat

- Daten: Fahrziele, Zoneneinteilung, Fahrtkosten
- Funktionalität: Tasten drücken, Preise anzeigen, Münzen einwerfen, Fahrkarten auswerfen

UML und Objekt-Orientierung

Konzepte

- Klasse: definiert einen Typ von Objekten mit bestimmten Arten von Daten und bestimmter Funktionalität.

  Beispiel: die Klasse der VRR-Fahrkartenautomaten

UML und Objekt-Orientierung

Konzepte

- Klasse: definiert einen Typ von Objekten mit bestimmten Arten von Daten und bestimmter Funktionalität.

  Beispiel: die Klasse der VRR-Fahrkartenautomaten

- Objekt: eine Instanz einer Klasse

  Beispiel: der Fahrkartenautomat am Duisburger Hauptbahnhof, Osteingang

Klassen- und Objektdiagramme

Klassen- und Objektdiagramme

Wir beginnen mit Klassen- und Objektdiagrammen.

Dabei geht es um statische Modellierung:

- Dinge, ihre Eigenschaften, und Beziehungen zwischen ihnen;

Klassen- und Objektdiagramme

Wir beginnen mit Klassen- und Objektdiagrammen.

Dabei geht es um statische Modellierung:

- Dinge, ihre Eigenschaften, und Beziehungen zwischen ihnen;
- wie sich der Zustand eines Systems jeweils zusammensetzt, nicht wie/wohin er sich entwickeln kann.

Klassen- und Objektdiagramme

Wir beginnen mit Klassen- und Objektdiagrammen.

Dabei geht es um statische Modellierung:

- Dinge, ihre Eigenschaften, und Beziehungen zwischen ihnen;
- wie sich der Zustand eines Systems jeweils zusammensetzt, nicht wie/wohin er sich entwickeln kann.

Das klingt weniger spannend als Modellierung des dynamischen Verhaltens eines Systems (etwa mittels Petrinetzen)

Klassen- und Objektdiagramme

Wir beginnen mit Klassen- und Objektdiagrammen.

Dabei geht es um statische Modellierung:

- Dinge, ihre Eigenschaften, und Beziehungen zwischen ihnen;
- wie sich der Zustand eines Systems jeweils zusammensetzt, nicht wie/wohin er sich entwickeln kann.

Das klingt weniger spannend als Modellierung des dynamischen Verhaltens eines Systems (etwa mittels Petrinetzen), aber:

- präzise statische Modellierung ist wichtige Hilfe für Implementierung größerer Software-Systeme

Klassen- und Objektdiagramme

Wir beginnen mit Klassen- und Objektdiagrammen.

Dabei geht es um statische Modellierung:

- Dinge, ihre Eigenschaften, und Beziehungen zwischen ihnen;
- wie sich der Zustand eines Systems jeweils zusammensetzt, nicht wie/wohin er sich entwickeln kann.

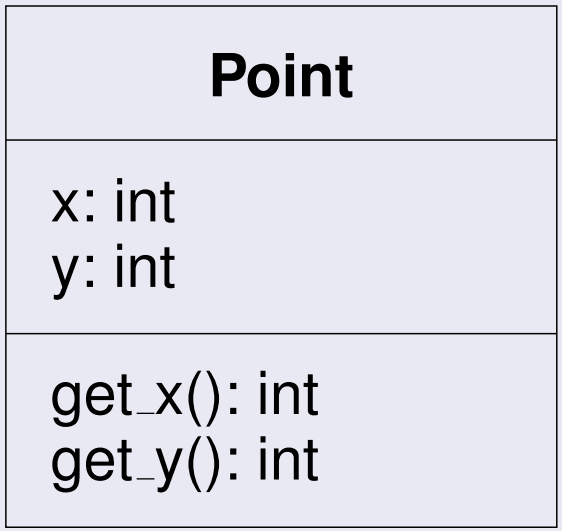
Das klingt weniger spannend als Modellierung des dynamischen Verhaltens eines Systems (etwa mittels Petrinetzen), aber:

- präzise statische Modellierung ist wichtige Hilfe für Implementierung größerer Software-Systeme,
- und erlaubt die Anwendung von anerkannten (aus Erfahrung gewonnenen) Design-Prinzipien, etwa zum angemessenen Einsatz von Vererbung.

Klassen- und Objektdiagramme

Beispiel: Klasse von Punkten mit x-, y-Koordinaten (also zweidimensional) und Operationen zum Auslesen der Koordinaten

Grafische Darstellung einer Klasse



Klassenname

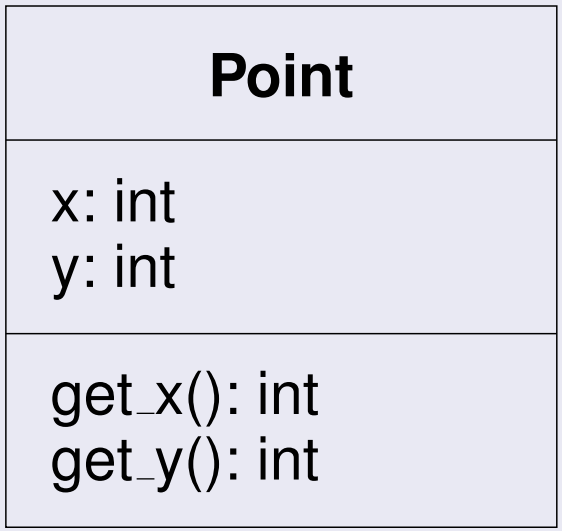
Attribute (evtl. mit Typ)

Operationen/Methoden

Klassen- und Objektdiagramme

Beispiel: Klasse von Punkten mit x-, y-Koordinaten (also zweidimensional) und Operationen zum Auslesen der Koordinaten

Grafische Darstellung einer Klasse



Klassenname

Attribute (evtl. mit Typ)

Operationen/Methoden

Bemerkung: Obwohl ja auch Aktivitäten aufgeführt werden (etwa get_x), handelt es sich dennoch um statische Modellierung. (Warum?)

Attribute & Operationen

Weitere Bemerkungen:

- Bei den Attributen handelt es sich um sogenannte Instanzattribute, das heißt, sie gehören zu den Instanzen einer Klasse (zu den Objekten, nicht zur Klasse selbst).

Attribute & Operationen

Weitere Bemerkungen:

- Bei den Attributen handelt es sich um sogenannte Instanzattribute, das heißt, sie gehören zu den Instanzen einer Klasse (zu den Objekten, nicht zur Klasse selbst).
- Man kann die Sichtbarkeit eines Attributes bzw. einer Methode spezifieren, indem man bestimmte Modifikatoren ($+$, $-$, $\#$, $\sim$) vor den Attribut-/Methodennamen schreibt.

Attribute & Operationen

Weitere Bemerkungen:

- Bei den Attributen handelt es sich um sogenannte Instanzattribute, das heißt, sie gehören zu den Instanzen einer Klasse (zu den Objekten, nicht zur Klasse selbst).
- Man kann die Sichtbarkeit eines Attributes bzw. einer Methode spezifieren, indem man bestimmte Modifikatoren ($+$, $-$, $\#$, $\sim$) vor den Attribut-/Methodennamen schreibt.
- Attribute haben im Allgemeinen Typen, manchmal auch Vorgabewerte (= initiale Werte). Dies wird dann folgendermaßen notiert: x: int = 0

Attribute & Operationen

Weitere Bemerkungen:

- Bei den Attributen handelt es sich um sogenannte Instanzattribute, das heißt, sie gehören zu den Instanzen einer Klasse (zu den Objekten, nicht zur Klasse selbst).
- Man kann die Sichtbarkeit eines Attributes bzw. einer Methode spezifieren, indem man bestimmte Modifikatoren ($+$, $-$, $\#$, $\sim$) vor den Attribut-/Methodennamen schreibt.
- Attribute haben im Allgemeinen Typen, manchmal auch Vorgabewerte (= initiale Werte). Dies wird dann folgendermaßen notiert: x: int = 0
- Operationen mit Argumenten (und Rückgabewert) werden mit ihren Typen folgendermaßen notiert: add(m: int, n: int): int

Instanziierung

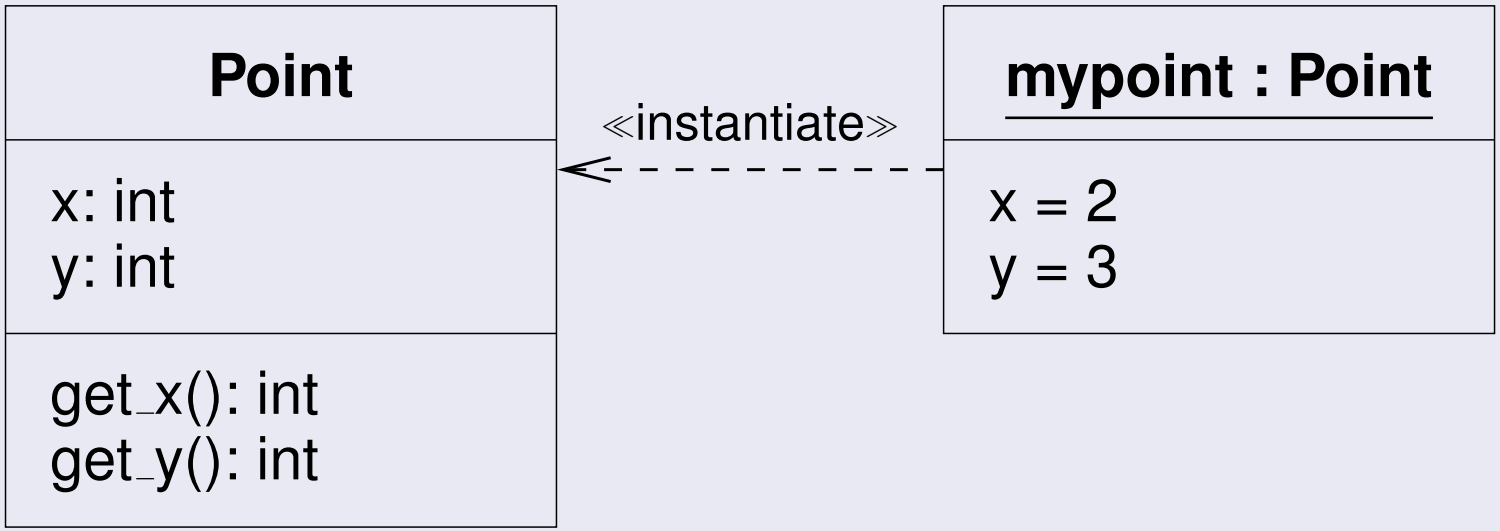
Eine Klasse beschreibt den allgemeinen Aufbau bestimmter Objekte.

Instanziierung

Eine Klasse beschreibt den allgemeinen Aufbau bestimmter Objekte.

Eine Instanz einer Klasse stellt ein konkretes Objekt mit den entsprechenden Werten für die Attribute dar.

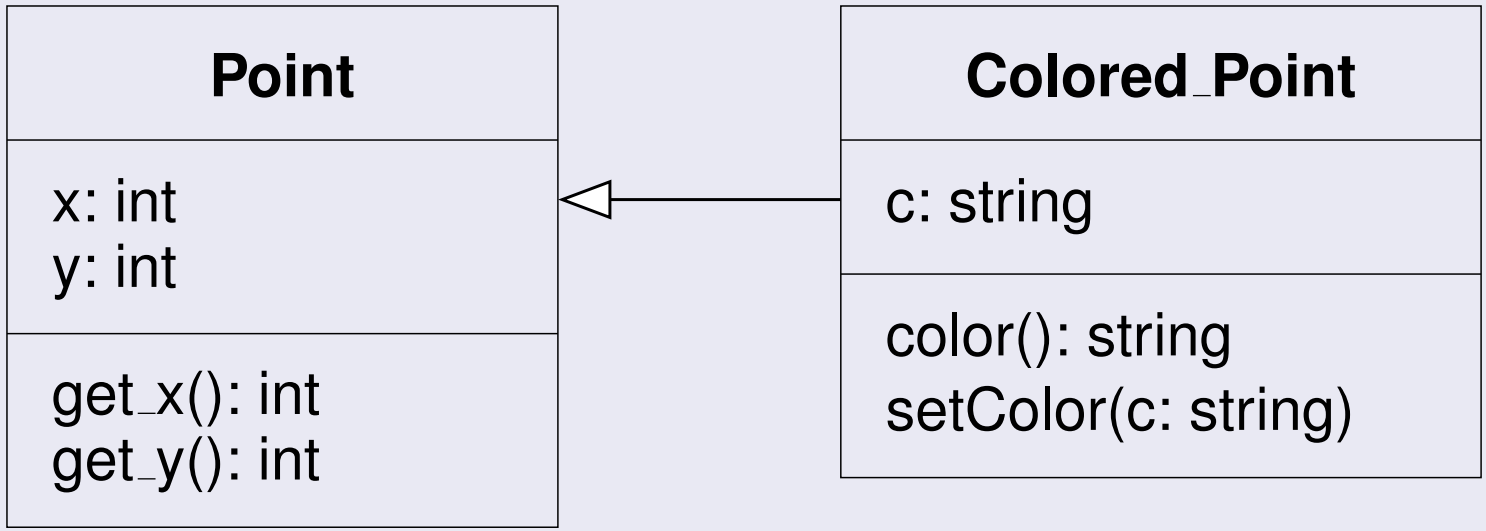
Grafische Darstellung einer Instanz (Objekt) einer Klasse



Generalisierung/Spezialisierung & Polymorphie

Es ist möglich, neue Klassen von bestehenden Klassen erben zu lassen. (Generalisierung/Spezialisierung)

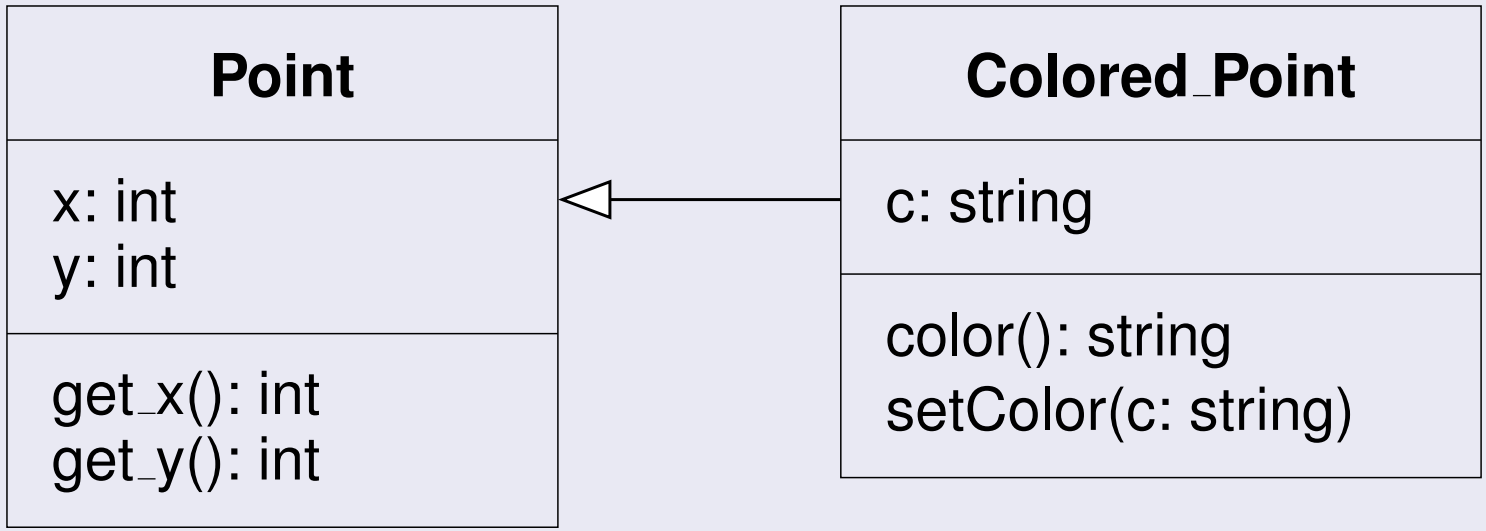
Grafische Darstellung von Vererbung



Generalisierung/Spezialisierung & Polymorphie

Es ist möglich, neue Klassen von bestehenden Klassen erben zu lassen. (Generalisierung/Spezialisierung)

Grafische Darstellung von Vererbung



In einer solchen Situation nennt man **Point** eine Superklasse bzw. Oberklasse und **Colored_Point** eine Subklasse bzw. Unterklasse.

Generalisierung/Spezialisierung & Polymorphie

Bemerkungen:

- Die Subklasse erbt die Attribute, Methoden und Assoziationen der Superklasse, und kann diesen noch weitere hinzufügen. Außerdem kann man in der Subklasse Methoden der Superklasse überschreiben, also durch neue ersetzen.

Generalisierung/Spezialisierung & Polymorphie

Bemerkungen:

- Die Subklasse erbt die Attribute, Methoden und Assoziationen der Superklasse, und kann diesen noch weitere hinzufügen. Außerdem kann man in der Subklasse Methoden der Superklasse überschreiben, also durch neue ersetzen.
- Ein Objekt einer Unterklasse kann auch als ein Objekt jeder seiner Oberklassen angesehen werden. $\Rightarrow$ Polymorphie

Generalisierung/Spezialisierung & Polymorphie

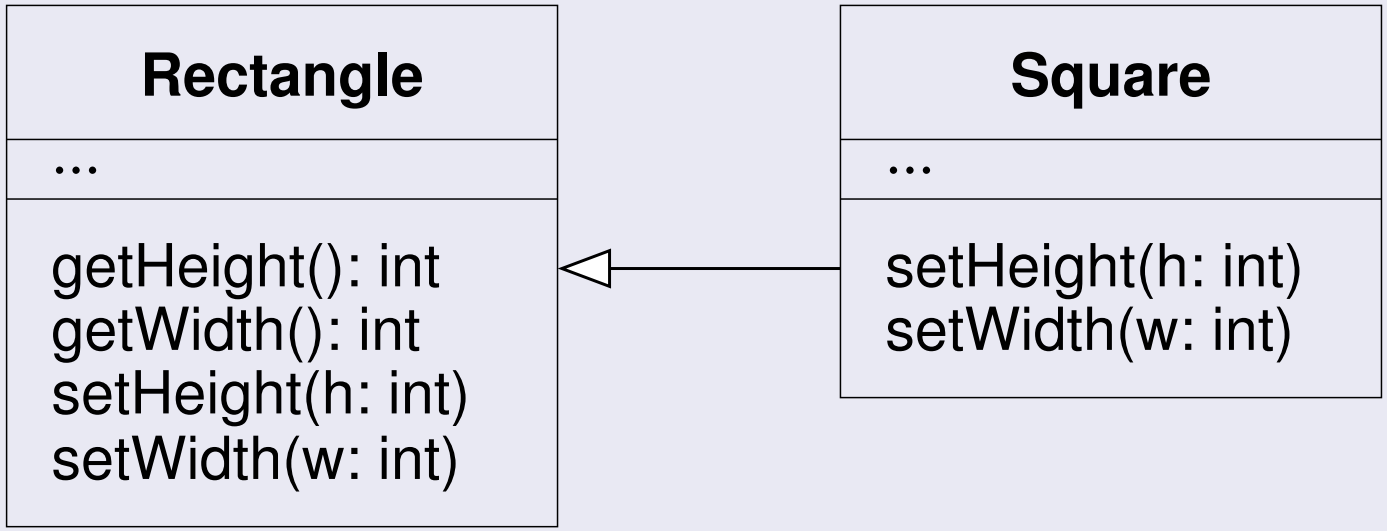
Bemerkungen:

- Die Subklasse erbt die Attribute, Methoden und Assoziationen der Superklasse, und kann diesen noch weitere hinzufügen. Außerdem kann man in der Subklasse Methoden der Superklasse überschreiben, also durch neue ersetzen.
- Ein Objekt einer Unterklasse kann auch als ein Objekt jeder seiner Oberklassen angesehen werden. $\Rightarrow$ Polymorphie
- Dabei sollte man darauf achten, dass sich das Verhalten eines Programms bei solch einer Substitution nicht ändert. (Liskovs Substitutions-Prinzip)

Generalisierung/Spezialisierung & Polymorphie

Beispiel für mögliche Verletzung des Substitutions-Prinzips

Scheinbar sinnvolle Vererbungsbeziehung:

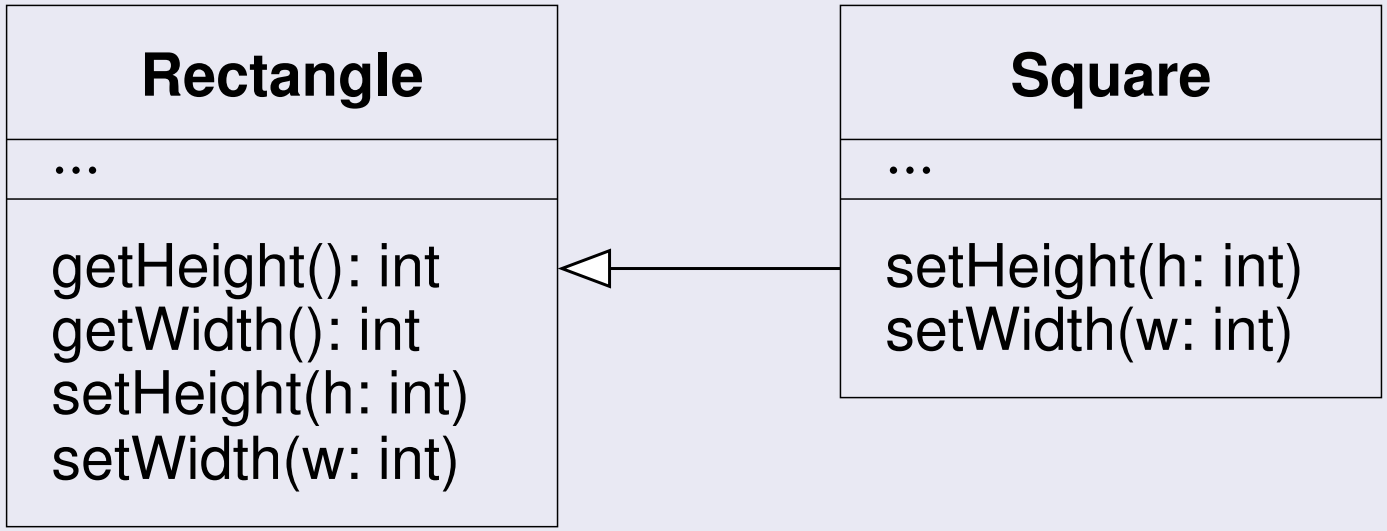


wobei die in der Unterklasse überschriebenen Methoden das Quadratisch-Sein erzwingen.

Generalisierung/Spezialisierung & Polymorphie

Beispiel für mögliche Verletzung des Substitutions-Prinzips

Scheinbar sinnvolle Vererbungsbeziehung:



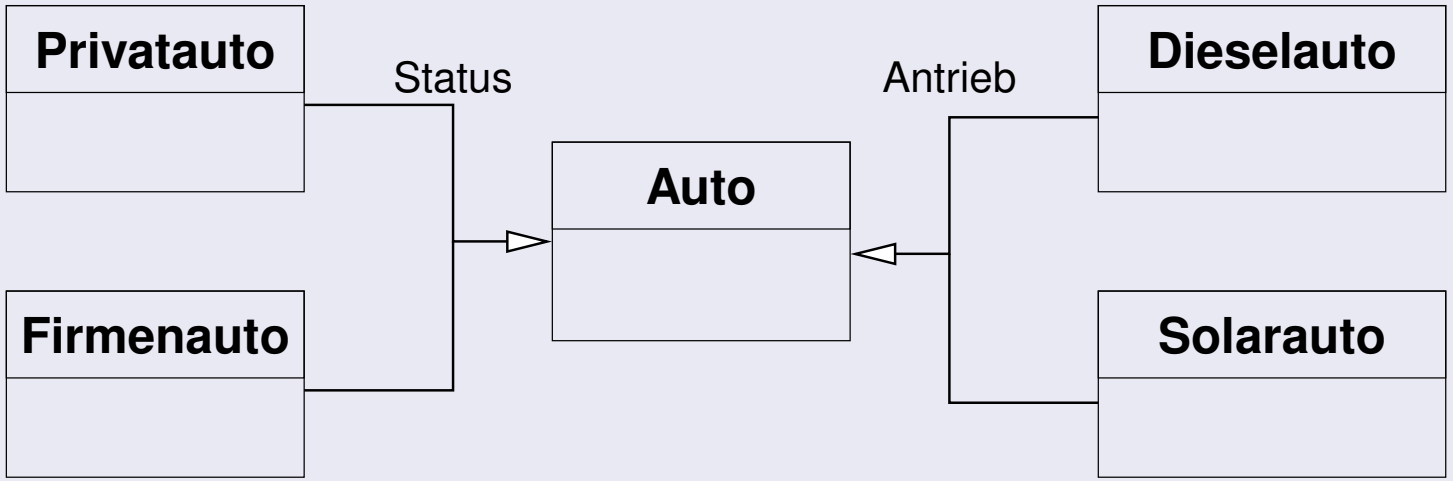
wobei die in der Unterklasse überschriebenen Methoden das Quadratisch-Sein erzwingen.

Was passiert, wenn wir in dem für **Rectangle**s gedachten (und getypten) Code: o.setHeight(10); o.setWidth(20); print(o.getHeight()); ein **Square**-Objekt einsetzen?

Generalisierung/Spezialisierung & Polymorphie

Generalisierungsgruppen

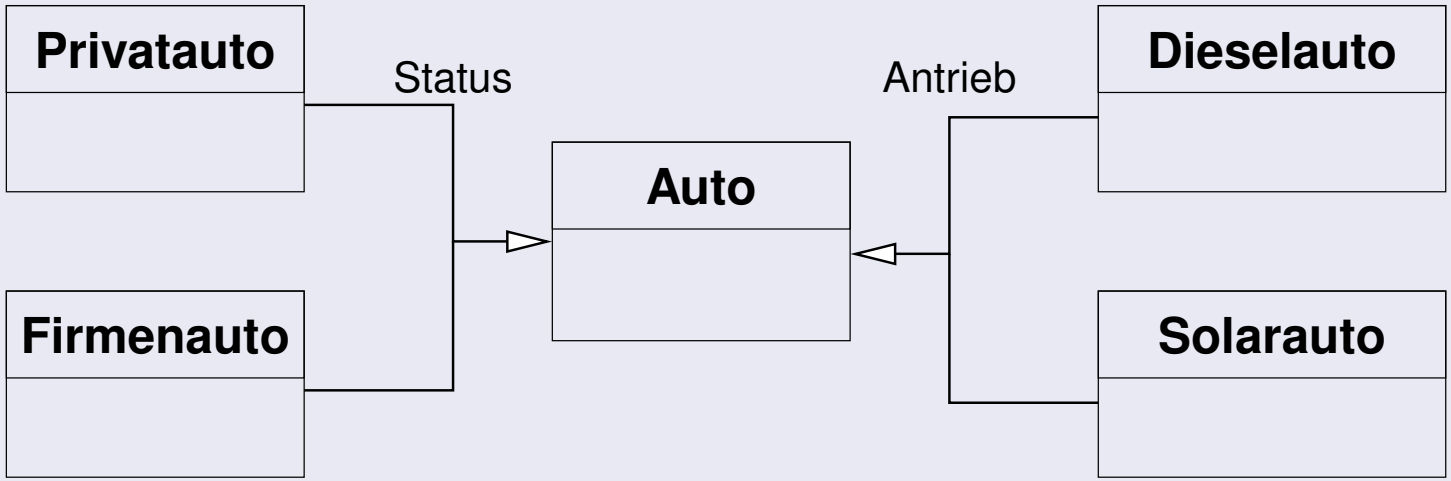
Mitunter können Klassen in unterschiedlicher Weise spezialisiert bzw. unterteilt werden. Einzelne Generalisierungsbeziehungen können dann zu Gruppen zusammengefasst werden.



Generalisierung/Spezialisierung & Polymorphie

Generalisierungsgruppen

Mitunter können Klassen in unterschiedlicher Weise spezialisiert bzw. unterteilt werden. Einzelne Generalisierungsbeziehungen können dann zu Gruppen zusammengefasst werden.



Dabei wird die jeweilige Generalisierungsgruppe (hier: Status bzw. Antrieb) im Diagramm annotiert.

Generalisierung/Spezialisierung & Polymorphie

Den Generalisierungsgruppen können Eigenschaften (in geschweiften Klammern) zugeordnet werden.

Generalisierung/Spezialisierung & Polymorphie

Den Generalisierungsgruppen können Eigenschaften (in geschweiften Klammern) zugeordnet werden.

- complete/incomplete:
  - complete: die Generalisierungsgruppe ist vollständig, das heißt, sie überdeckt konzeptionell alle denkbaren Instanzen der Oberklasse.

Generalisierung/Spezialisierung & Polymorphie

Den Generalisierungsgruppen können Eigenschaften (in geschweiften Klammern) zugeordnet werden.

- complete/incomplete:
  - complete: die Generalisierungsgruppe ist vollständig, das heißt, sie überdeckt konzeptionell alle denkbaren Instanzen der Oberklasse.
  - incomplete: die Generalisierungsgruppe ist unvollständig, das heißt, es gibt durch sie nicht erfasste Instanzen.

Generalisierung/Spezialisierung & Polymorphie

Den Generalisierungsgruppen können Eigenschaften (in geschweiften Klammern) zugeordnet werden.

- complete/incomplete:
  - complete: die Generalisierungsgruppe ist vollständig, das heißt, sie überdeckt konzeptionell alle denkbaren Instanzen der Oberklasse.
  - incomplete: die Generalisierungsgruppe ist unvollständig, das heißt, es gibt durch sie nicht erfasste Instanzen.
- overlapping/disjoint:
  - overlapping: es sind Instanzen denkbar, die konzeptionell zu mehr als einer der spezialisierenden Klassen gehören könnten.

Generalisierung/Spezialisierung & Polymorphie

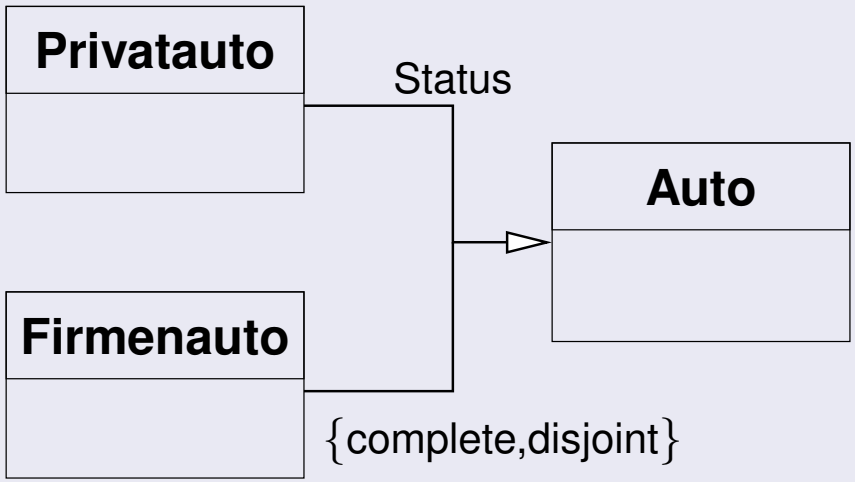
Den Generalisierungsgruppen können Eigenschaften (in geschweiften Klammern) zugeordnet werden.

- complete/incomplete:
  - complete: die Generalisierungsgruppe ist vollständig, das heißt, sie überdeckt konzeptionell alle denkbaren Instanzen der Oberklasse.
  - incomplete: die Generalisierungsgruppe ist unvollständig, das heißt, es gibt durch sie nicht erfasste Instanzen.
- overlapping/disjoint:
  - overlapping: es sind Instanzen denkbar, die konzeptionell zu mehr als einer der spezialisierenden Klassen gehören könnten.
  - disjoint: die spezialisierenden Klassen überlappen sich konzeptionell nicht.

Generalisierung/Spezialisierung & Polymorphie

Beispiele für die Eigenschaften complete/incomplete und disjoint/overlapping:

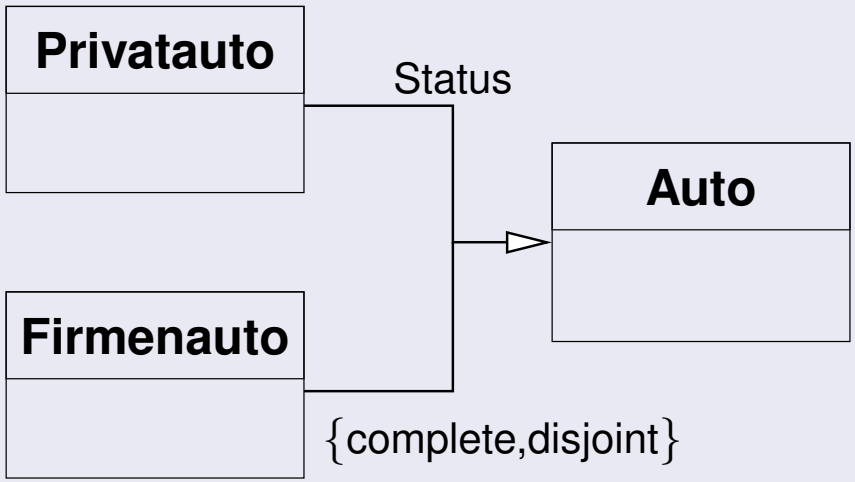
{complete,disjoint}



Generalisierung/Spezialisierung & Polymorphie

Beispiele für die Eigenschaften complete/incomplete und disjoint/overlapping:

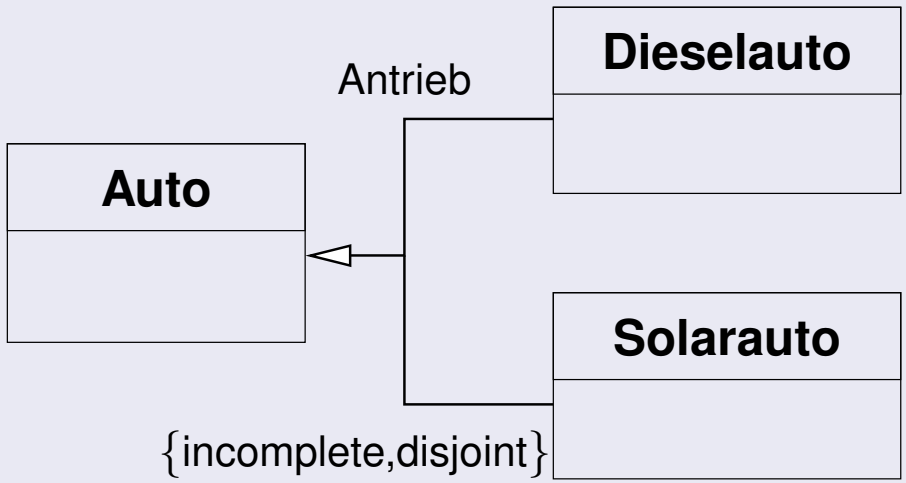
{complete,disjoint}



Hier handelt es sich um eine konzeptionelle Partitionierung der Instanzen der Oberklasse.

Generalisierung/Spezialisierung & Polymorphie

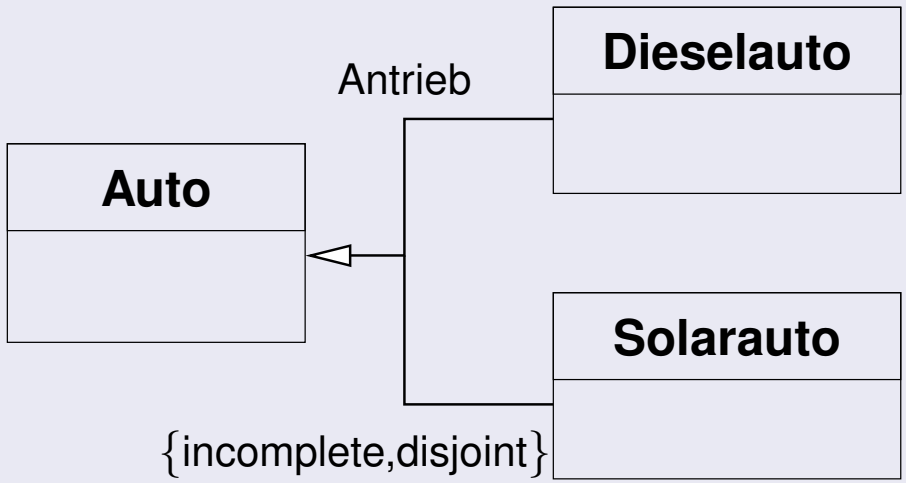
{incomplete,disjoint}



Warum unvollständig?

Generalisierung/Spezialisierung & Polymorphie

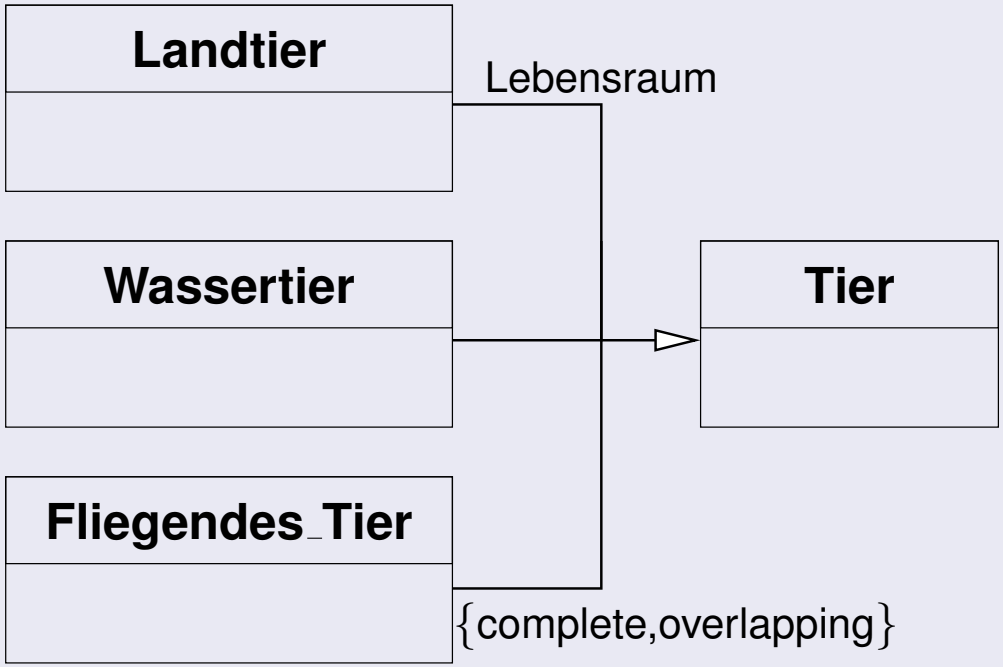
{incomplete,disjoint}



Warum unvollständig? $\rightsquigarrow$ Es fehlt z.B. eine Klasse Benzinauto.

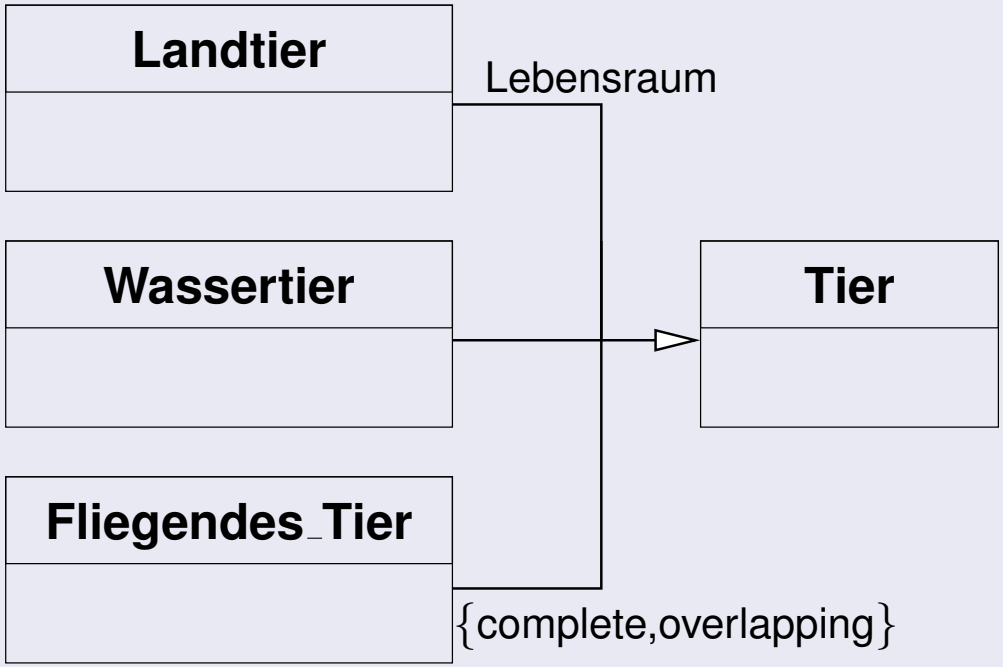
Generalisierung/Spezialisierung & Polymorphie

{complete,overlapping}



Generalisierung/Spezialisierung & Polymorphie

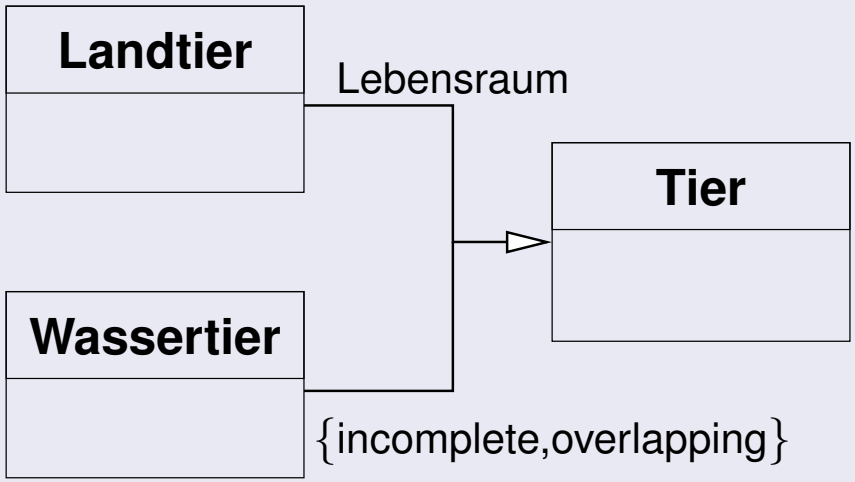
{complete,overlapping}



Schildkröten sind sowohl Land- als auch Wassertiere.

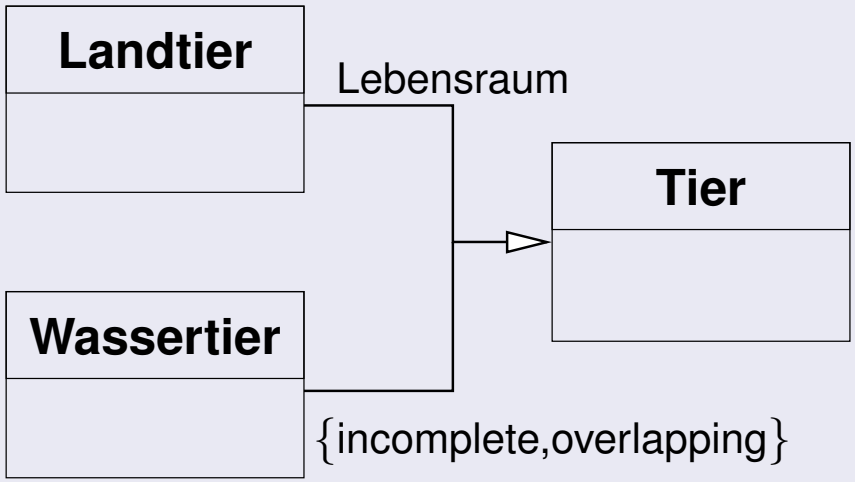
Generalisierung/Spezialisierung & Polymorphie

{incomplete,overlapping}



Generalisierung/Spezialisierung & Polymorphie

{incomplete,overlapping}



Fliegende Tiere fehlen.

Beziehungen zwischen Klassen

Neben Vererbung (also Generalisierung/Spezialisierung) sind noch andere Beziehungen zwischen Klassen (und letztlich zwischen deren Objekten) in UML darstellbar.

Beziehungen zwischen Klassen

Neben Vererbung (also Generalisierung/Spezialisierung) sind noch andere Beziehungen zwischen Klassen (und letztlich zwischen deren Objekten) in UML darstellbar.

Wir betrachten folgende Arten von Beziehungen:

- Assoziation
- Aggregation
- Komposition

Beziehungen zwischen Klassen

Neben Vererbung (also Generalisierung/Spezialisierung) sind noch andere Beziehungen zwischen Klassen (und letztlich zwischen deren Objekten) in UML darstellbar.

Wir betrachten folgende Arten von Beziehungen:

- Assoziation
- Aggregation
- Komposition

Die Art der Beziehung drückt die jeweilige Stärke der Verbindung aus.

Komposition *ist_stärker_als* Aggregation, und
Aggregation *ist_stärker_als* Assoziation.

Beziehungen zwischen Klassen: Assoziation

Wir beginnen mit der schwächsten Beziehung: der Assoziation.

Assoziation

Es gibt eine Assoziation zwischen den Klassen **A** und **B**, wenn

> es irgendeinen semantischen, nicht-hierarchischen Zusammenhang zwischen den Klassen gibt, der sinnvoll benannt werden kann.

Beziehungen zwischen Klassen: Assoziation

Wir beginnen mit der schwächsten Beziehung: der Assoziation.

Assoziation

Es gibt eine Assoziation zwischen den Klassen **A** und **B**, wenn

> es irgendeinen semantischen, nicht-hierarchischen Zusammenhang zwischen den Klassen gibt, der sinnvoll benannt werden kann.

Letztlich drückt eine Assoziation nur aus, dass Objekte der einen Klasse für zumindest einen Teil ihrer Funktionalität eine persistente (abzuspeichernde) Verbindung zu bestimmten Objekten der anderen Klasse brauchen.

Beziehungen zwischen Klassen: Assoziation

Wir beginnen mit der schwächsten Beziehung: der Assoziation.

Assoziation

Es gibt eine Assoziation zwischen den Klassen **A** und **B**, wenn

> es irgendeinen semantischen, nicht-hierarchischen Zusammenhang zwischen den Klassen gibt, der sinnvoll benannt werden kann.

Letztlich drückt eine Assoziation nur aus, dass Objekte der einen Klasse für zumindest einen Teil ihrer Funktionalität eine persistente (abzuspeichernde) Verbindung zu bestimmten Objekten der anderen Klasse brauchen.

Üblicherweise werden Assoziationen durch Referenzen realisiert. Das heißt, eine der Klassen hat ein Attribut vom Typ der anderen Klasse. Diese Attribute werden im Klassendiagramm jedoch nicht explizit angegeben.

Beziehungen zwischen Klassen: Assoziation

Beispiel für eine Assoziation

Eine Person kann ein Auto besitzen.

Beziehungen zwischen Klassen: Assoziation

Beispiel für eine Assoziation

Eine Person kann ein Auto besitzen.

Obige Angabe schließt bewusst nicht aus, dass eine Person auch mehrere, oder gar kein Auto, besitzen könnte. Oder dass ein Auto von mehreren Personen besessen werden könnte.

Beziehungen zwischen Klassen: Assoziation

Beispiel für eine Assoziation

Eine Person kann ein Auto besitzen.

Obige Angabe schließt bewusst nicht aus, dass eine Person auch mehrere, oder gar kein Auto, besitzen könnte. Oder dass ein Auto von mehreren Personen besessen werden könnte.

Mögliche Ausprägungen auf der Ebene von Objekten wären also etwa:

1. $\{(person_1, auto_1), (person_2, auto_2), (person_3, auto_3)\}$
2. $\{(person_1, auto_1), (person_1, auto_2), (person_3, auto_3)\}$
3. $\{(person_1, auto_1), (person_1, auto_2), (person_2, auto_2)\}$

Beziehungen zwischen Klassen: Assoziation

Oft wird eine Leserichtung der Assoziation eingeführt:

Beziehungen zwischen Klassen: Assoziation

Oft wird eine Leserichtung der Assoziation eingeführt:

Separat kann eine Navigationsrichtung eingeführt werden, die beschreibt, welche Klasse ihren Assoziationspartner kennt (und daher seine Methoden aufrufen kann):

Beziehungen zwischen Klassen: Assoziation

Oft wird eine Leserichtung der Assoziation eingeführt:

Separat kann eine Navigationsrichtung eingeführt werden, die beschreibt, welche Klasse ihren Assoziationspartner kennt (und daher seine Methoden aufrufen kann):

Hier hätten also **Person**-Objekte Referenzen auf **Auto**-Objekte.

Beziehungen zwischen Klassen: Assoziation

Oft wird eine Leserichtung der Assoziation eingeführt:

Separat kann eine Navigationsrichtung eingeführt werden, die beschreibt, welche Klasse ihren Assoziationspartner kennt (und daher seine Methoden aufrufen kann):

Hier hätten also **Person**-Objekte Referenzen auf **Auto**-Objekte. Mengentheoretisch ausgedrückt, zum Beispiel:

2. $person_1 \mapsto \{auto_1, auto_2\}$, $person_2 \mapsto \emptyset$, $person_3 \mapsto \{auto_3\}$

Beziehungen zwischen Klassen: Assoziation

Die Navigationsrichtung kann im Prinzip von der Leserichtung abweichen:

Beziehungen zwischen Klassen: Assoziation

Die Navigationsrichtung kann im Prinzip von der Leserichtung abweichen:

Allerdings ist das ungewöhnlich und deutet auf einen Modellierungsfehler hin.

Beziehungen zwischen Klassen: Assoziation

Die Navigationsrichtung kann im Prinzip von der Leserichtung abweichen:

Allerdings ist das ungewöhnlich und deutet auf einen Modellierungsfehler hin.

Hier kennen sich Vertreter beider Klassen gegenseitig:

Beziehungen zwischen Klassen: Assoziation

An beiden Enden einer Assoziation können Multiplizitäten in Form von Intervallen *m..n* (oder einfach nur *m* für *m..m*) angegeben werden.

Beziehungen zwischen Klassen: Assoziation

An beiden Enden einer Assoziation können Multiplizitäten in Form von Intervallen *m..n* (oder einfach nur *m* für *m..m*) angegeben werden.

Hier besitzt jede Person bis zu fünf Autos. Und jedes Auto ist im Besitz genau einer Person.

Beziehungen zwischen Klassen: Assoziation

An beiden Enden einer Assoziation können Multiplizitäten in Form von Intervallen *m..n* (oder einfach nur *m* für *m..m*) angegeben werden.

Hier besitzt jede Person bis zu fünf Autos. Und jedes Auto ist im Besitz genau einer Person.

Falls die Multiplizität (am anderen Ende einer navigierbaren Assoziation) größer als Eins möglich ist, muss dies in der Implementierung durch eine Kollektion (Liste, Menge, Array) von Referenzen realisiert werden.

Beziehungen zwischen Klassen: Assoziation

Was bedeuten zum Beispiel folgende Multiplizitäten?

Beziehungen zwischen Klassen: Assoziation

Was bedeuten zum Beispiel folgende Multiplizitäten?

Jedenfalls nicht, dass immer zwei Personen zusammen genau die gleichen Autos besitzen müssen.

Beziehungen zwischen Klassen: Assoziation

Was bedeuten zum Beispiel folgende Multiplizitäten?

Jedenfalls nicht, dass immer zwei Personen zusammen genau die gleichen Autos besitzen müssen.

Möglich wäre etwa folgende Situation:

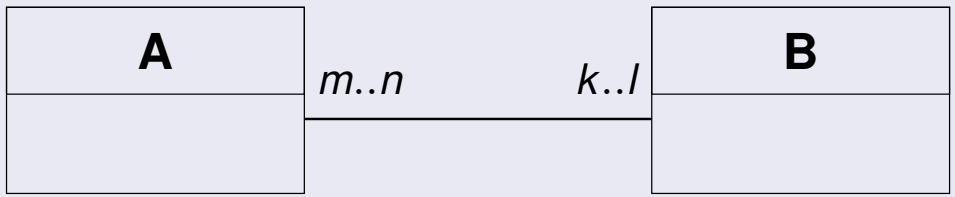
$$\{(person_1, auto_1), (person_2, auto_1), (person_1, auto_2), (person_3, auto_2)\}$$

und noch Existenz weiterer Personen, aber nicht weiterer Autos.

Beziehungen zwischen Klassen: Assoziation

Multiplizitäten allgemein:

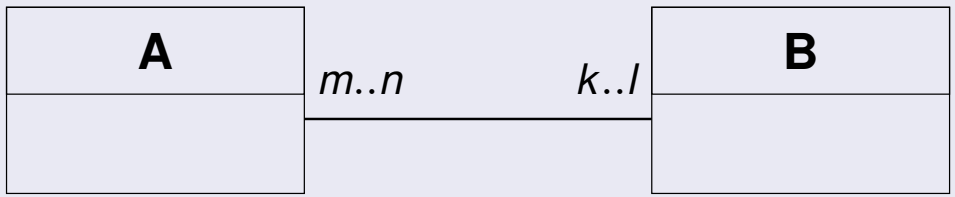
In folgendem Diagramm können i Instanzen von **A**, mit $m \leq i \leq n$, mit einer Instanz von **B** assoziiert sein. Umgekehrt können j Instanzen von **B**, mit $k \leq j \leq l$, mit einer Instanz von **A** assoziiert sein.



Beziehungen zwischen Klassen: Assoziation

Multiplizitäten allgemein:

In folgendem Diagramm können i Instanzen von **A**, mit $m \leq i \leq n$, mit einer Instanz von **B** assoziiert sein. Umgekehrt können j Instanzen von **B**, mit $k \leq j \leq l$, mit einer Instanz von **A** assoziiert sein.

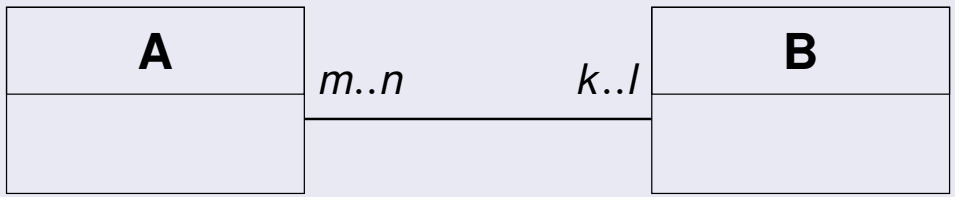


Falls es keine obere Schranke geben soll, wird ein Stern (= unendlich) verwendet.
Beispielsweise steht $2..*$ für „mindestens zwei“.

Beziehungen zwischen Klassen: Assoziation

Multiplizitäten allgemein:

In folgendem Diagramm können i Instanzen von **A**, mit $m \leq i \leq n$, mit einer Instanz von **B** assoziiert sein. Umgekehrt können j Instanzen von **B**, mit $k \leq j \leq l$, mit einer Instanz von **A** assoziiert sein.

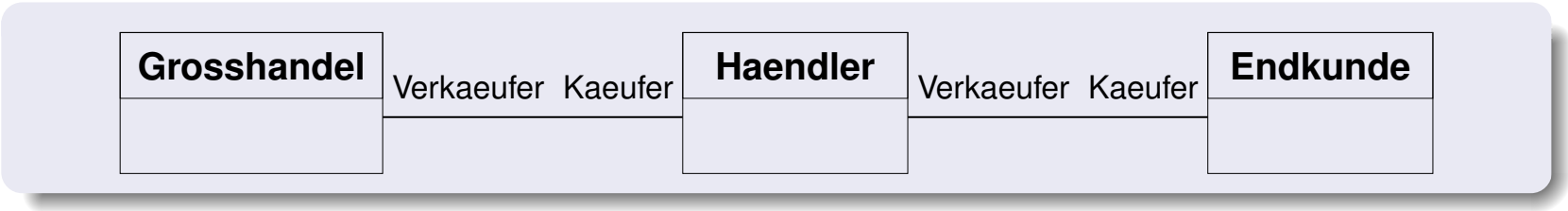


Falls es keine obere Schranke geben soll, wird ein Stern (= unendlich) verwendet.
Beispielsweise steht $2..*$ für „mindestens zwei“.

Der UML-Standard gibt keine Standardmultiplizität vor.
Für die folgenden Diagramme wird, wenn keine Angabe vorhanden ist, die Multiplizität $0..*$ als Standard angenommen.

Beziehungen zwischen Klassen: Assoziation

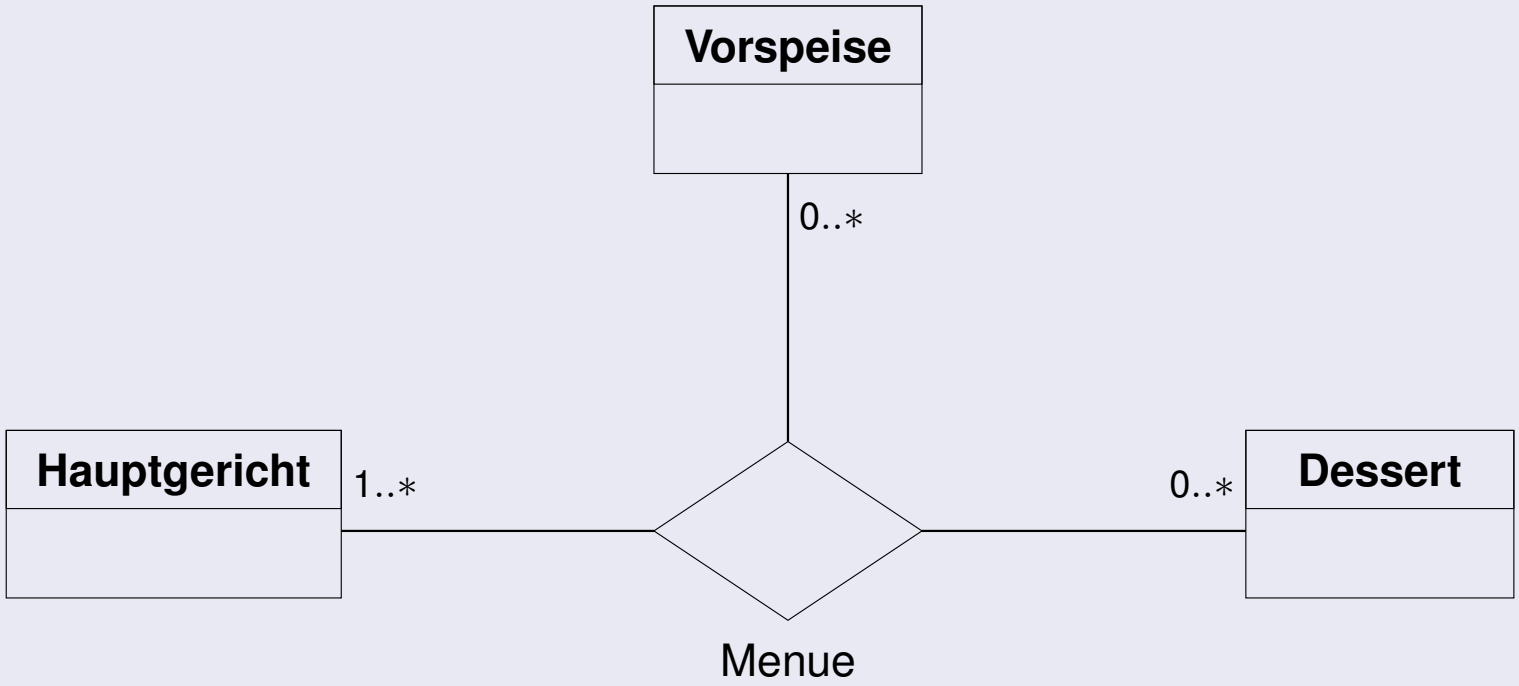
Klassen können in verschiedenen Assoziationen verschiedene Rollen spielen. Rollen werden auch an den Assoziationen notiert (und machen es manchmal überflüssig, die Assoziation selbst zu benennen oder eine Leserichtung anzugeben).



Beziehungen zwischen Klassen: Assoziation

n-äre Assoziation

Neben binären (zweistelligen) Assoziationen gibt es auch *n*-äre Assoziationen, die eine Beziehung zwischen $n > 2$ Klassen beschreiben.



Beziehungen zwischen Klassen: Aggregation

Die nächste Beziehung – Aggregation – ist etwas stärker.

Aggregation

Es gibt eine Aggregation zwischen den Klassen **A** und **B**, wenn

> Instanzen der Klasse **A** Instanzen der Klasse **B** als Teile enthalten. (Ein „Ganzes“ enthält mehrere „Teile“.)

Beziehungen zwischen Klassen: Aggregation

Die nächste Beziehung – Aggregation – ist etwas stärker.

Aggregation

Es gibt eine Aggregation zwischen den Klassen **A** und **B**, wenn

> Instanzen der Klasse **A** Instanzen der Klasse **B** als Teile enthalten. (Ein „Ganzes“ enthält mehrere „Teile“.)

Dabei ist durchaus denkbar, dass ein Teil zu mehreren Ganzen gehört.

Beziehungen zwischen Klassen: Aggregation

Die nächste Beziehung – Aggregation – ist etwas stärker.

Aggregation

Es gibt eine Aggregation zwischen den Klassen **A** und **B**, wenn

> Instanzen der Klasse **A** Instanzen der Klasse **B** als Teile enthalten. (Ein „Ganzes“ enthält mehrere „Teile“.)

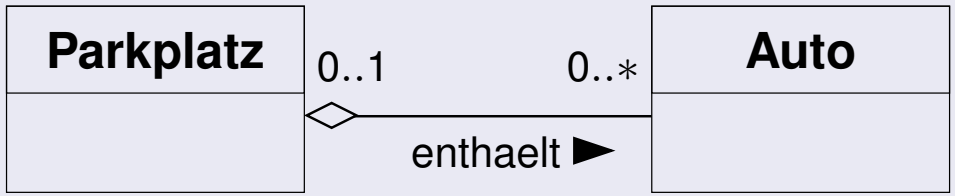
Dabei ist durchaus denkbar, dass ein Teil zu mehreren Ganzen gehört.

Eine explizite Benennung ist oft überflüssig, da aus der Angabe als Aggregation bereits die Natur der Beziehung folgt.

Beziehungen zwischen Klassen: Aggregation

Beispiel für eine Aggregation (mit Benennung)

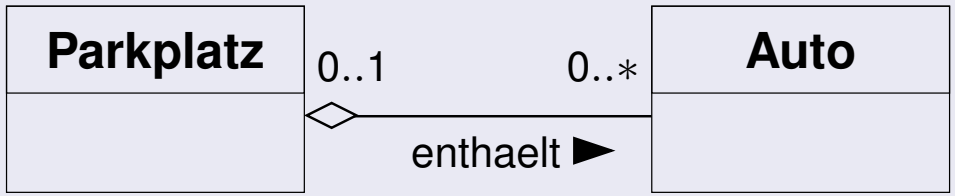
Ein Parkplatz „enthält“ mehrere Autos.



Beziehungen zwischen Klassen: Aggregation

Beispiel für eine Aggregation (mit Benennung)

Ein Parkplatz „enthält“ mehrere Autos.



Jedoch existiert ein Auto nicht nur solange es auf einem Parkplatz steht.

Beziehungen zwischen Klassen: Komposition

Die stärkste Beziehung ist die Komposition.

Komposition

Es gibt eine Komposition zwischen den Klassen **A** und **B**, wenn

> Instanzen der Klasse **A** Instanzen der Klasse **B** als Teile enthalten und die Lebenszeit der Teile vom „Ganzen“ kontrolliert wird.

Beziehungen zwischen Klassen: Komposition

Die stärkste Beziehung ist die Komposition.

Komposition

Es gibt eine Komposition zwischen den Klassen **A** und **B**, wenn

> Instanzen der Klasse **A** Instanzen der Klasse **B** als Teile enthalten und die Lebenszeit der Teile vom „Ganzen“ kontrolliert wird.
>
> Das heißt, die Teile können (oder müssen sogar) gelöscht werden, sobald die zugehörige Instanz der Klasse **A** gelöscht wird.

Beziehungen zwischen Klassen: Komposition

Die stärkste Beziehung ist die Komposition.

Komposition

Es gibt eine Komposition zwischen den Klassen **A** und **B**, wenn

- Instanzen der Klasse **A** Instanzen der Klasse **B** als Teile enthalten und die Lebenszeit der Teile vom „Ganzen“ kontrolliert wird.
- Das heißt, die Teile können (oder müssen sogar) gelöscht werden, sobald die zugehörige Instanz der Klasse **A** gelöscht wird.
- Außerdem (oder eigentlich wegen Obigem) darf ein Teil nicht gleichzeitig zu mehr als einem Ganzen gehören.

Beziehungen zwischen Klassen: Komposition

Die stärkste Beziehung ist die Komposition.

Komposition

Es gibt eine Komposition zwischen den Klassen **A** und **B**, wenn

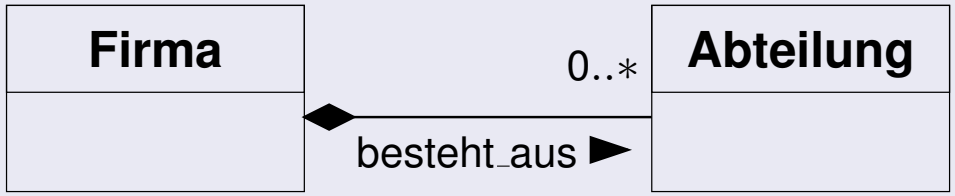
- Instanzen der Klasse **A** Instanzen der Klasse **B** als Teile enthalten und die Lebenszeit der Teile vom „Ganzen“ kontrolliert wird.
- Das heißt, die Teile können (oder müssen sogar) gelöscht werden, sobald die zugehörige Instanz der Klasse **A** gelöscht wird.
- Außerdem (oder eigentlich wegen Obigem) darf ein Teil nicht gleichzeitig zu mehr als einem Ganzen gehören.

Auch hier ist eine explizite Benennung oft überflüssig.

Beziehungen zwischen Klassen: Komposition

Beispiel für eine Komposition (mit Benennung)

Eine Firma besteht aus einer beliebigen Anzahl Abteilungen.

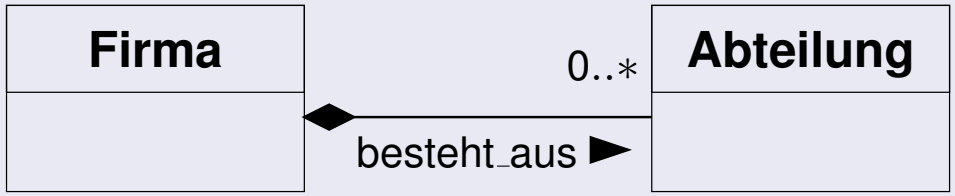


Die Abteilungen existieren nicht mehr, sobald die Firma nicht mehr existiert.

Beziehungen zwischen Klassen: Komposition

Beispiel für eine Komposition (mit Benennung)

Eine Firma besteht aus einer beliebigen Anzahl Abteilungen.



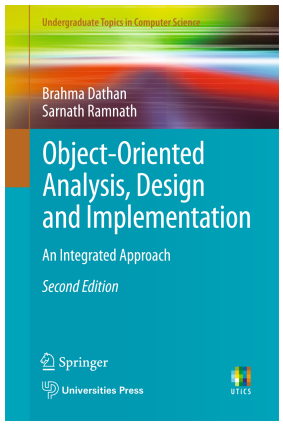
Die Abteilungen existieren nicht mehr, sobald die Firma nicht mehr existiert.

Bemerkung: Bei einer Komposition darf die Multiplizität, die an der schwarzen Raute stünde, nur 0..1 oder 1 sein. Am üblichsten ist 1, dementsprechend wird in dem Fall an diesem Ende gar keine Multiplizität angegeben: jedes Teil gehört zu genau einem Ganzen.

Beziehungen zwischen Klassen

„Merksätze“ (aus: Dathan & Ramnath, Object-Oriented Analysis, Design and Implementation – An Integrated Approach, siehe Literaturfolien zu Beginn des Semesters):

- An association normally represents something that will be stored as part of the data and reflects all links between objects of two classes that may ever exist. It describes a relationship that will exist between instances at run time and has an example.



Beziehungen zwischen Klassen

„Merksätze“ (aus: Dathan & Ramnath, Object-Oriented Analysis, Design and Implementation – An Integrated Approach, siehe Literaturfolien zu Beginn des Semesters):

- An association normally represents something that will be stored as part of the data and reflects all links between objects of two classes that may ever exist. It describes a relationship that will exist between instances at run time and has an example.
- . . . , associations should be shown if a class possesses, controls, is connected to, is related to, is a part of, has as parts, is a member of, or has as members some other class in the system.

Beziehungen zwischen Klassen

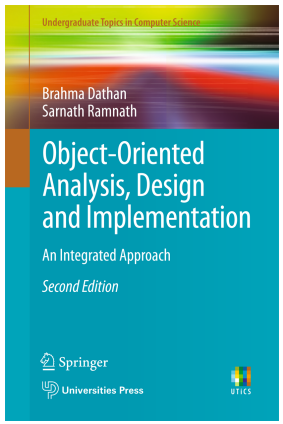
„Merksätze“ (aus: Dathan & Ramnath, Object-Oriented Analysis, Design and Implementation – An Integrated Approach, siehe Literaturfolien zu Beginn des Semesters):

- An association normally represents something that will be stored as part of the data and reflects all links between objects of two classes that may ever exist. It describes a relationship that will exist between instances at run time and has an example.
- . . . , associations should be shown if a class possesses, controls, is connected to, is related to, is a part of, has as parts, is a member of, or has as members some other class in the system.
- . . . association should not be used to denote relationships that: (i) can be drawn as a hierarchy, (ii) stems from a dependency alone, (iii) or relationships whose links will not survive beyond the execution of any particular operation.

Beziehungen zwischen Klassen

„Merksätze“ (aus: Dathan & Ramnath, Object-Oriented Analysis, Design and Implementation – An Integrated Approach, siehe Literaturfolien zu Beginn des Semesters):

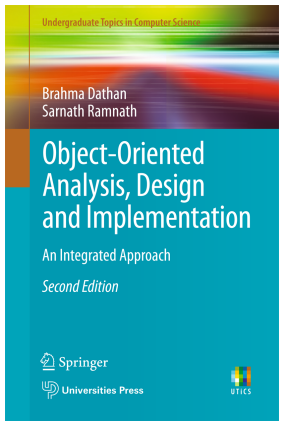
- Aggregation is a kind of association where the object of class A is ‘made up of’ objects of class B. This suggests some kind of whole-part relationship between A and B.



Beziehungen zwischen Klassen

„Merksätze“ (aus: Dathan & Ramnath, Object-Oriented Analysis, Design and Implementation – An Integrated Approach, siehe Literaturfolien zu Beginn des Semesters):

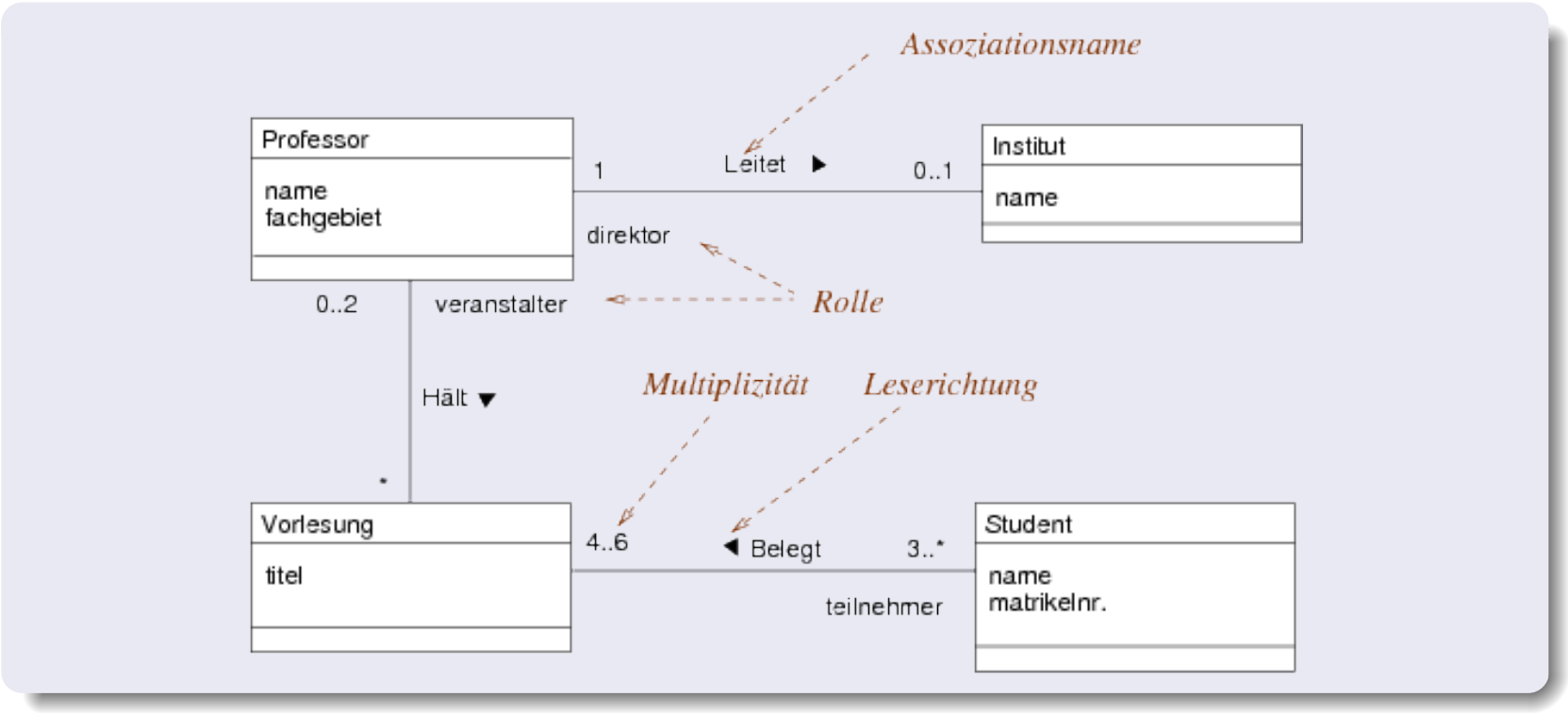
- Aggregation is a kind of association where the object of class A is ‘made up of’ objects of class B. This suggests some kind of whole-part relationship between A and B.
- Composition implies that each instance of the part belongs to only one instance of the whole, and that the part cannot exist except as part of the whole.



Beziehungen zwischen Klassen

In Klassendiagrammen befinden sich normalerweise nicht nur zwei Klassen mit irgendeiner Beziehung, sondern viele verschiedene Klassen eines Programms oder Moduls, mit ihren diversen Beziehungen untereinander.

Beispiel:



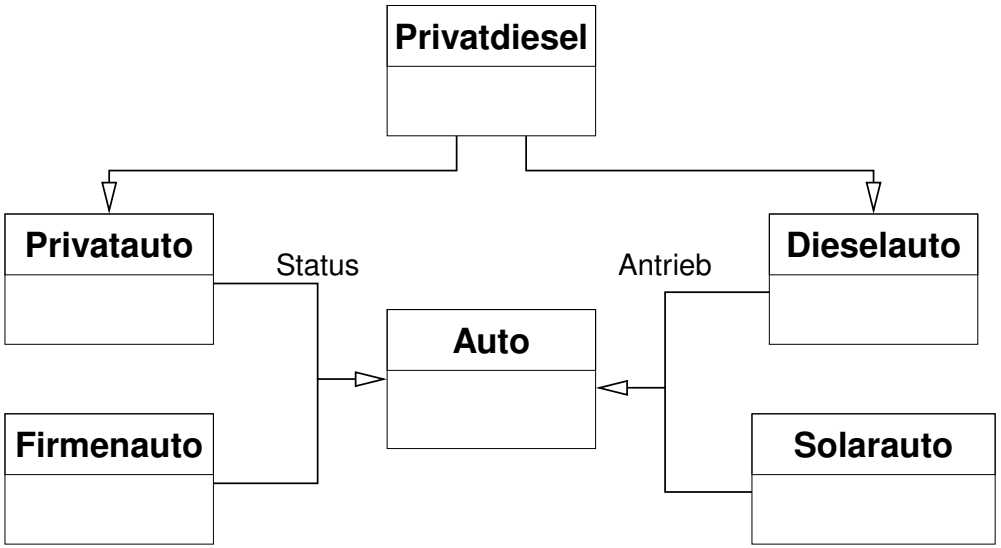
Beziehungen zwischen Klassen

Geeigneter Einsatz von Assoziation / Aggregation / Komposition kann auch Vererbungen überflüssig machen.

Beziehungen zwischen Klassen

Geeigneter Einsatz von Assoziation / Aggregation / Komposition kann auch Vererbungen überflüssig machen.

Zum Beispiel ist konzeptionell denkbare Mehrfachvererbung:

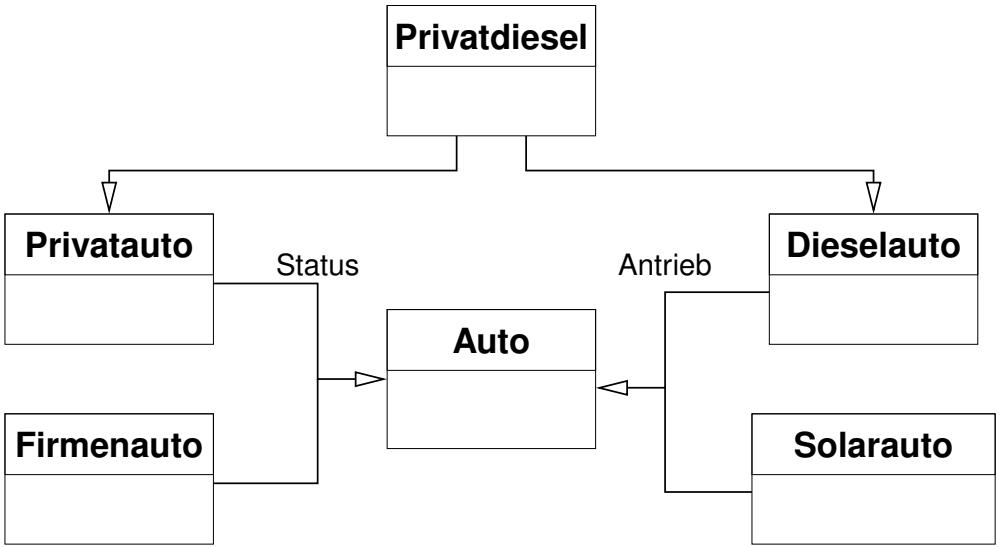


in der Praxis eher problematisch.

Beziehungen zwischen Klassen

Geeigneter Einsatz von Assoziation / Aggregation / Komposition kann auch Vererbungen überflüssig machen.

Zum Beispiel ist konzeptionell denkbare Mehrfachvererbung:

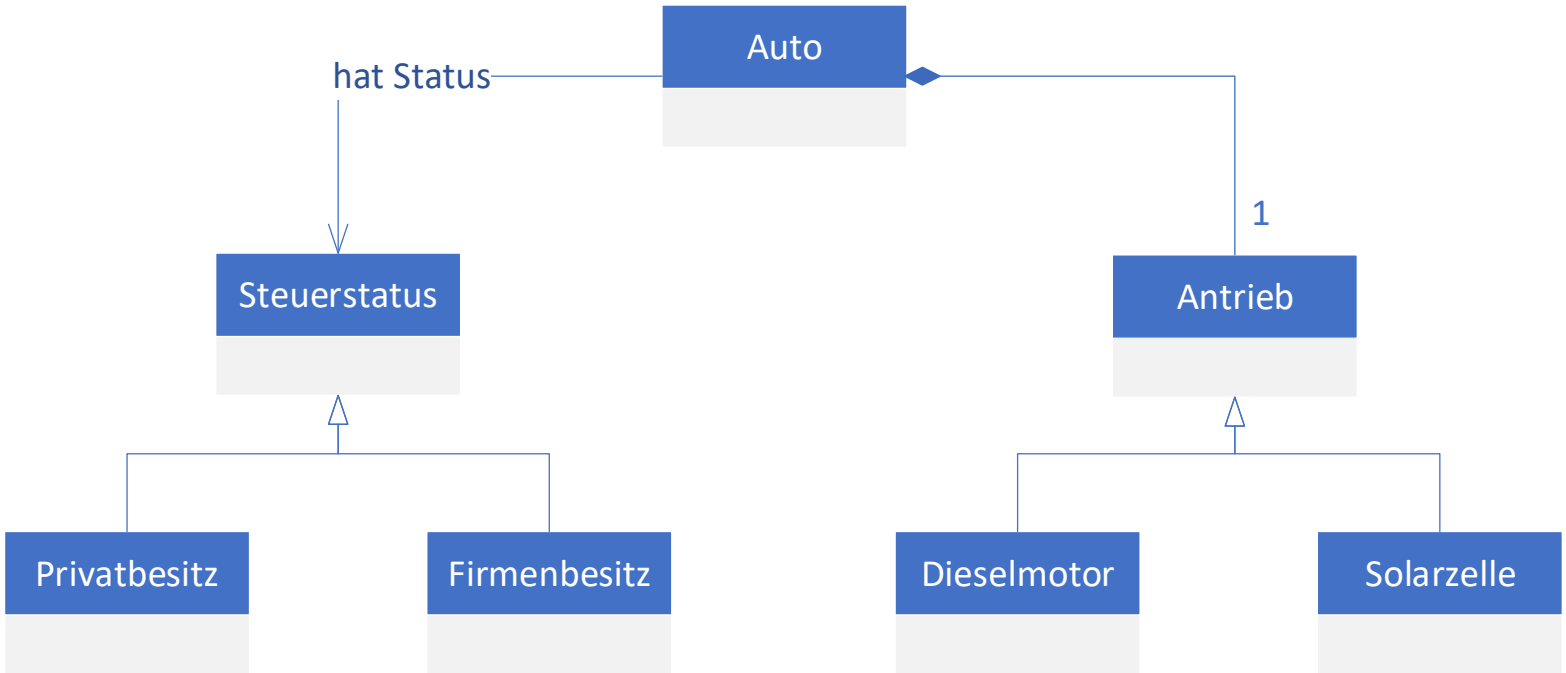


in der Praxis eher problematisch.

Eine alternative Modellierung ist hier möglich . . .

Beziehungen zwischen Klassen

. . . durch Repräsentation der Aspekte „steuerlicher Status“ und „Antrieb“ in eigenen Klassen, und deren Verwendung über Assoziation / Aggregation / Komposition, etwa wie folgt:



Systemmodellierung

Vor Implementierung eines objekt-orientierten Systems stellen sich bei der Modellierung insbesondere folgende Fragen:

- Welche Objekte und Klassen werden benötigt?
- Welche Merkmale haben diese Klassen und welche Beziehungen bestehen zwischen Ihnen?
- Welche Methoden stellen diese Klassen zur Verfügung? Wie wirken diese Methoden zusammen?
- In welchen Zuständen können sich Objekte befinden und welche Nachrichten werden wann an andere Objekte geschickt?

Systemmodellierung

Vor Implementierung eines objekt-orientierten Systems stellen sich bei der Modellierung insbesondere folgende Fragen:

- Welche Objekte und Klassen werden benötigt?
- Welche Merkmale haben diese Klassen und welche Beziehungen bestehen zwischen Ihnen?
- Welche Methoden stellen diese Klassen zur Verfügung? Wie wirken diese Methoden zusammen?
- In welchen Zuständen können sich Objekte befinden und welche Nachrichten werden wann an andere Objekte geschickt?

Zur Beantwortung kennt die Literatur bestimmte mehr oder weniger systematische Vorgehensweisen.

Beispiel: Modellierung einer Bibliothek

Am Beispiel, Klassen finden: Use Case “Register New Member”

Actions performed	System responses
1. The customer fills out an application form containing the customer’s name, address and phone number, and gives this to the clerk.	
2. The clerk issues a request to add a new member.	
	3. The system asks for data about the new member.
4. The clerk enters the data into the system.	
	5. Reads in the data, and if the member can be added, generates an identification number and remembers information about the member. Informs the clerk whether the member was added and outputs the member’s name, address, phone and id number.
6. The clerk gives the user their identification number.	

Beispiel: Modellierung einer Bibliothek

An Naivität kaum zu übertreffen: Alle Nomen heraussuchen

Actions performed	System responses
1. The **customer** fills out an **application form** containing the **customer's name**, **address** and **phone number**, and gives this to the **clerk**.	
2. The **clerk** issues a **request** to add a new **member**.	
	3. The **system** asks for **data** about the new **member**.
4. The **clerk** enters the **data** into the **system**.	
	5. Reads in the **data**, and if the **member** can be added, generates an **identification number** and remembers **information** about the **member**. Informs the **clerk** whether the **member** was added and outputs the **member's name**, **address**, **phone** and **id number**.
6. The **clerk** gives the **user** their **identification number**.	

Beispiel: Modellierung einer Bibliothek

Bereinigung unserer Wortsammlung:

- Als (zusammengesetzte) Einheiten stechen hervor: **member**, **system**

Beispiel: Modellierung einer Bibliothek

Bereinigung unserer Wortsammlung:

- Als (zusammengesetzte) Einheiten stechen hervor: **member**, **system**
- Bezüglich der anderen vorkommenden Nomen:
  - **customer** – wird ein **member**, ist also ein Synonym

Beispiel: Modellierung einer Bibliothek

Bereinigung unserer Wortsammlung:

- Als (zusammengesetzte) Einheiten stechen hervor: **member**, **system**
- Bezüglich der anderen vorkommenden Nomen:
  - **customer** – wird ein **member**, ist also ein Synonym
  - **user** – ein weiteres Synonym

Beispiel: Modellierung einer Bibliothek

Bereinigung unserer Wortsammlung:

- Als (zusammengesetzte) Einheiten stechen hervor: **member**, **system**
- Bezüglich der anderen vorkommenden Nomen:
  - **customer** – wird ein **member**, ist also ein Synonym
  - **user** – ein weiteres Synonym
  - **application form** – ist ein externes Konstrukt zur Informationsabfrage

Beispiel: Modellierung einer Bibliothek

Bereinigung unserer Wortsammlung:

- Als (zusammengesetzte) Einheiten stechen hervor: **member**, **system**
- Bezüglich der anderen vorkommenden Nomen:
  - **customer** – wird ein **member**, ist also ein Synonym
  - **user** – ein weiteres Synonym
  - **application form** – ist ein externes Konstrukt zur Informationsabfrage
  - **request** – nur ein Menüeintrag, wird wie **application form** nicht Teil einer Datenstruktur sein

Beispiel: Modellierung einer Bibliothek

Bereinigung unserer Wortsammlung:

- Als (zusammengesetzte) Einheiten stechen hervor: **member**, **system**
- Bezüglich der anderen vorkommenden Nomen:
  - **customer** – wird ein **member**, ist also ein Synonym
  - **user** – ein weiteres Synonym
  - **application form** – ist ein externes Konstrukt zur Informationsabfrage
  - **request** – nur ein Menüeintrag, wird wie **application form** nicht Teil einer Datenstruktur sein
  - **customer's name**, **address**, **phone number** – Attribute von **member**

Beispiel: Modellierung einer Bibliothek

Bereinigung unserer Wortsammlung:

- Als (zusammengesetzte) Einheiten stechen hervor: **member**, **system**
- Bezüglich der anderen vorkommenden Nomen:
  - **customer** – wird ein **member**, ist also ein Synonym
  - **user** – ein weiteres Synonym
  - **application form** – ist ein externes Konstrukt zur Informationsabfrage
  - **request** – nur ein Menüeintrag, wird wie **application form** nicht Teil einer Datenstruktur sein
  - **customer's name**, **address**, **phone number** – Attribute von **member**
  - **clerk** – lediglich ein Akteur, hat keine Repräsentation in Software

Beispiel: Modellierung einer Bibliothek

Bereinigung unserer Wortsammlung:

- Als (zusammengesetzte) Einheiten stechen hervor: **member**, **system**
- Bezüglich der anderen vorkommenden Nomen:
  - **customer** – wird ein **member**, ist also ein Synonym
  - **user** – ein weiteres Synonym
  - **application form** – ist ein externes Konstrukt zur Informationsabfrage
  - **request** – nur ein Menüeintrag, wird wie **application form** nicht Teil einer Datenstruktur sein
  - **customer's name**, **address**, **phone number** – Attribute von **member**
  - **clerk** – lediglich ein Akteur, hat keine Repräsentation in Software
  - **identification number** – wird Teil von **member**

Beispiel: Modellierung einer Bibliothek

Bereinigung unserer Wortsammlung:

- Als (zusammengesetzte) Einheiten stechen hervor: **member**, **system**
- Bezüglich der anderen vorkommenden Nomen:
  - **customer** – wird ein **member**, ist also ein Synonym
  - **user** – ein weiteres Synonym
  - **application form** – ist ein externes Konstrukt zur Informationsabfrage
  - **request** – nur ein Menüeintrag, wird wie **application form** nicht Teil einer Datenstruktur sein
  - **customer's name**, **address**, **phone number** – Attribute von **member**
  - **clerk** – lediglich ein Akteur, hat keine Repräsentation in Software
  - **identification number** – wird Teil von **member**
  - **data**, **information** – werden als **member** gespeichert

Beispiel: Modellierung einer Bibliothek

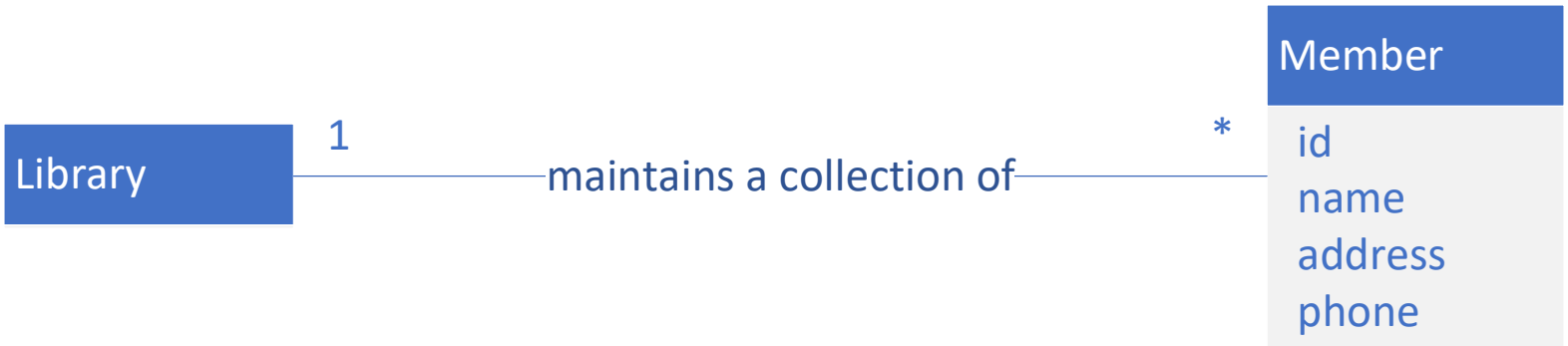
Ermittlung von Beziehungen:

"... [the **system**] remembers **information** about the **member**"

Beispiel: Modellierung einer Bibliothek

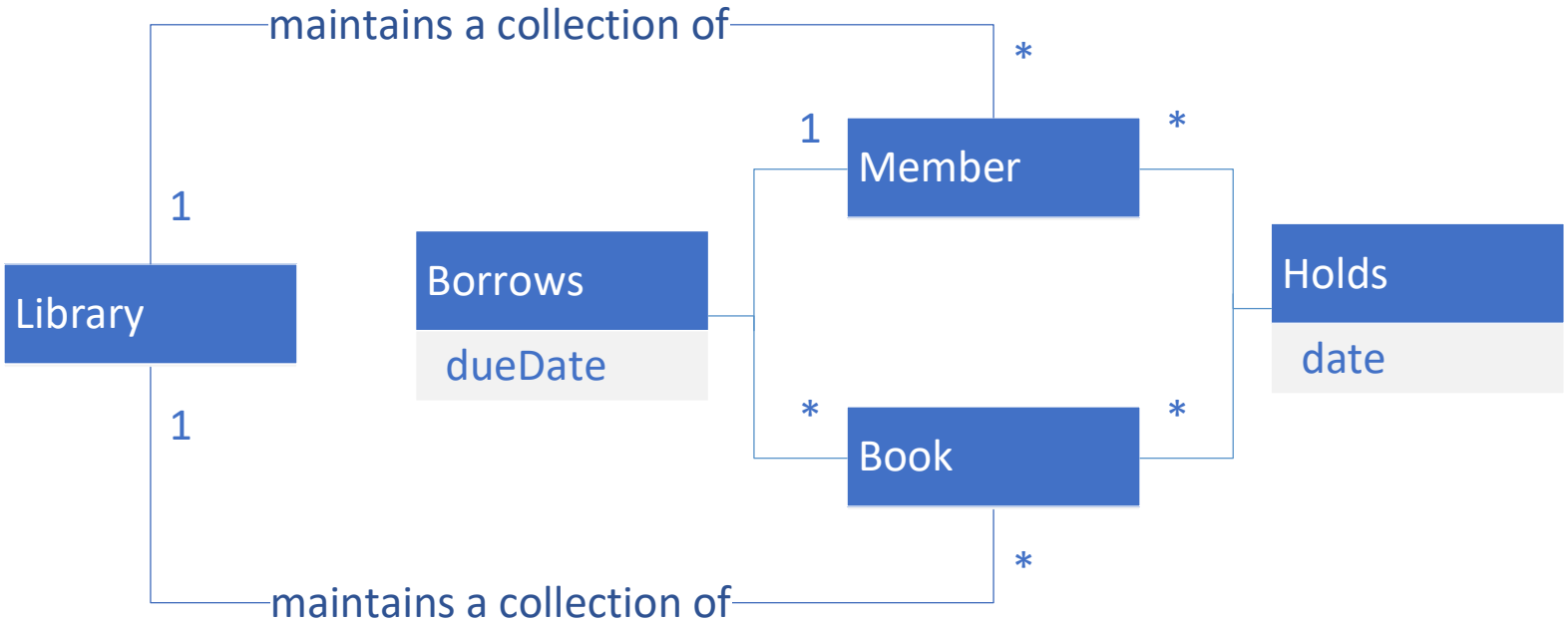
Ermittlung von Beziehungen:

"... [the **system**] remembers **information** about the **member**"



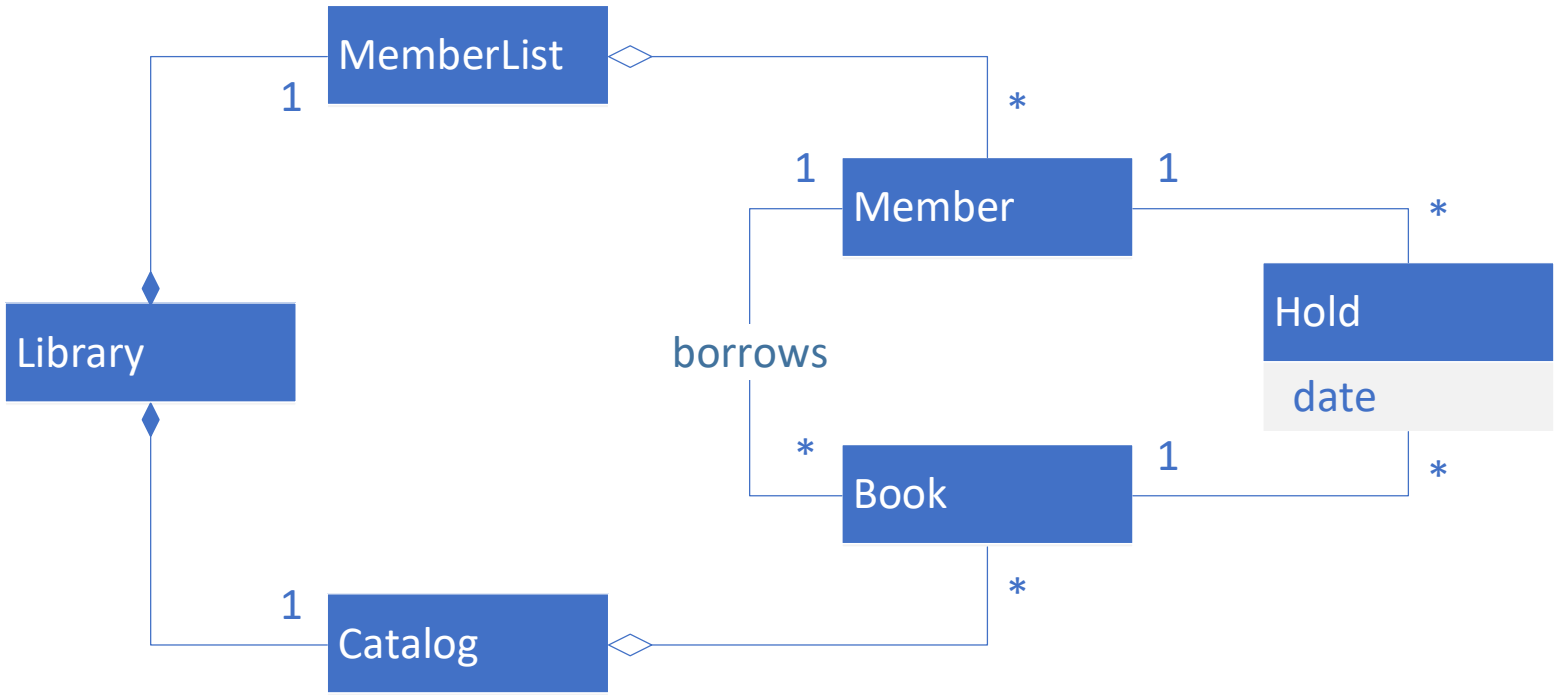
Beispiel: Modellierung einer Bibliothek

Informiert durch weitere Use Cases:



Beispiel: Modellierung einer Bibliothek

Verfeinert für die Implementierung:



Beispiel: Modellierung einer Bank

Ein weiteres Beispiel für objekt-orientierte Modellierung:

Wir modellieren eine Bank.

Beispiel: Modellierung einer Bank

Ein weiteres Beispiel für objekt-orientierte Modellierung:

Wir modellieren eine Bank.

Folgende Anforderungen werden gestellt:

- Eine Bank
  - hat mehrere Kunden

Beispiel: Modellierung einer Bank

Ein weiteres Beispiel für objekt-orientierte Modellierung:

Wir modellieren eine Bank.

Folgende Anforderungen werden gestellt:

- Eine Bank
  - hat mehrere Kunden
  - und mehrere Angestellte

Beispiel: Modellierung einer Bank

Ein weiteres Beispiel für objekt-orientierte Modellierung:

Wir modellieren eine Bank.

Folgende Anforderungen werden gestellt:

- Eine Bank
  - hat mehrere Kunden
  - und mehrere Angestellte
  - und führt eine Menge von Konten.

Beispiel: Modellierung einer Bank

Ein weiteres Beispiel für objekt-orientierte Modellierung:

Wir modellieren eine Bank.

Folgende Anforderungen werden gestellt:

- Eine Bank
  - hat mehrere Kunden
  - und mehrere Angestellte
  - und führt eine Menge von Konten.
- Konten können Giro- oder Sparkonten sein. (Ein Sparkonto wirft höhere Zinsen ab, darf aber nicht ins Minus absinken.)

Beispiel: Modellierung einer Bank

Ein weiteres Beispiel für objekt-orientierte Modellierung:

Wir modellieren eine Bank.

Folgende Anforderungen werden gestellt:

- Eine Bank
  - hat mehrere Kunden
  - und mehrere Angestellte
  - und führt eine Menge von Konten.
- Konten können Giro- oder Sparkonten sein. (Ein Sparkonto wirft höhere Zinsen ab, darf aber nicht ins Minus absinken.)
- Auf den Konten sollen folgende Operationen ausgeführt werden können:
  - Einzahlen
  - Abheben
  - Umbuchen
  - Verzinsen

Beispiel: Modellierung einer Bank

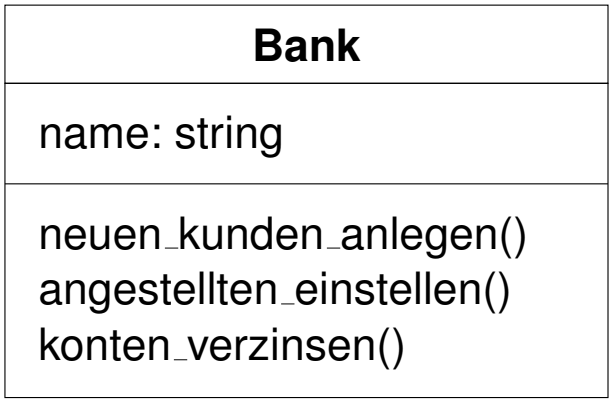
Klasse Bank:

Bank
name: string
neuen_kunden_anlegen() angestellten_einstellen() konten_verzinsen()

Methoden: neuen Kunden anlegen; neuen Angestellten einstellen; alle Konten verzinsen

Beispiel: Modellierung einer Bank

Klasse Bank:

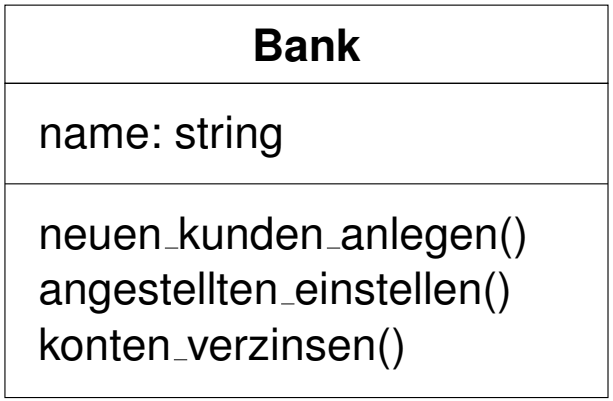


Methoden: neuen Kunden anlegen; neuen Angestellten einstellen; alle Konten verzinsen

Außerdem: Eine Bank besteht (in Kompositionsbeziehung) aus einer Menge von Konten, die verschwinden, wenn die Bank verschwindet.

Beispiel: Modellierung einer Bank

Klasse Bank:



Methoden: neuen Kunden anlegen; neuen Angestellten einstellen; alle Konten verzinsen

Außerdem: Eine Bank besteht (in Kompositionsbeziehung) aus einer Menge von Konten, die verschwinden, wenn die Bank verschwindet. Und es gibt Aggregationen mit je einer Menge von Kunden und von Angestellten.

Beispiel: Modellierung einer Bank

Klasse Konto:

Konto
nummer: string zins: float kontostand: float
abheben(wert: float) umbuchen_auf(kto: Konto, wert: float einzahlen(wert: float) verzinsen()

Attribute: Kontonummer, Zins, Kontostand

Methoden: einzahlen, abheben, umbuchen eines Betrags auf ein anderes Konto, Konto verzinsen

Beispiel: Modellierung einer Bank

Girokonto: Die Klasse Girokonto wird von Konto abgeleitet. Typischerweise ist der Zins bei Girokonten niedriger als bei Sparkonten. Daher wird dieser auf einen sehr niedrigen Anfangswert gesetzt.

Sparkonto: Bei der Klasse Sparkonto muss – wie beim Girokonto – ein bestimmter Anfangswert für den Zins gesetzt werden.

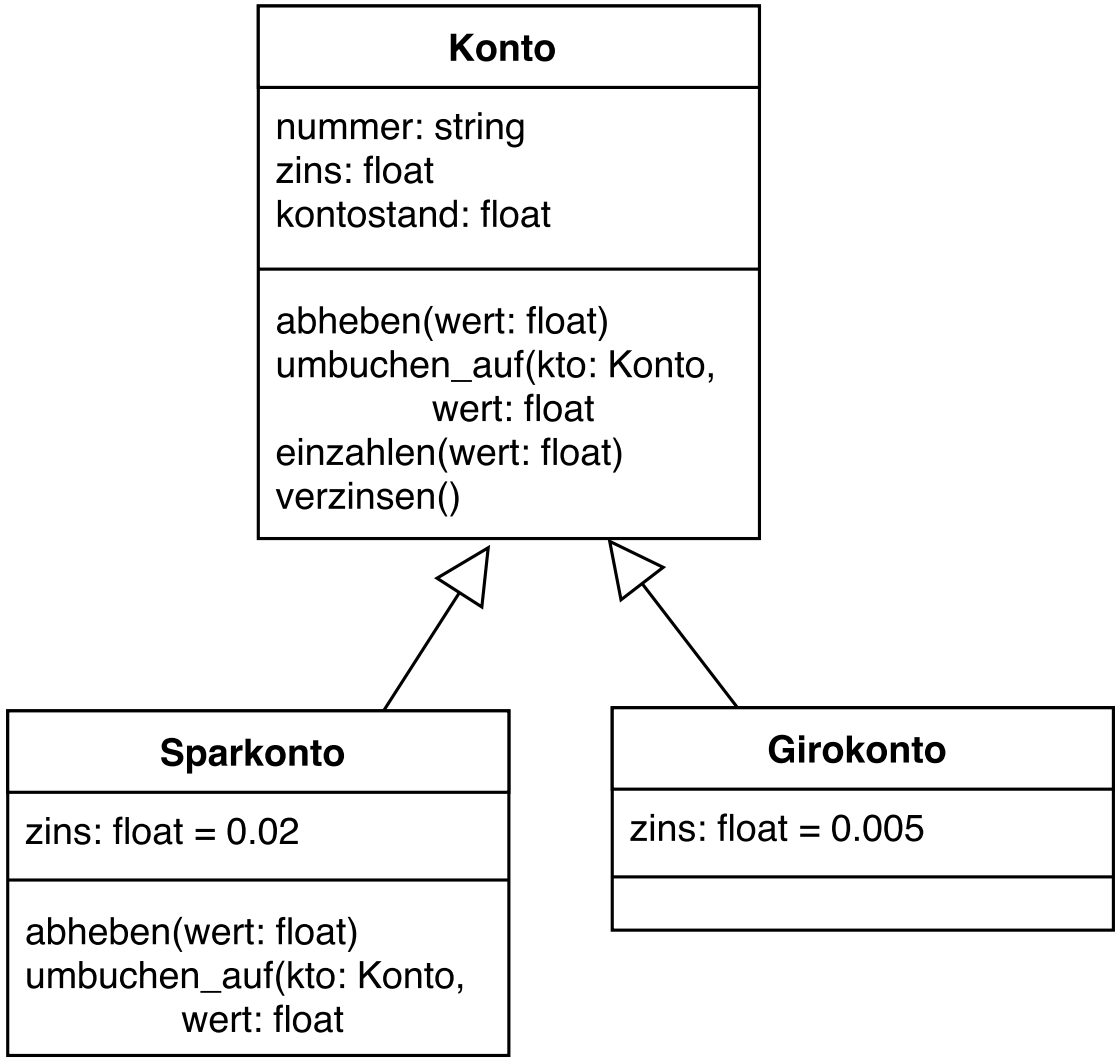
Beispiel: Modellierung einer Bank

Girokonto: Die Klasse Girokonto wird von Konto abgeleitet. Typischerweise ist der Zins bei Girokonten niedriger als bei Sparkonten. Daher wird dieser auf einen sehr niedrigen Anfangswert gesetzt.

Sparkonto: Bei der Klasse Sparkonto muss – wie beim Girokonto – ein bestimmter Anfangswert für den Zins gesetzt werden.

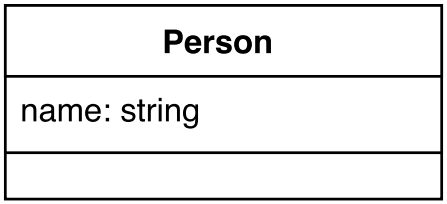
Außerdem: Es muss darauf geachtet werden, dass das Konto nicht ins Minus abgleitet. Dazu werden die entsprechenden Methoden überschrieben. (In der Implementierung muss die Bedingung entsprechend getestet werden.)

Beispiel: Modellierung einer Bank



Beispiel: Modellierung einer Bank

Klasse Person:



Beispiel: Modellierung einer Bank

Klasse Person:

Person
name: string

Jede Person steht in einer Assoziationsbeziehung mit einer Menge von Konten, die entweder dieser Person gehören (Kunde) oder auf die diese Person Zugriff hat (Angestellter).

Beispiel: Modellierung einer Bank

Angestellter: Methoden: z.B. Zugriff auf ein Konto erlangen

Kunde: Methoden: z.B. Konto eröffnen

Beispiel: Modellierung einer Bank

Angestellter: Methoden: z.B. Zugriff auf ein Konto erlangen

Kunde: Methoden: z.B. Konto eröffnen (Dabei könnte noch ein Parameter übergeben werden, der beschreibt, ob das Konto ein Giro- oder Sparkonto sein soll und in welcher Währung es geführt werden soll.)

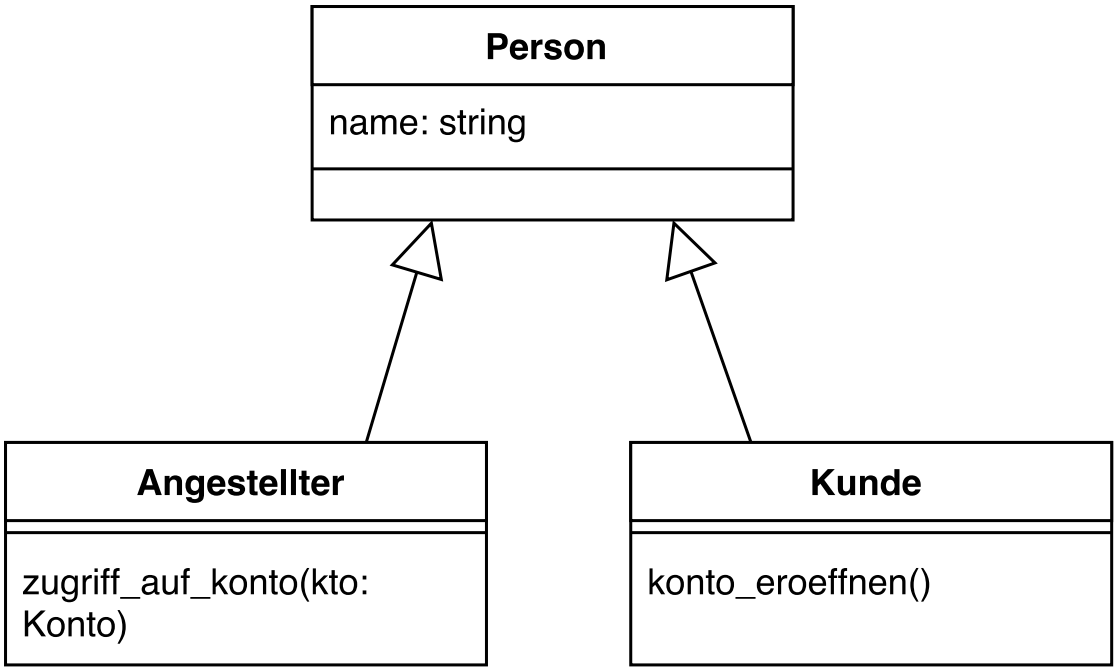
Beispiel: Modellierung einer Bank

Angestellter: Methoden: z.B. Zugriff auf ein Konto erlangen

Kunde: Methoden: z.B. Konto eröffnen (Dabei könnte noch ein Parameter übergeben werden, der beschreibt, ob das Konto ein Giro- oder Sparkonto sein soll und in welcher Währung es geführt werden soll.)

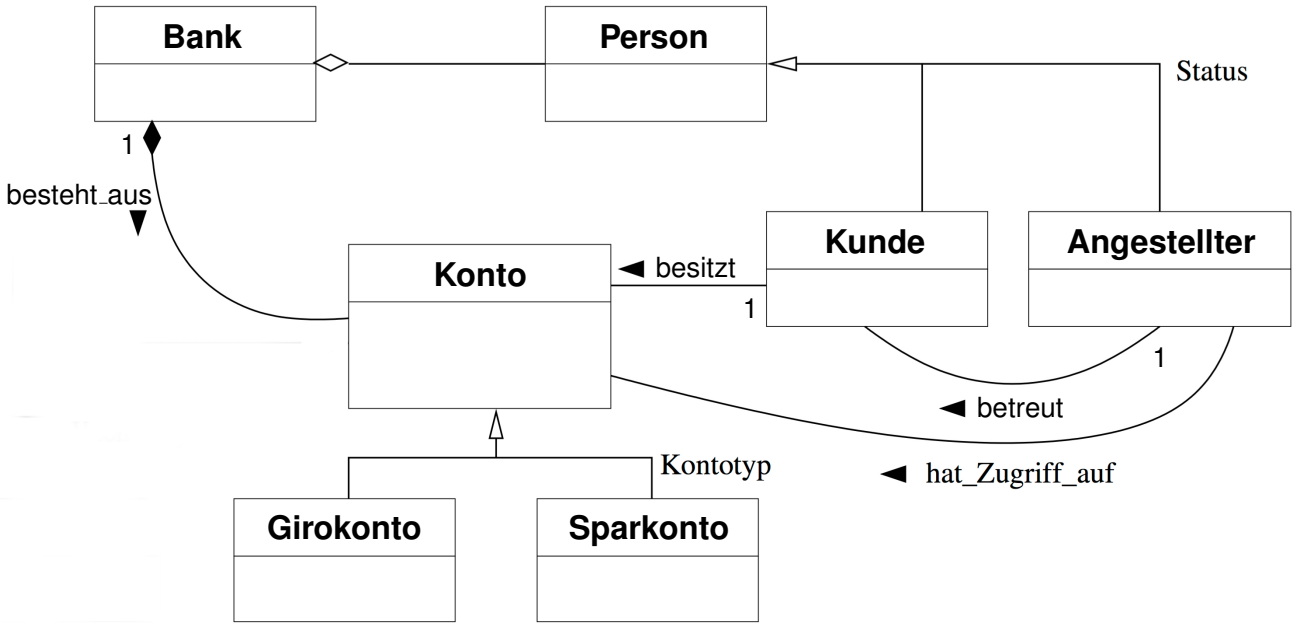
Außerdem: Ein Kunde steht in einer Assoziationsbeziehung mit einem ihn betreuenden Angestellten.

Beispiel: Modellierung einer Bank



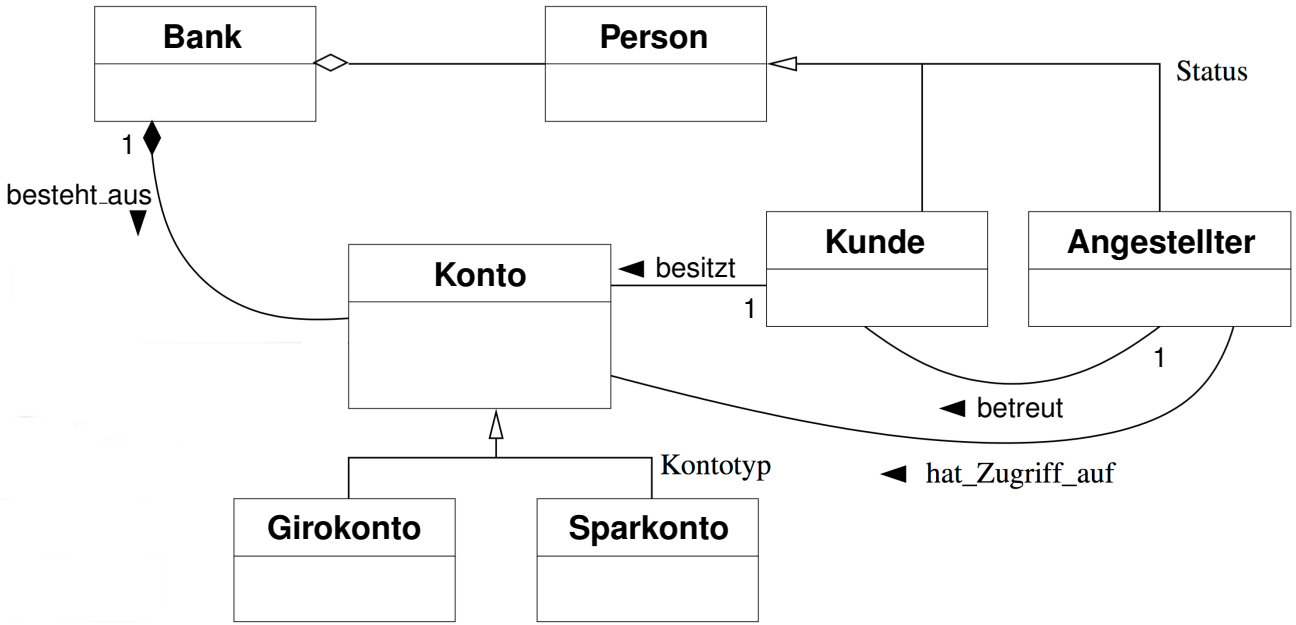
Beispiel: Modellierung einer Bank

Insgesamt:



Beispiel: Modellierung einer Bank

Insgesamt:



Allerdings gibt es eine Abweichung in diesem Modell vom zuvor Gesagten. Welche?

Abstrakte Klassen

Klassen ohne (vollständige) Implementierung nennt man abstrakt.
Sie können nicht konkret instanziiert werden.

Abstrakte Klasse

Der Klassenname wird *kursiv* geschrieben.

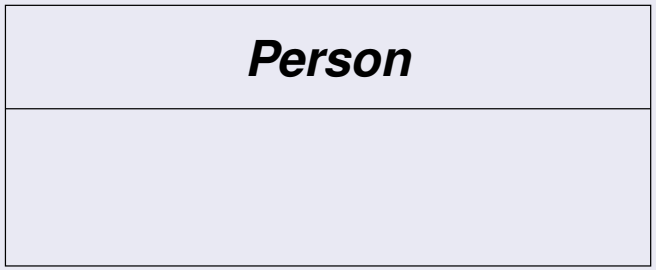
Abstrakte Klassen

Klassen ohne (vollständige) Implementierung nennt man abstrakt. Sie können nicht konkret instanziiert werden.

Abstrakte Klasse

Der Klassenname wird *kursiv* geschrieben.

In dem Bank-Beispiel möchte man etwa nie eine Instanz der Klasse Person bilden, sondern immer nur von Kunde oder Angestellter.



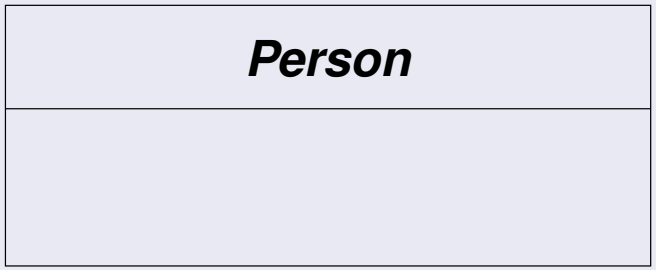
Abstrakte Klassen

Klassen ohne (vollständige) Implementierung nennt man abstrakt. Sie können nicht konkret instanziiert werden.

Abstrakte Klasse

Der Klassenname wird *kursiv* geschrieben.

In dem Bank-Beispiel möchte man etwa nie eine Instanz der Klasse Person bilden, sondern immer nur von Kunde oder Angestellter.

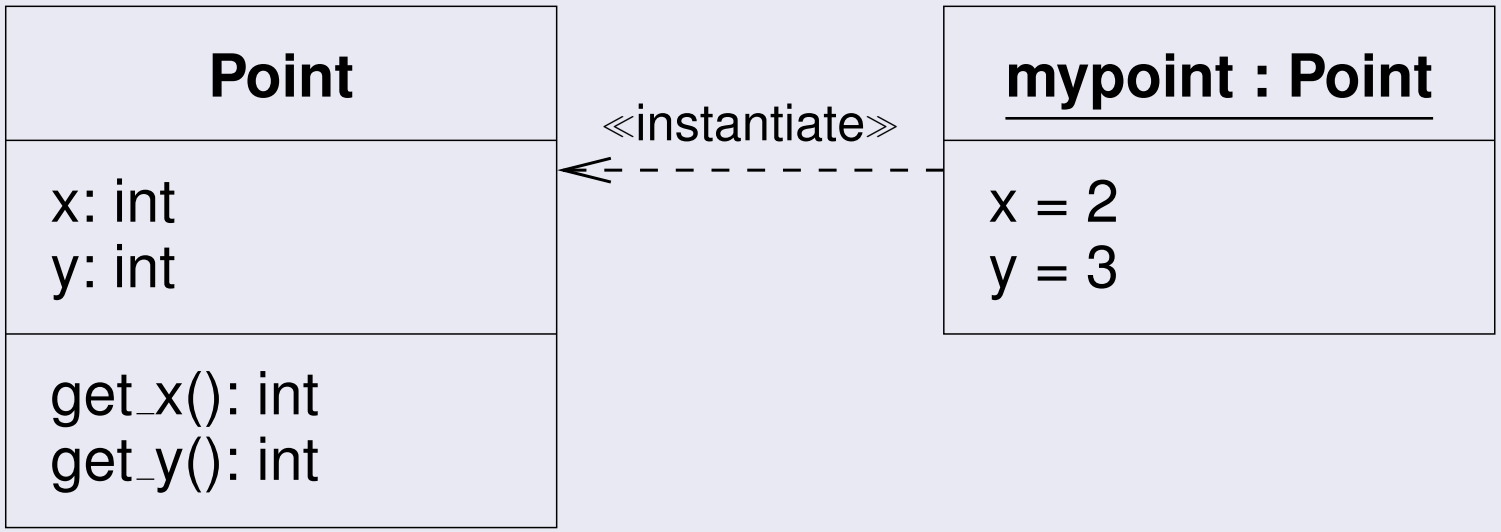


Eine Vererbungsbeziehung zu einer abstrakten Klasse ist allgemein besseres Design als zu einer Implementierungsklasse.

Objektdiagramme

Wir betrachten nun Objektdiagramme. Ein Objekt ist eine Ausprägung einer Klasse.

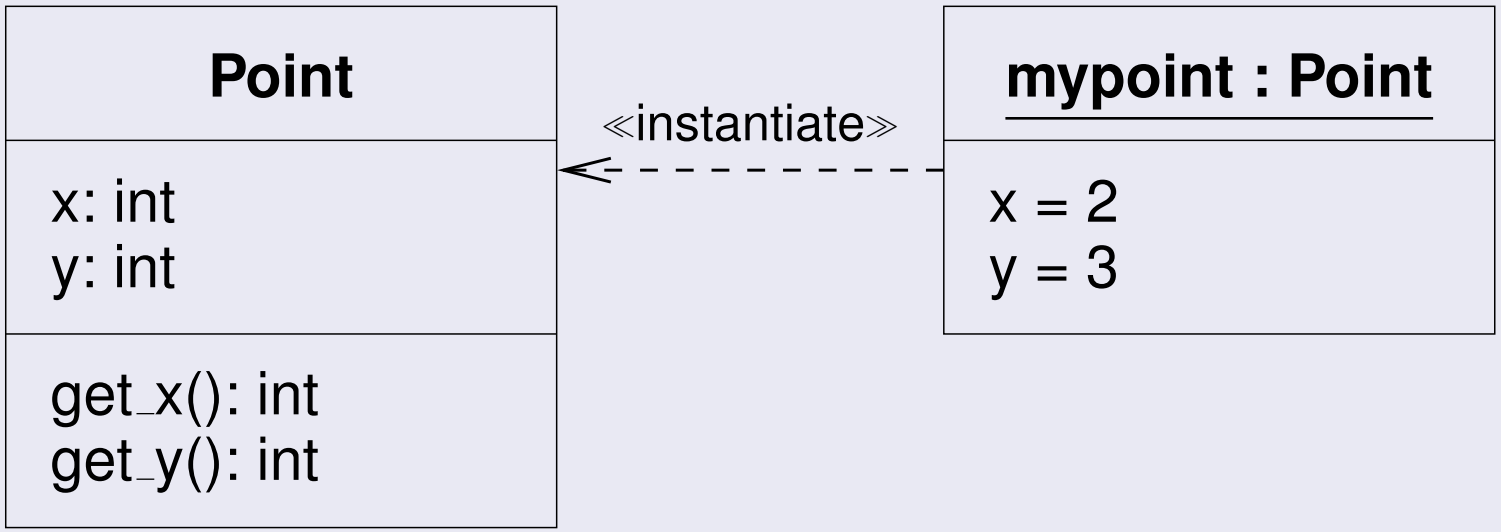
Klassen und Objekte



Objektdiagramme

Wir betrachten nun Objektdiagramme. Ein Objekt ist eine Ausprägung einer Klasse.

Klassen und Objekte



Ein Objektdiagramm beschreibt eine Art Momentaufnahme des Systems: eine Menge von Objekten, wie sie zu einem bestimmten Zeitpunkt vorhanden sind, samt ihren Beziehungen zueinander.

Objektdiagramme

Anmerkungen:

- Ein Objekt kann, muss aber nicht, durch einen Namen bezeichnet werden. Es muss aber die Klasse angegeben werden, von der dieses Objekt eine Ausprägung ist (in der Form : Point).

Objektdiagramme

Anmerkungen:

- Ein Objekt kann, muss aber nicht, durch einen Namen bezeichnet werden. Es muss aber die Klasse angegeben werden, von der dieses Objekt eine Ausprägung ist (in der Form : Point).
- Die Klassen müssen in dem Diagramm nicht mit abgebildet werden. Es kann aber manchmal angemessen sein, dies zu tun, also sozusagen „Bauplan“ und „Produkt“ gemeinsam darzustellen.

Objektdiagramme

Anmerkungen:

- Ein Objekt kann, muss aber nicht, durch einen Namen bezeichnet werden. Es muss aber die Klasse angegeben werden, von der dieses Objekt eine Ausprägung ist (in der Form : Point).
- Die Klassen müssen in dem Diagramm nicht mit abgebildet werden. Es kann aber manchmal angemessen sein, dies zu tun, also sozusagen „Bauplan“ und „Produkt“ gemeinsam darzustellen.
- Nicht alle Attributbelegungen eines Objekts müssen angegeben werden.

Objektdiagramme

Anmerkungen:

- Ein Objekt kann, muss aber nicht, durch einen Namen bezeichnet werden. Es muss aber die Klasse angegeben werden, von der dieses Objekt eine Ausprägung ist (in der Form : Point).
- Die Klassen müssen in dem Diagramm nicht mit abgebildet werden. Es kann aber manchmal angemessen sein, dies zu tun, also sozusagen „Bauplan“ und „Produkt“ gemeinsam darzustellen.
- Nicht alle Attributbelegungen eines Objekts müssen angegeben werden. Daher können durchaus auch Ausprägungen abstrakter Klassen auftauchen.

Objektdiagramme

Links

Ein Link beschreibt eine Beziehung zwischen Objekten. Er ist eine Ausprägung einer auf Klassenebene bestehenden Assoziation (auch Aggregation oder Komposition).

Objektdiagramme

Links

Ein Link beschreibt eine Beziehung zwischen Objekten. Er ist eine Ausprägung einer auf Klassenebene bestehenden Assoziation (auch Aggregation oder Komposition).

Bemerkungen:

- Links sind nicht mit Multiplizitäten beschriftet, denn ein Link repräsentiert genau eine Beziehung zwischen konkreten Partnern.

Objektdiagramme

Links

Ein Link beschreibt eine Beziehung zwischen Objekten. Er ist eine Ausprägung einer auf Klassenebene bestehenden Assoziation (auch Aggregation oder Komposition).

Bemerkungen:

- Links sind nicht mit Multiplizitäten beschriftet, denn ein Link repräsentiert genau eine Beziehung zwischen konkreten Partnern.
- Es ist jedoch darauf zu achten, dass insgesamt die Multiplizitätsbedingungen des Klassendiagramms eingehalten werden. Das heißt, die Anzahlen der Objekte, die miteinander in Beziehung stehen, müssen innerhalb der jeweiligen Schranken sein.

Objektdiagramme

Zurück zu altem Beispiel:

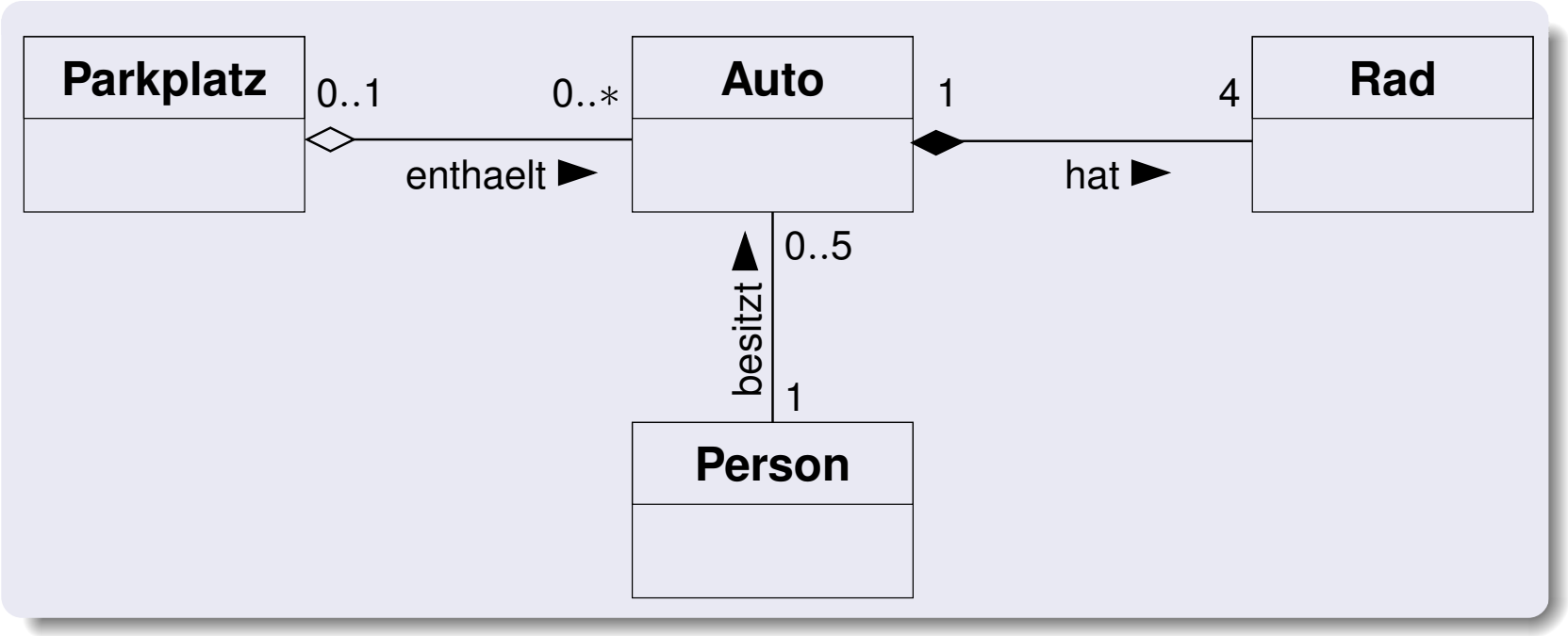
Parkplatz, Autos mit Besitzern, und nun auch noch Räder

Objektdiagramme

Zurück zu altem Beispiel:

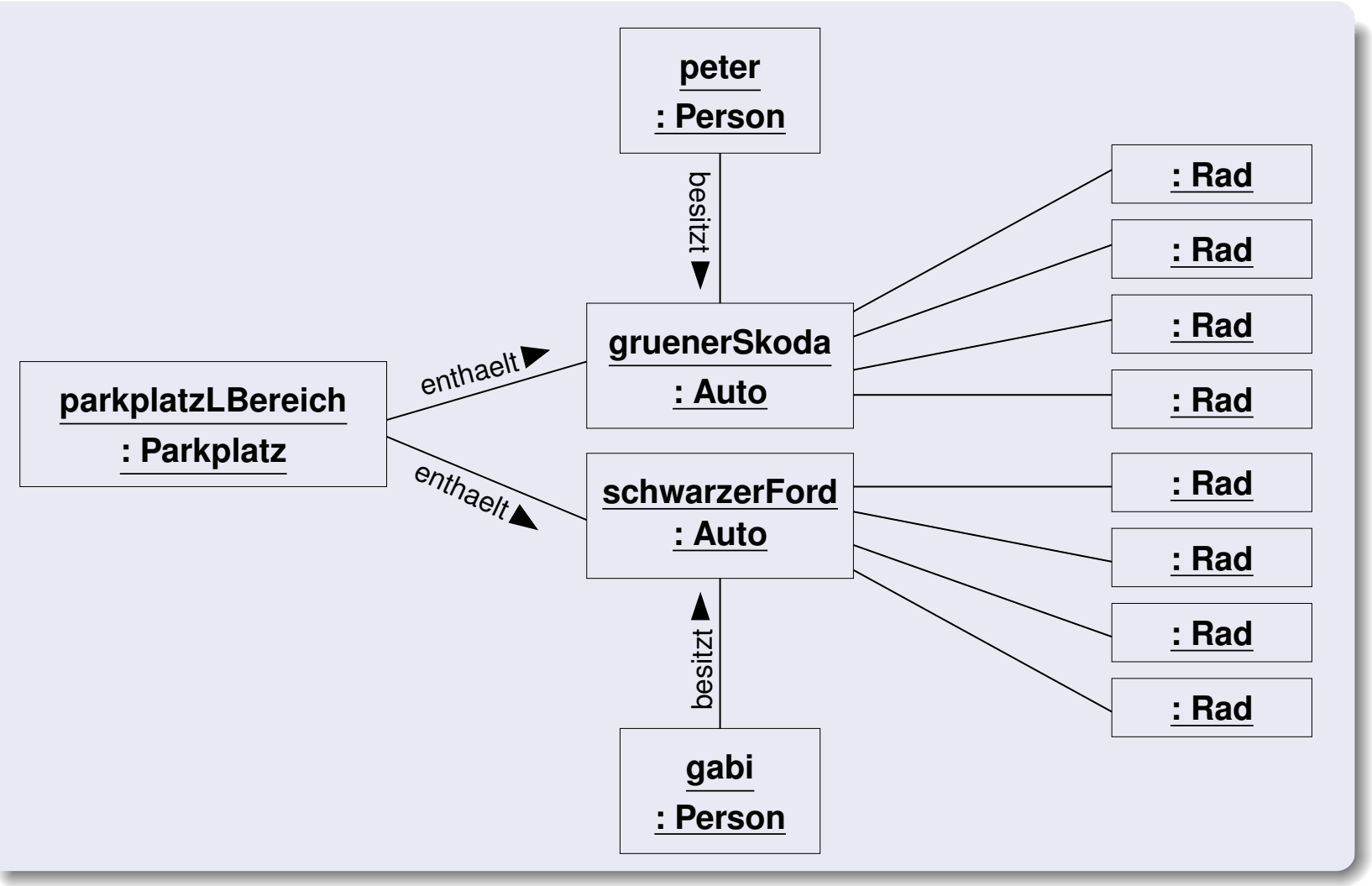
Parkplatz, Autos mit Besitzern, und nun auch noch Räder

Wir betrachten zunächst das (nun erweiterte) Klassendiagramm:



Objektdiagramme

Objektdiagramm passend zu diesem Klassendiagramm:



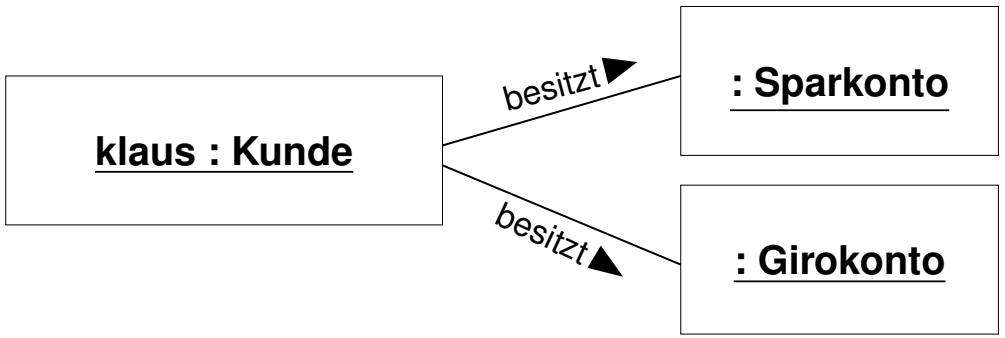
Objektdiagramme

Achtung: Beziehungen (Assoziationen, Aggregationen, Kompositionen) zwischen Klassen werden vererbt und müssen dann auch entsprechend im Objektdiagramm bei den Ausprägungen der Unterklassen auftauchen.

Objektdiagramme

Achtung: Beziehungen (Assoziationen, Aggregationen, Kompositionen) zwischen Klassen werden vererbt und müssen dann auch entsprechend im Objektdiagramm bei den Ausprägungen der Unterklassen auftauchen.

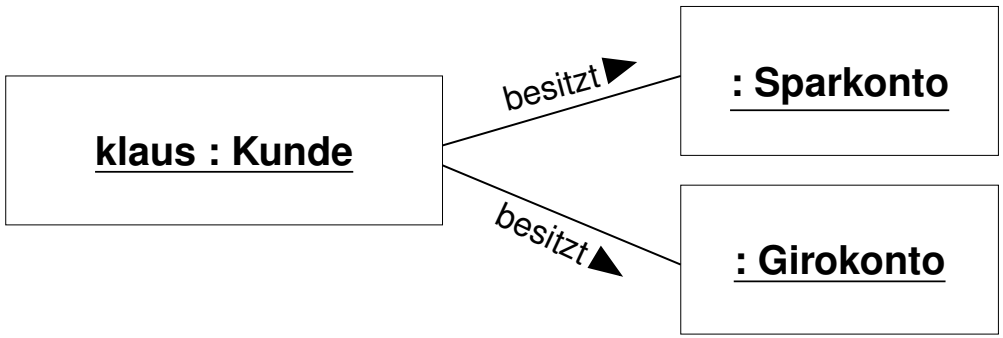
Beispiel:



Objektdiagramme

Achtung: Beziehungen (Assoziationen, Aggregationen, Kompositionen) zwischen Klassen werden vererbt und müssen dann auch entsprechend im Objektdiagramm bei den Ausprägungen der Unterklassen auftauchen.

Beispiel:

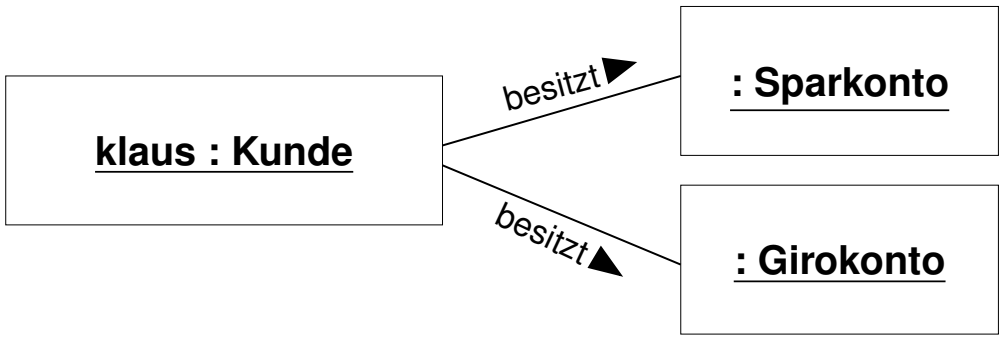


Auch Aggregations- und Kompositionssymbole dürfen in Objektdiagrammen auftauchen.

Objektdiagramme

Achtung: Beziehungen (Assoziationen, Aggregationen, Kompositionen) zwischen Klassen werden vererbt und müssen dann auch entsprechend im Objektdiagramm bei den Ausprägungen der Unterklassen auftauchen.

Beispiel:



Auch Aggregations- und Kompositionssymbole dürfen in Objektdiagrammen auftauchen.

Vererbungsbeziehungen dagegen tauchen in Objektdiagrammen nicht auf.

Aktivitätsdiagramme

Aktivitätsdiagramme

Wir betrachten nun Aktivitätsdiagramme (activity diagrams). Das sind UML-Diagramme, mit denen man Ablaufpläne, Reihenfolgen von Aktionen, parallele Aktionen, etc. modellieren kann.

Aktivitätsdiagramme

Wir betrachten nun Aktivitätsdiagramme (activity diagrams). Das sind UML-Diagramme, mit denen man Ablaufpläne, Reihenfolgen von Aktionen, parallele Aktionen, etc. modellieren kann.

Sie werden beispielsweise verwendet, um Geschäftsprozesse (auch Workflow-Prozesse) zu modellieren. Sie können ebenso eingesetzt werden, um Use Cases oder interne Systemprozesse zu beschreiben. Generell: wann immer man Einzelschritte hat, die sich auf gewisse typische Arten zu Gesamtabläufen zusammenfügen.

Aktivitätsdiagramme

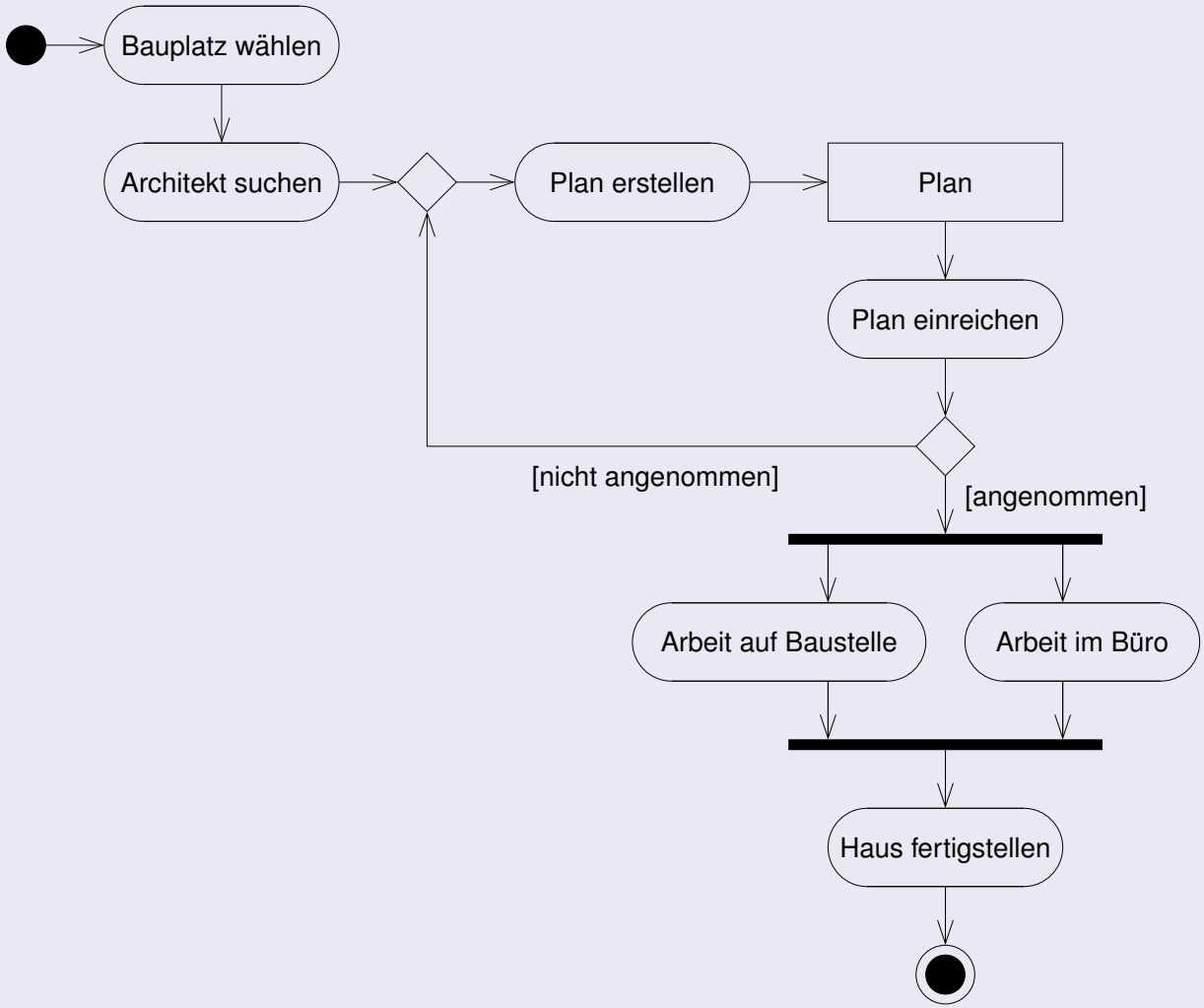
Wir betrachten nun Aktivitätsdiagramme (activity diagrams). Das sind UML-Diagramme, mit denen man Ablaufpläne, Reihenfolgen von Aktionen, parallele Aktionen, etc. modellieren kann.

Sie werden beispielsweise verwendet, um Geschäftsprozesse (auch Workflow-Prozesse) zu modellieren. Sie können ebenso eingesetzt werden, um Use Cases oder interne Systemprozesse zu beschreiben. Generell: wann immer man Einzelschritte hat, die sich auf gewisse typische Arten zu Gesamtabläufen zusammenfügen.

Aktivitätsdiagramme sind in vielen Aspekten ähnlich zu Petrinetzen. Im Unterschied zu Petrinetzen bieten sie zusätzliche Modellierungsmöglichkeiten, haben jedoch keine vollständige formale Semantik.

Aktivitätsdiagramme: Beispiel

Beispiel: Hausbau



Aktivitätsdiagramme: Elemente

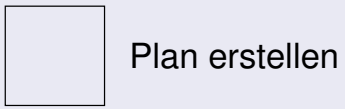
Wir vergleichen im Folgenden Aktivitätsdiagramme und ihre Bestandteile mit Petrinetzen (angelehnt an eine von Störrle aufgestellte Semantik für Teile von Aktivitätsdiagrammen).

Aktivitätsdiagramme: Elemente

Wir vergleichen im Folgenden Aktivitätsdiagramme und ihre Bestandteile mit Petrinetzen (angelehnt an eine von Störrle aufgestellte Semantik für Teile von Aktivitätsdiagrammen).

Aktionen

Eine Aktion im Aktivitätsdiagramm wird durch ein Rechteck mit abgerundeten Ecken dargestellt. Es entspricht einer Transition eines Petrinetzes.

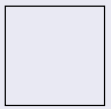


Aktivitätsdiagramme: Elemente

Wir vergleichen im Folgenden Aktivitätsdiagramme und ihre Bestandteile mit Petrinetzen (angelehnt an eine von Störrle aufgestellte Semantik für Teile von Aktivitätsdiagrammen).

Aktionen

Eine Aktion im Aktivitätsdiagramm wird durch ein Rechteck mit abgerundeten Ecken dargestellt. Es entspricht einer Transition eines Petrinetzes.



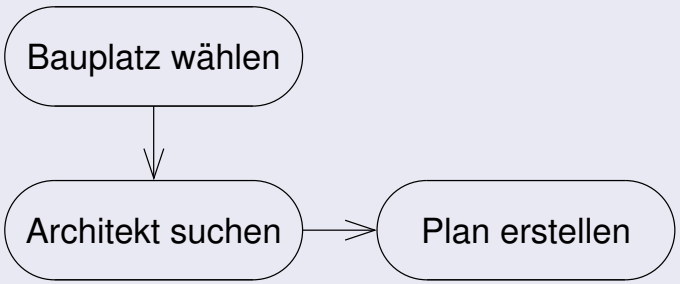
Plan erstellen

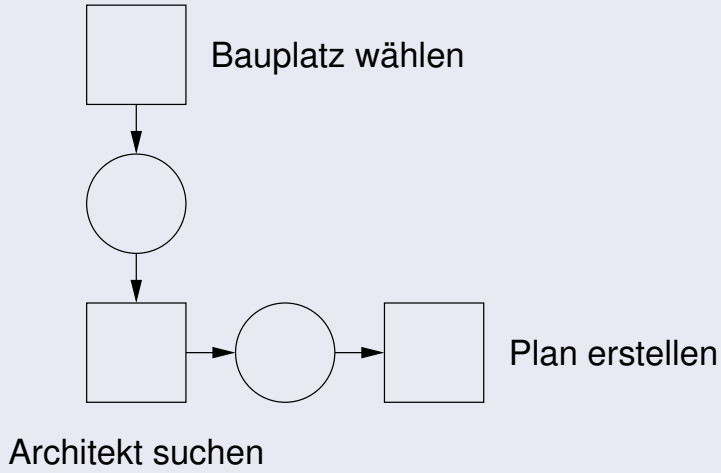
Intuition: Aktionen stehen für Tätigkeiten, mit Zeitaufwand.

Aktivitätsdiagramme: Elemente

Kontroll- oder Objektfluss

Der Kontroll- oder Objektfluss zwischen Aktionen (sowie Objektknoten) im Aktivitätsdiagramm, dargestellt durch die Kanten/Pfeile dazwischen, wird in dem entsprechenden Petrinetz mittels Hilfsstellen umgesetzt.

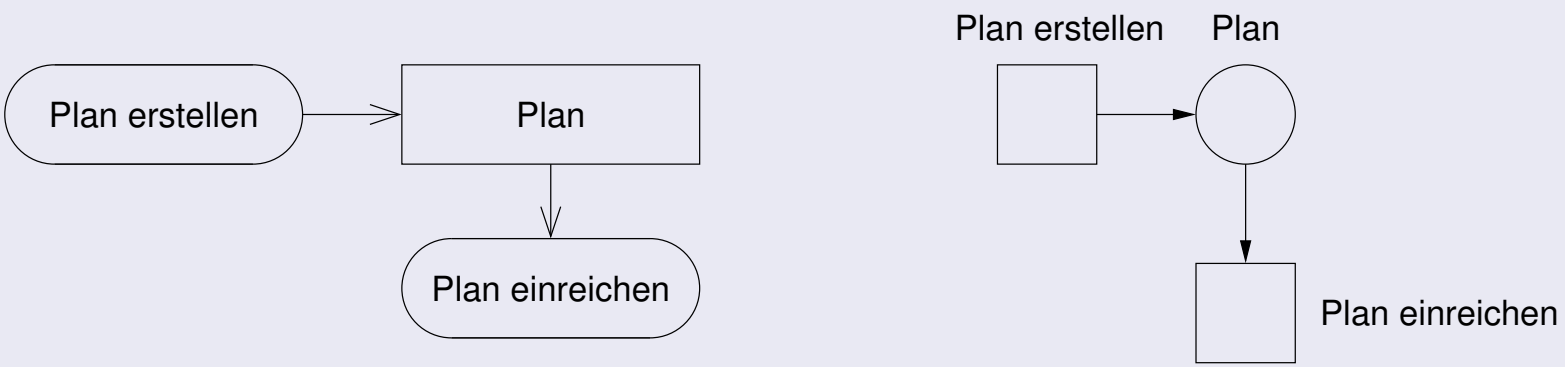




Aktivitätsdiagramme: Elemente

Objektknoten

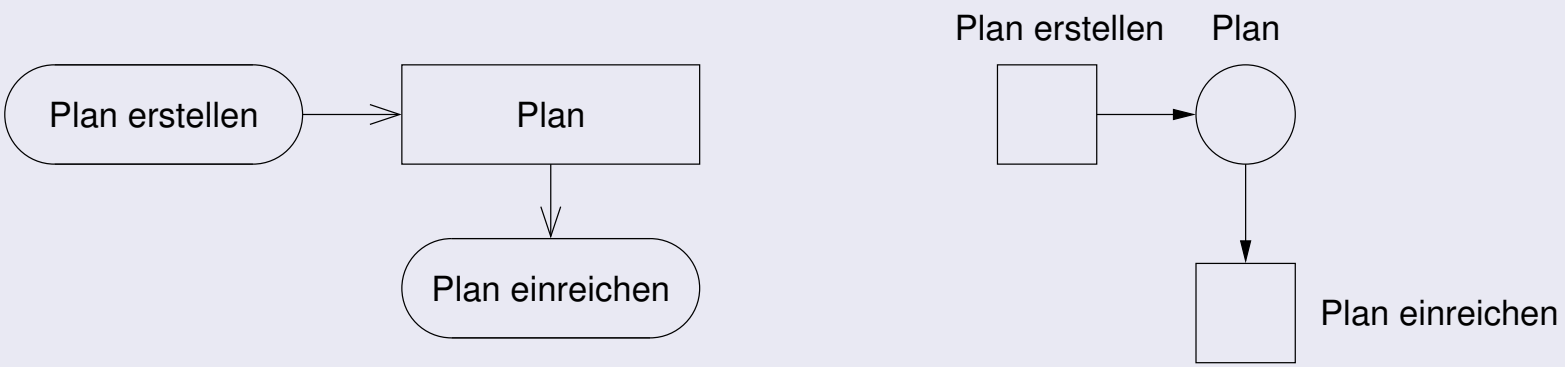
Objektknoten im Aktivitätsdiagramm beschreiben Speicher für die Ablage und Übergabe von konkreten Objekten. Sie entsprechen „normalen“ Stellen im Petrinetz, also solchen, die in irgendeiner Weise Ressourcen abbilden (und einen sinnvollen Namen haben).



Aktivitätsdiagramme: Elemente

Objektknoten

Objektknoten im Aktivitätsdiagramm beschreiben Speicher für die Ablage und Übergabe von konkreten Objekten. Sie entsprechen „normalen“ Stellen im Petrinetz, also solchen, die in irgendeiner Weise Ressourcen abbilden (und einen sinnvollen Namen haben).

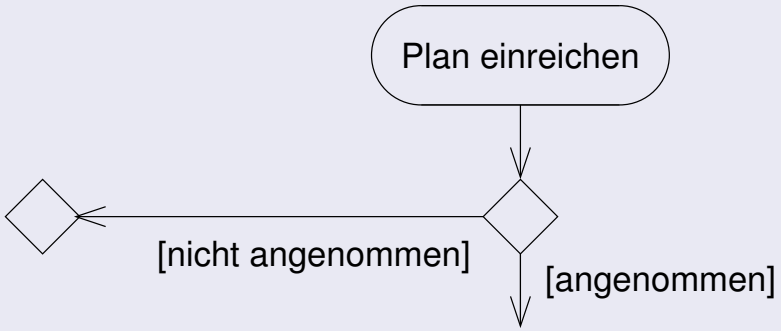


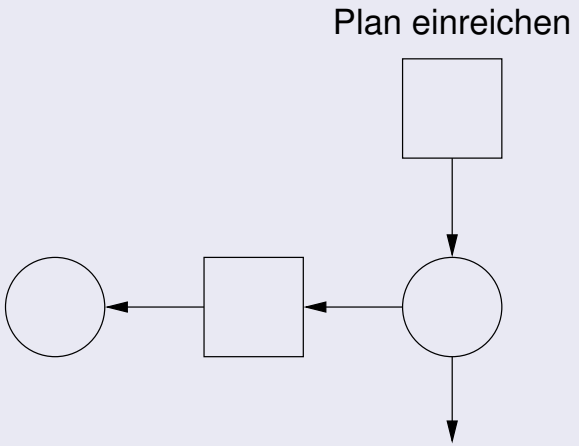
Bemerkung: Objektknoten sind bei Softwaremodellierung in der Regel mit einem Klassennamen beschriftet. Sie können dann (mehrere) Ausprägungen dieser Klasse aufnehmen.

Aktivitätsdiagramme: Elemente

Verzweigungsknoten

Verzweigungsknoten (decision nodes) im Aktivitätsdiagramm beschreiben eine Verzweigung des Kontroll- oder Objektflusses, wobei aus den alternativen Wegen jedes Mal genau einer ausgewählt wird. Sie werden im Petrinetz mittels Hilfsstellen umgesetzt, die sich in der Vorbedingung mehrerer Transitionen befinden.

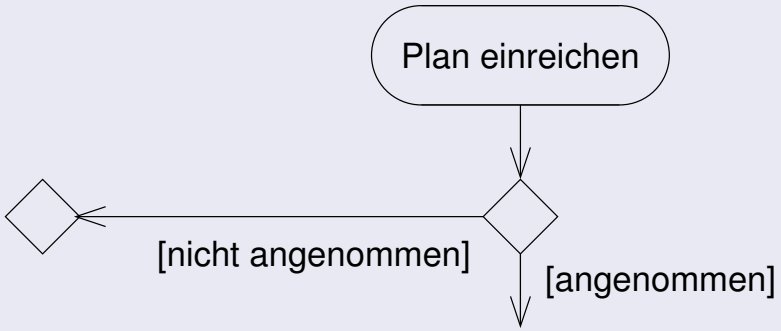


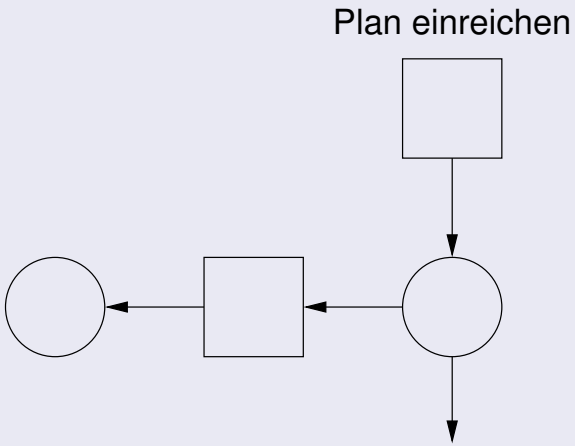


Aktivitätsdiagramme: Elemente

Verzweigungsknoten

Verzweigungsknoten (decision nodes) im Aktivitätsdiagramm beschreiben eine Verzweigung des Kontroll- oder Objektflusses, wobei aus den alternativen Wegen jedes Mal genau einer ausgewählt wird. Sie werden im Petrinetz mittels Hilfsstellen umgesetzt, die sich in der Vorbedingung mehrerer Transitionen befinden.





Bei Bedarf (etwa bei nachfolgenden Verzweigungs- oder Objektknoten) müssen Hilfstransitionen eingeführt werden.

Aktivitätsdiagramme: Elemente

Anmerkungen:

- Wesentlicher Grund für die Einführung von Hilfsstellen und Hilfstransitionen bei der Übersetzung in Petrinetze sind die strukturellen Erfordernisse für ein korrektes Petrinetz (nämlich abwechselndes Auftreten von Stellen und Transitionen).

Aktivitätsdiagramme: Elemente

Anmerkungen:

- Wesentlicher Grund für die Einführung von Hilfsstellen und Hilfstransitionen bei der Übersetzung in Petrinetze sind die strukturellen Erfordernisse für ein korrektes Petrinetz (nämlich abwechselndes Auftreten von Stellen und Transitionen).
- Der Kontroll- oder Objektfluss an Verzweigungsknoten wird durch Überwachungsbedingungen (Guards) gesteuert. Sie werden in eckigen Klammern an den ausgehenden Wegen notiert. (Ähnliche Bedingungen, an Transitionen, existieren auch in attributierten Petrinetzen.)

Aktivitätsdiagramme: Elemente

Anmerkungen:

- Wesentlicher Grund für die Einführung von Hilfsstellen und Hilfstransitionen bei der Übersetzung in Petrinetze sind die strukturellen Erfordernisse für ein korrektes Petrinetz (nämlich abwechselndes Auftreten von Stellen und Transitionen).
- Der Kontroll- oder Objektfluss an Verzweigungsknoten wird durch Überwachungsbedingungen (Guards) gesteuert. Sie werden in eckigen Klammern an den ausgehenden Wegen notiert. (Ähnliche Bedingungen, an Transitionen, existieren auch in attributierten Petrinetzen.)
- Die Bedingungen dürfen sich logisch nicht überlappen, und sollten insgesamt alle möglichen Fälle abdecken.

Aktivitätsdiagramme: Elemente

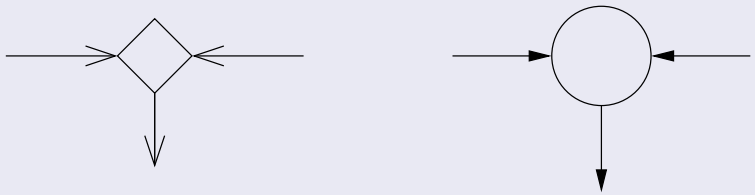
Anmerkungen:

- Wesentlicher Grund für die Einführung von Hilfsstellen und Hilfstransitionen bei der Übersetzung in Petrinetze sind die strukturellen Erfordernisse für ein korrektes Petrinetz (nämlich abwechselndes Auftreten von Stellen und Transitionen).
- Der Kontroll- oder Objektfluss an Verzweigungsknoten wird durch Überwachungsbedingungen (Guards) gesteuert. Sie werden in eckigen Klammern an den ausgehenden Wegen notiert. (Ähnliche Bedingungen, an Transitionen, existieren auch in attributierten Petrinetzen.)
- Die Bedingungen dürfen sich logisch nicht überlappen, und sollten insgesamt alle möglichen Fälle abdecken.
- Während Aktionen einen Zeitaufwand haben, werden Kontrollelemente wie Verzweigungsknoten (und weitere gleich gezeigte) als instantan durchlaufen angesehen.

Aktivitätsdiagramme: Elemente

Verbindungsknoten

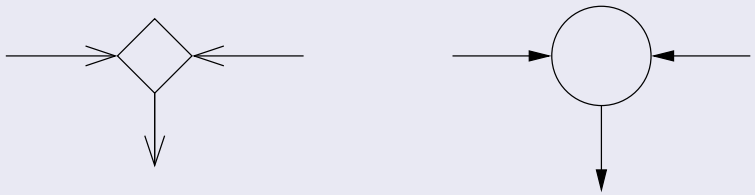
Es gibt auch Verbindungsknoten (merge nodes), die alternative Kontroll- oder Objektflüsse zusammenführen. Sie werden im entsprechenden Petrinetz mittels Hilfsstellen umgesetzt, die sich in der Nachbedingung mehrerer Transitionen (gegebenenfalls Hilfstransitionen) befinden.



Aktivitätsdiagramme: Elemente

Verbindungsknoten

Es gibt auch Verbindungsknoten (merge nodes), die alternative Kontroll- oder Objektflüsse zusammenführen. Sie werden im entsprechenden Petrinetz mittels Hilfsstellen umgesetzt, die sich in der Nachbedingung mehrerer Transitionen (gegebenenfalls Hilfstransitionen) befinden.



Es gibt auch Knoten (mit gleicher Darstellung), die sowohl Verbindungs- als auch Verzweigungsknoten sind, also mehrere eingehende und mehrere ausgehende Wege haben.

Aktivitätsdiagramme: Elemente

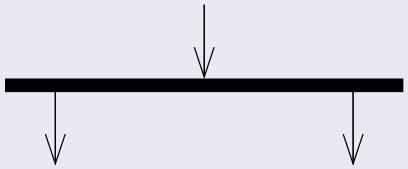
Gabelung (auch: Parallelisierungsknoten)

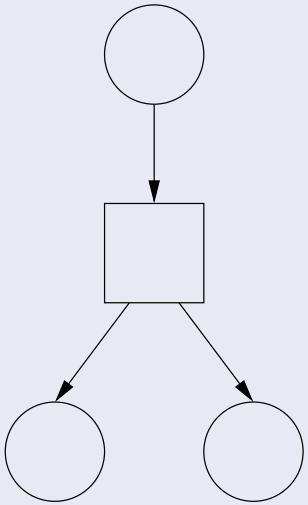
Eine Gabelung (fork node) im Aktivitätsdiagramm teilt einen Kontroll- oder Objektfluss in mehrere parallele Flüsse auf.

Aktivitätsdiagramme: Elemente

Gabelung (auch: Parallelisierungsknoten)

Eine Gabelung (fork node) im Aktivitätsdiagramm teilt einen Kontroll- oder Objektfluss in mehrere parallele Flüsse auf. Ihr entspricht im Petrinetz eine Transition mit mehreren Stellen (gegebenenfalls Hilfsstellen) in der Nachbedingung.





Aktivitätsdiagramme: Elemente

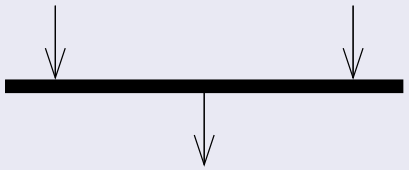
Vereinigung (auch: Synchronisationsknoten)

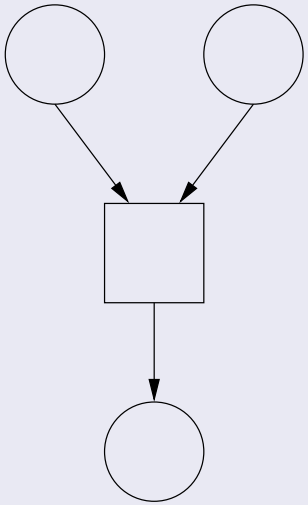
Dual dazu gibt es die Vereinigung (join node), die mehrere parallele Kontroll- oder Objektflüsse zusammenführt.

Aktivitätsdiagramme: Elemente

Vereinigung (auch: Synchronisationsknoten)

Dual dazu gibt es die Vereinigung (join node), die mehrere parallele Kontroll- oder Objektflüsse zusammenführt. Sie wird im entsprechenden Petrinetz durch eine Transition umgesetzt, die mehrere Stellen (ggfs. Hilfsstellen) in der Vorbedingung hat.

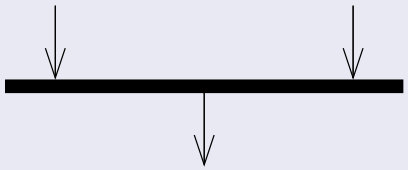


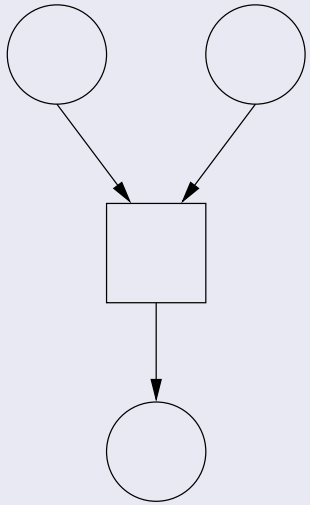


Aktivitätsdiagramme: Elemente

Vereinigung (auch: Synchronisationsknoten)

Dual dazu gibt es die Vereinigung (join node), die mehrere parallele Kontroll- oder Objektflüsse zusammenführt. Sie wird im entsprechenden Petrinetz durch eine Transition umgesetzt, die mehrere Stellen (ggfs. Hilfsstellen) in der Vorbedingung hat.



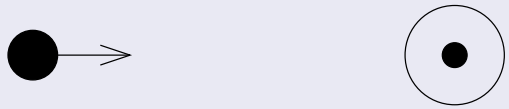


Wie Verbindungs- und Verzweigungsknoten kann man manchmal auch Vereinigungen und Gabelungen zu einem Knoten zusammenfassen.

Aktivitätsdiagramme: Elemente

Startknoten

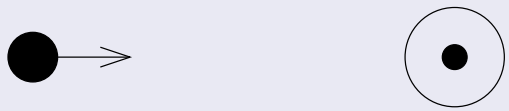
Ein Startknoten im Aktivitätsdiagramm entspricht einer initial markierten Stelle im Petrinetz. In Startknoten dürfen keine Kanten hineinführen.



Aktivitätsdiagramme: Elemente

Startknoten

Ein Startknoten im Aktivitätsdiagramm entspricht einer initial markierten Stelle im Petrinetz. In Startknoten dürfen keine Kanten hineinführen.



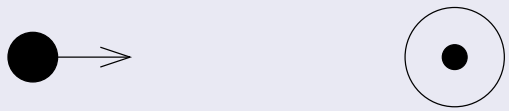
Anmerkungen:

- Im übersetzten Petrinetz wird die entsprechende Stelle initial mit so vielen Marken bestückt, wie es ausgehende Kanten vom Startknoten gibt.

Aktivitätsdiagramme: Elemente

Startknoten

Ein Startknoten im Aktivitätsdiagramm entspricht einer initial markierten Stelle im Petrinetz. In Startknoten dürfen keine Kanten hineinführen.



Anmerkungen:

- Im übersetzten Petrinetz wird die entsprechende Stelle initial mit so vielen Marken bestückt, wie es ausgehende Kanten vom Startknoten gibt.
- Diese Kanten führen generell nicht zu Objektknoten.

Aktivitätsdiagramme: Elemente

Aktivitätsende

Das Aktivitätsende signalisiert, dass alle Kontroll- und Objektflüsse beendet werden. Es gibt keine allgemeine Entsprechung in Petrinetzen.

Aktivitätsdiagramme: Elemente

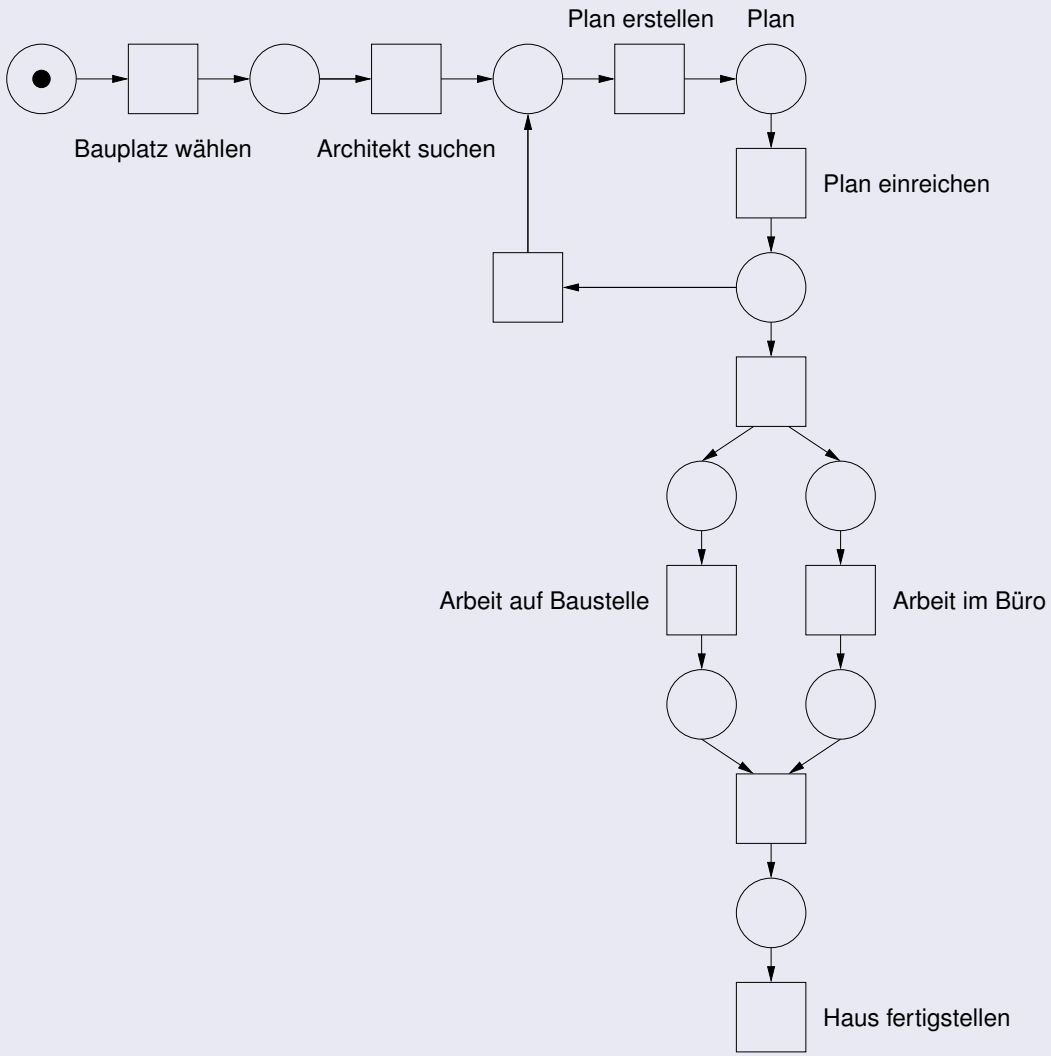
Aktivitätsende

Das Aktivitätsende signalisiert, dass alle Kontroll- und Objektflüsse beendet werden. Es gibt keine allgemeine Entsprechung in Petrinetzen.

Es gibt auch ein Symbol für das Flussende, das nur den in es hineinlaufenden Kontrollfluss beendet.

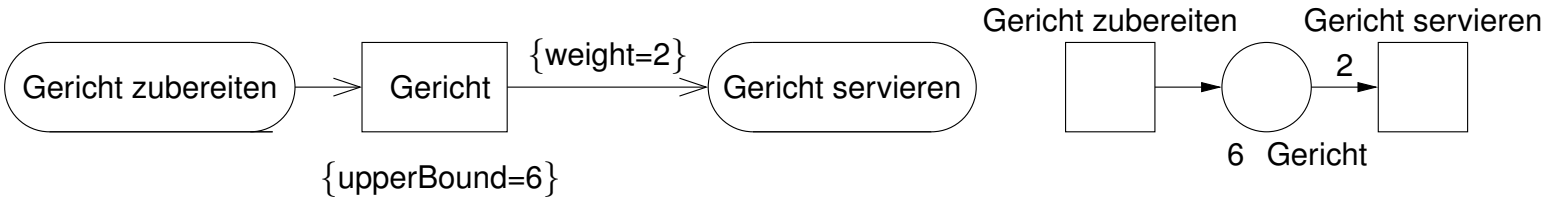
Aktivitätsdiagramme: Beispiel übersetzt

Übersetztes Beispiel, als Petrinetz:



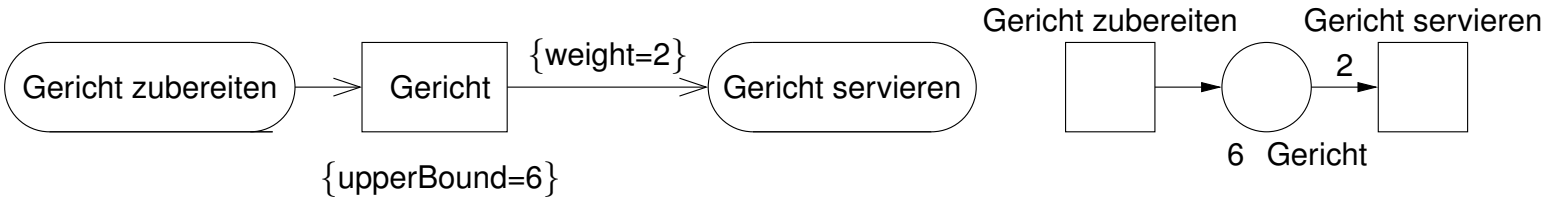
Aktivitätsdiagramme & Petrinetze

Aktivitätsdiagramme enthalten noch mehr Anleihen aus Petrinetzen. Beispielsweise können auch die Kapazität eines Objektknotens und das Gewicht eines Objektflusses spezifiziert werden.



Aktivitätsdiagramme & Petrinetze

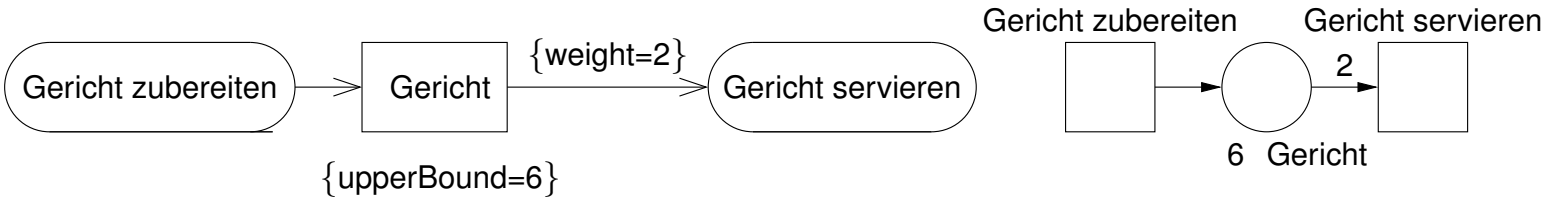
Aktivitätsdiagramme enthalten noch mehr Anleihen aus Petrinetzen. Beispielsweise können auch die Kapazität eines Objektknotens und das Gewicht eines Objektflusses spezifiziert werden.



Dabei beschreibt upperBound die Kapazität (maximal 6 Gerichte dürfen hier gleichzeitig fertig sein) und weight das Gewicht (immer 2 Gerichte werden hier gleichzeitig serviert).

Aktivitätsdiagramme & Petrinetze

Aktivitätsdiagramme enthalten noch mehr Anleihen aus Petrinetzen. Beispielsweise können auch die Kapazität eines Objektknotens und das Gewicht eines Objektflusses spezifiziert werden.



Dabei beschreibt upperBound die Kapazität (maximal 6 Gerichte dürfen hier gleichzeitig fertig sein) und weight das Gewicht (immer 2 Gerichte werden hier gleichzeitig serviert).

In Aktivitätsdiagrammen darf zusätzlich sogar noch spezifiziert werden, in welcher Reihenfolge die Objekte aus dem Objektknoten genommen werden (unordered, ordered, LIFO, FIFO).

Aktivitätsdiagramme & Petrinetze

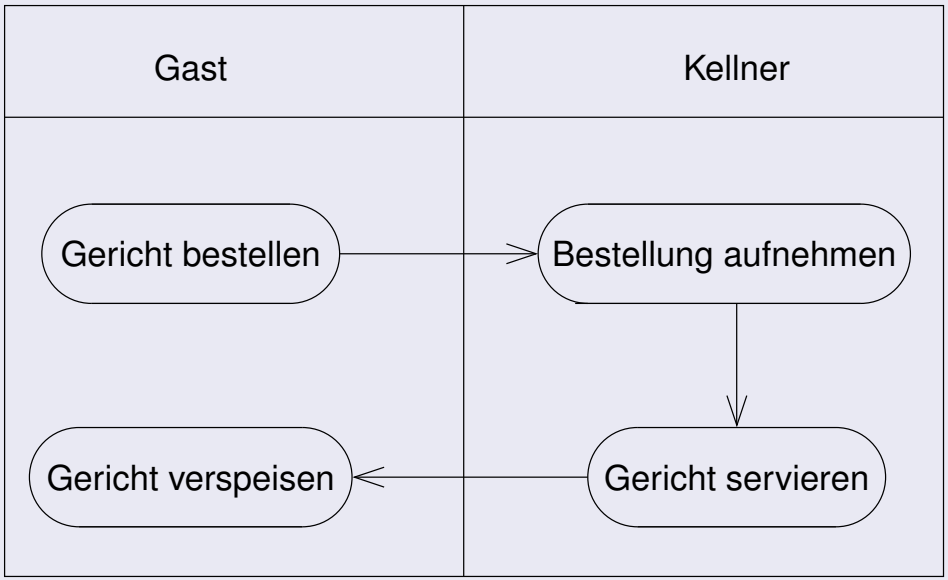
Vorsicht:

- Die Entsprechung zwischen Aktivitätsdiagrammen und Petrinetzen ist nicht immer ganz exakt. Nicht für alle Konzepte gibt es eine direkte Übersetzung.
- Das liegt teilweise auch daran, dass Aktivitätsdiagramme nur ein semi-formales Modelliergungsmittel sind und gar nicht alle Aspekte vollständig spezifiziert sind.

Aktivitätsdiagramme: Strukturierung

Aktivitätsbereiche (activity partitions / swimlanes)

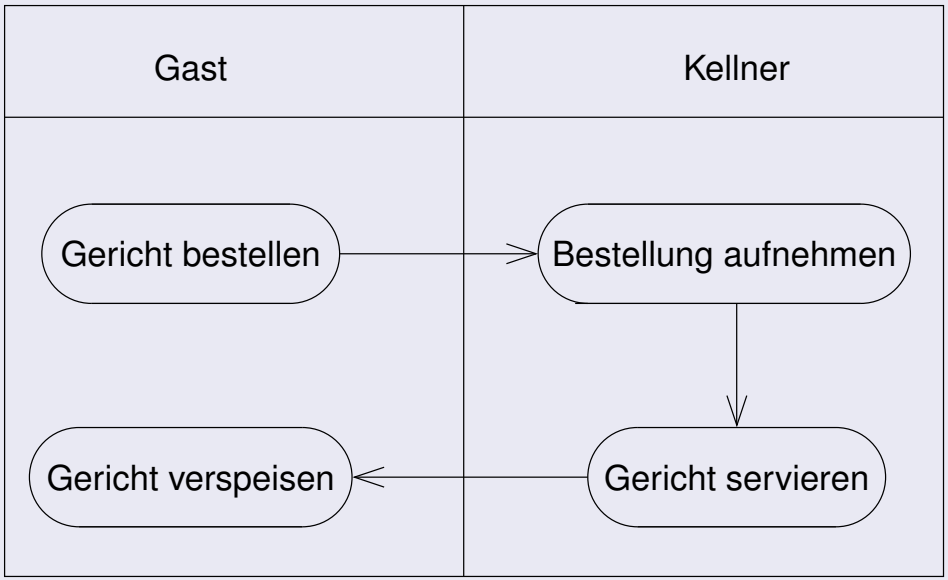
Zur Strukturierung können Elemente gruppiert werden. Dies dient etwa dazu, die Verantwortung für bestimmte Aktionen festzulegen oder räumliche Verteilung auszudrücken.



Aktivitätsdiagramme: Strukturierung

Aktivitätsbereiche (activity partitions / swimlanes)

Zur Strukturierung können Elemente gruppiert werden. Dies dient etwa dazu, die Verantwortung für bestimmte Aktionen festzulegen oder räumliche Verteilung auszudrücken.



Dabei können zum Beispiel Objektknoten auch auf einer Grenze zwischen Bereichen liegen, um eine Übergabe auszudrücken.

Aktivitätsdiagramme: Elemente (Fortsetzung)

Weitere Elemente von Aktivitätsdiagrammen:

- Pins: Parameter und Parametersätze für Aktionen
- Senden und Empfang von Signalen
- Kontrollstrukturen: Sprungmarken, Schleifenknoten, Bedingungsknoten
- Unterbrechungsbereiche (interruptible activity region) zur Behandlung von Ausnahmen (Exceptions)
- Expansionsbereiche (expansion region) zur wiederholten Ausführung von Aktivitäten für mehrere übergebene Objekte

Zustandsdiagramme

Zustandsdiagramme

UML-Zustandsdiagramme (state diagrams, auch state machine diagrams oder statecharts genannt), sind eng verwandt mit den bereits eingeführten Zustandsübergangsdiagrammen.

Zustandsdiagramme

UML-Zustandsdiagramme (state diagrams, auch state machine diagrams oder statecharts genannt), sind eng verwandt mit den bereits eingeführten Zustandsübergangsdiagrammen.

Sie werden eingesetzt, wenn bei der Modellierung der Fokus auf die Zustände und Zustandsübergänge des Systems gelegt werden soll.

Zustandsdiagramme

UML-Zustandsdiagramme (state diagrams, auch state machine diagrams oder statecharts genannt), sind eng verwandt mit den bereits eingeführten Zustandsübergangsdiagrammen.

Sie werden eingesetzt, wenn bei der Modellierung der Fokus auf die Zustände und Zustandsübergänge des Systems gelegt werden soll.

Im Gegensatz zu Aktivitätsdiagrammen werden auch weniger die Aktionen eines Systems beschrieben, sondern eher die Reaktionen eines Systems auf seine Umgebung.

Zustandsdiagramme

Anwendungen sind die Modellierung von:

- Protokollen, Komponenten verteilter Systeme
- Benutzungsoberflächen
- eingebetteten Systemen
- ...

Zustandsdiagramme

Zustandsdiagramme wurden 1987 von David Harel unter dem Namen Statecharts eingeführt.

Zustandsdiagramme

Zustandsdiagramme wurden 1987 von David Harel unter dem Namen Statecharts eingeführt.

Features, mit denen Zustandsdiagramme ausgestattet sind:

- Zustände und Zustandsübergänge

Zustandsdiagramme

Zustandsdiagramme wurden 1987 von David Harel unter dem Namen Statecharts eingeführt.

Features, mit denen Zustandsdiagramme ausgestattet sind:

- Zustände und Zustandsübergänge
- hierarchische Verfeinerung von Zuständen

Zustandsdiagramme

Zustandsdiagramme wurden 1987 von David Harel unter dem Namen Statecharts eingeführt.

Features, mit denen Zustandsdiagramme ausgestattet sind:

- Zustände und Zustandsübergänge
- hierarchische Verfeinerung von Zuständen
- „Parallelschalten“ durch Regionen

Zustandsdiagramme

Zustandsdiagramme wurden 1987 von David Harel unter dem Namen Statecharts eingeführt.

Features, mit denen Zustandsdiagramme ausgestattet sind:

- Zustände und Zustandsübergänge
- hierarchische Verfeinerung von Zuständen
- „Parallelschalten“ durch Regionen
- Historien, um sich früher besuchte Zustände zu merken und in diese zurückzukehren

Zustandsdiagramme

Zustandsdiagramme wurden 1987 von David Harel unter dem Namen Statecharts eingeführt.

Features, mit denen Zustandsdiagramme ausgestattet sind:

- Zustände und Zustandsübergänge
- hierarchische Verfeinerung von Zuständen
- „Parallelschalten“ durch Regionen
- Historien, um sich früher besuchte Zustände zu merken und in diese zurückzukehren
- Kommunikation über Effekte oder Flags

Zustandsdiagramme

Zustandsdiagramme wurden 1987 von David Harel unter dem Namen Statecharts eingeführt.

Features, mit denen Zustandsdiagramme ausgestattet sind:

- Zustände und Zustandsübergänge
- hierarchische Verfeinerung von Zuständen
- „Parallelschalten“ durch Regionen
- Historien, um sich früher besuchte Zustände zu merken und in diese zurückzukehren
- Kommunikation über Effekte oder Flags

Viele dieser Ausstattungsmerkmale dienen dazu, Diagramme mit vielen Zuständen und Übergängen übersichtlicher und kompakter zu gestalten.

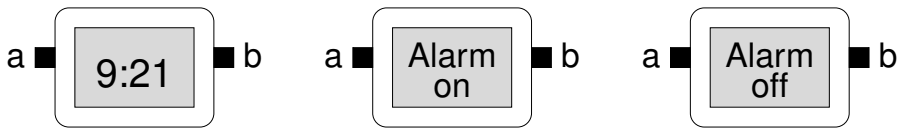
Zustandsdiagramme: Beispiel

Wir lernen Zustandsdiagramme am Beispiel der Modellierung einer Armbanduhr kennen (stark vereinfacht gegenüber einem Beispiel von Harel aus dem ursprünglichen wissenschaftlichen Artikel).

Zustandsdiagramme: Beispiel

Wir lernen Zustandsdiagramme am Beispiel der Modellierung einer Armbanduhr kennen (stark vereinfacht gegenüber einem Beispiel von Harel aus dem ursprünglichen wissenschaftlichen Artikel).

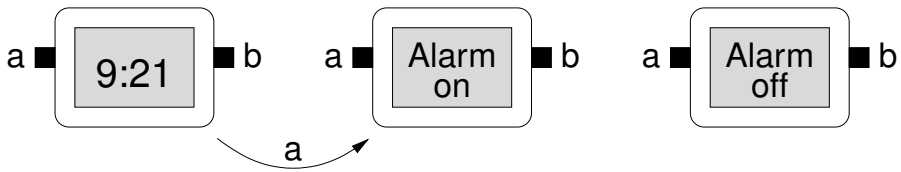
Die Armbanduhr hat zwei Knöpfe (a, b) und zwei Modi (Zeitanzeige, Alarmeinstellung).



Zustandsdiagramme: Beispiel

Wir lernen Zustandsdiagramme am Beispiel der Modellierung einer Armbanduhr kennen (stark vereinfacht gegenüber einem Beispiel von Harel aus dem ursprünglichen wissenschaftlichen Artikel).

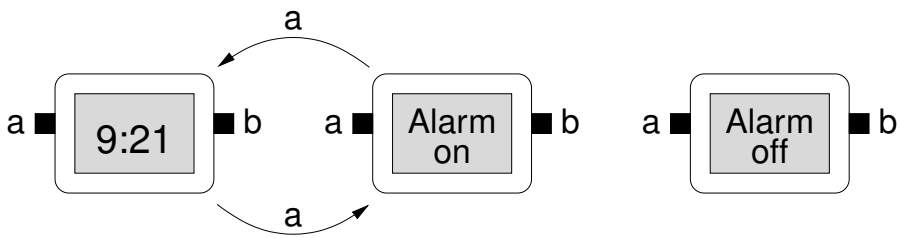
Die Armbanduhr hat zwei Knöpfe (a, b) und zwei Modi (Zeitanzeige, Alarmeinstellung). Zwischen den Modi wechselt man mit Hilfe von Knopf a.



Zustandsdiagramme: Beispiel

Wir lernen Zustandsdiagramme am Beispiel der Modellierung einer Armbanduhr kennen (stark vereinfacht gegenüber einem Beispiel von Harel aus dem ursprünglichen wissenschaftlichen Artikel).

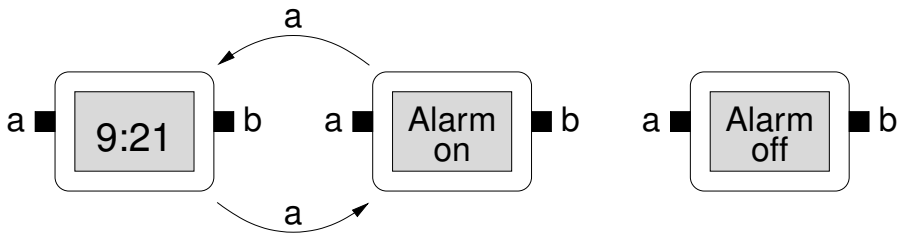
Die Armbanduhr hat zwei Knöpfe (a, b) und zwei Modi (Zeitanzeige, Alarmeinstellung). Zwischen den Modi wechselt man mit Hilfe von Knopf a.



Zustandsdiagramme: Beispiel

Wir lernen Zustandsdiagramme am Beispiel der Modellierung einer Armbanduhr kennen (stark vereinfacht gegenüber einem Beispiel von Harel aus dem ursprünglichen wissenschaftlichen Artikel).

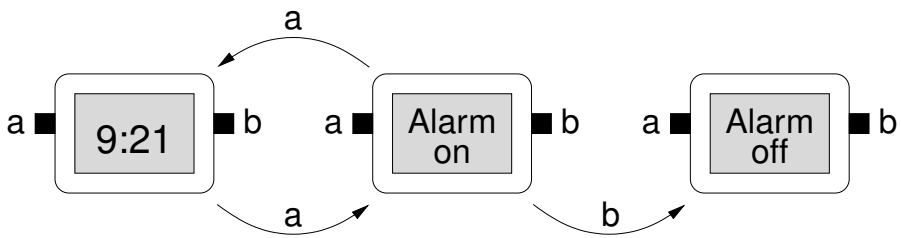
Die Armbanduhr hat zwei Knöpfe (a, b) und zwei Modi (Zeitanzeige, Alarmeinstellung). Zwischen den Modi wechselt man mit Hilfe von Knopf a. Der Alarm kann an (on) oder aus (off) sein.



Zustandsdiagramme: Beispiel

Wir lernen Zustandsdiagramme am Beispiel der Modellierung einer Armbanduhr kennen (stark vereinfacht gegenüber einem Beispiel von Harel aus dem ursprünglichen wissenschaftlichen Artikel).

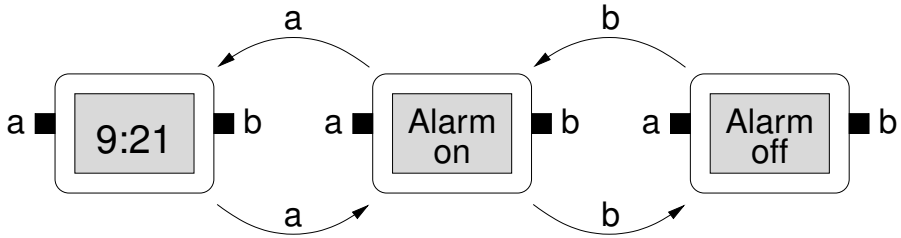
Die Armbanduhr hat zwei Knöpfe (a, b) und zwei Modi (Zeitanzeige, Alarmeinstellung). Zwischen den Modi wechselt man mit Hilfe von Knopf a. Der Alarm kann an (on) oder aus (off) sein. Zwischen den Alarmzuständen wechselt man mit Hilfe von Knopf b (im entsprechenden Modus).



Zustandsdiagramme: Beispiel

Wir lernen Zustandsdiagramme am Beispiel der Modellierung einer Armbanduhr kennen (stark vereinfacht gegenüber einem Beispiel von Harel aus dem ursprünglichen wissenschaftlichen Artikel).

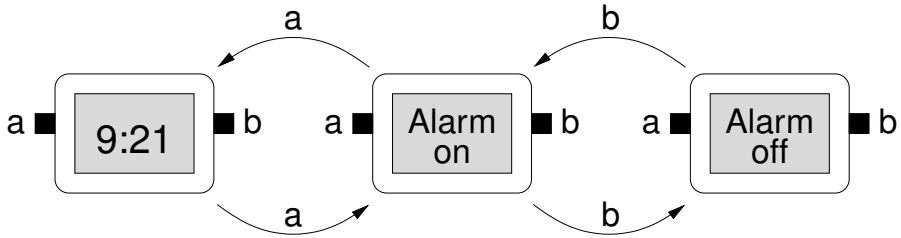
Die Armbanduhr hat zwei Knöpfe (a, b) und zwei Modi (Zeitanzeige, Alarmeinstellung). Zwischen den Modi wechselt man mit Hilfe von Knopf a. Der Alarm kann an (on) oder aus (off) sein. Zwischen den Alarmzuständen wechselt man mit Hilfe von Knopf b (im entsprechenden Modus).



Zustandsdiagramme: Beispiel

Wir lernen Zustandsdiagramme am Beispiel der Modellierung einer Armbanduhr kennen (stark vereinfacht gegenüber einem Beispiel von Harel aus dem ursprünglichen wissenschaftlichen Artikel).

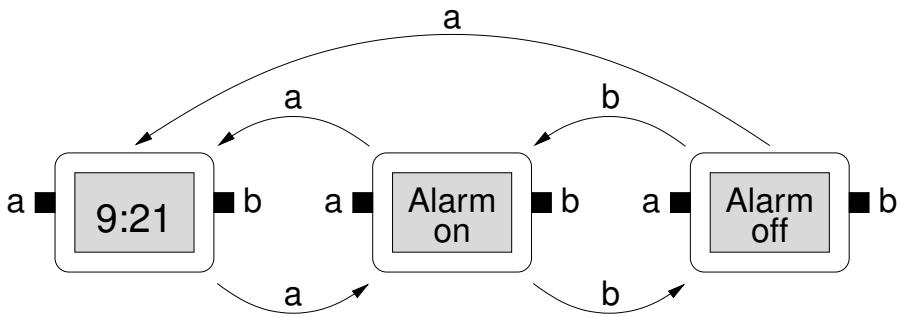
Die Armbanduhr hat zwei Knöpfe (a, b) und zwei Modi (Zeitanzeige, Alarmeinstellung). Zwischen den Modi wechselt man mit Hilfe von Knopf a. Der Alarm kann an (on) oder aus (off) sein. Zwischen den Alarmzuständen wechselt man mit Hilfe von Knopf b (im entsprechenden Modus). Wenn der Alarm an ist, erzeugt die Uhr zu jeder vollen Stunde einen Piepton.



Zustandsdiagramme: Beispiel

Wir lernen Zustandsdiagramme am Beispiel der Modellierung einer Armbanduhr kennen (stark vereinfacht gegenüber einem Beispiel von Harel aus dem ursprünglichen wissenschaftlichen Artikel).

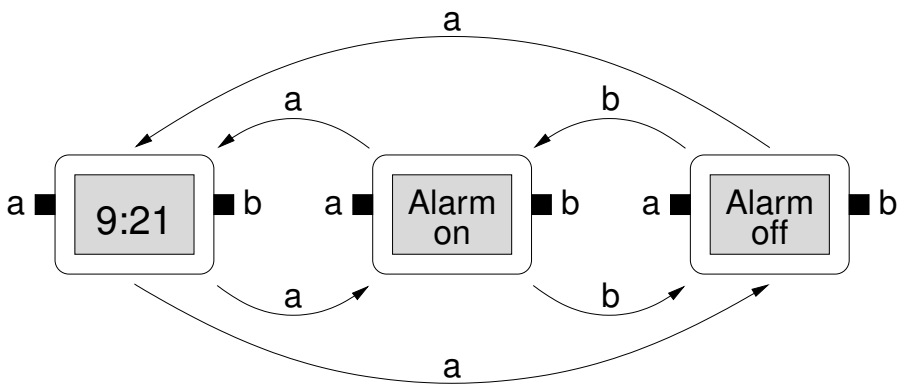
Die Armbanduhr hat zwei Knöpfe (a, b) und zwei Modi (Zeitanzeige, Alarmeinstellung). Zwischen den Modi wechselt man mit Hilfe von Knopf a. Der Alarm kann an (on) oder aus (off) sein. Zwischen den Alarmzuständen wechselt man mit Hilfe von Knopf b (im entsprechenden Modus). Wenn der Alarm an ist, erzeugt die Uhr zu jeder vollen Stunde einen Piepton.



Zustandsdiagramme: Beispiel

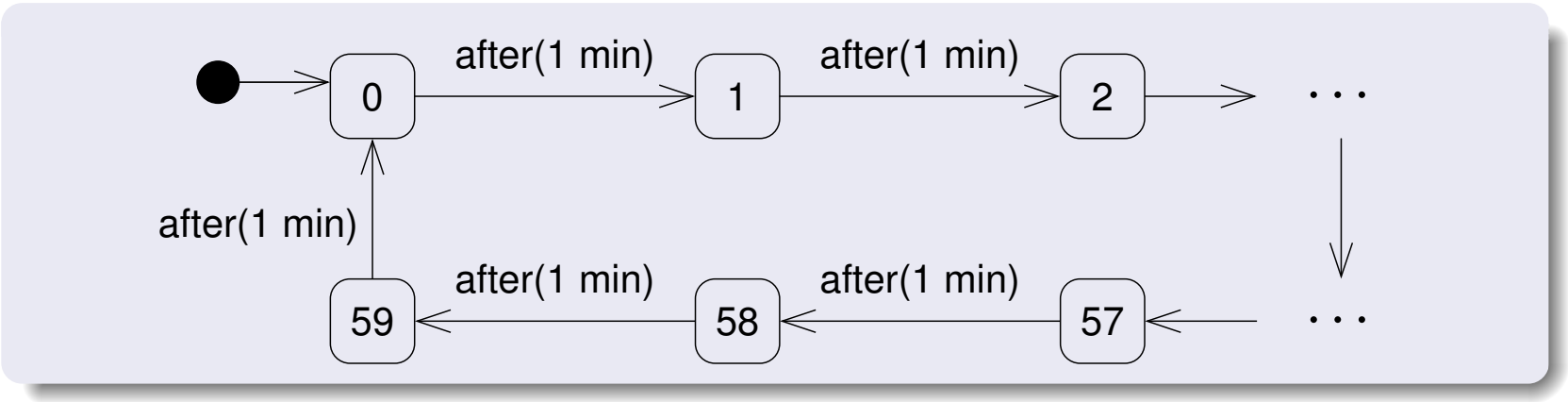
Wir lernen Zustandsdiagramme am Beispiel der Modellierung einer Armbanduhr kennen (stark vereinfacht gegenüber einem Beispiel von Harel aus dem ursprünglichen wissenschaftlichen Artikel).

Die Armbanduhr hat zwei Knöpfe (a, b) und zwei Modi (Zeitanzeige, Alarmeinstellung). Zwischen den Modi wechselt man mit Hilfe von Knopf a. Der Alarm kann an (on) oder aus (off) sein. Zwischen den Alarmzuständen wechselt man mit Hilfe von Knopf b (im entsprechenden Modus). Wenn der Alarm an ist, erzeugt die Uhr zu jeder vollen Stunde einen Piepton.



Zustandsdiagramme: Beispiel

Wir beginnen zunächst mit der Modellierung der Minutenanzeige.



Zustandsdiagramme: Elemente

Zustand

Ein Zustand wird durch ein Rechteck mit abgerundeten Ecken dargestellt.

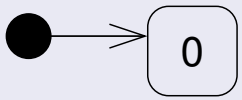
Zustandsdiagramme: Elemente

Zustand

Ein Zustand wird durch ein Rechteck mit abgerundeten Ecken dargestellt.

Startzustand

Der Startzustand wird ähnlich zum Startknoten bei Aktivitätsdiagrammen gekennzeichnet.

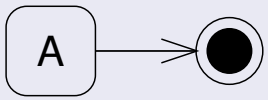


Es dürfen in den schwarzen ausgefüllten Kreis keine Kanten hineinführen, und nur genau eine heraus.

Zustandsdiagramme: Elemente

Endzustand

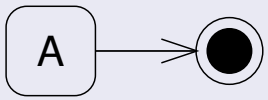
Endzustände werden wie das Aktivitätsende bei Aktivitätsdiagrammen gekennzeichnet.



Zustandsdiagramme: Elemente

Endzustand

Endzustände werden wie das Aktivitätsende bei Aktivitätsdiagrammen gekennzeichnet.



In Fällen, in denen man (wie in unserem Beispiel) ein System modelliert, das nicht terminieren soll, gibt es keinen Endzustand.

Zustandsdiagramme: Elemente

Transition (= Zustandsübergang)

Eine Transition ist eine Kante/Pfeil, beschriftet mit

Ereignis [Bedingung] / Effekt ,

wobei Bedingung und Effekt optional sind.

Zustandsdiagramme: Elemente

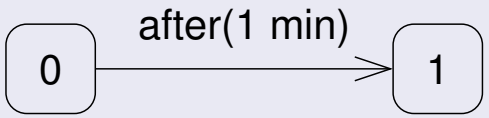
Transition (= Zustandsübergang)

Eine Transition ist eine Kante/Pfeil, beschriftet mit

Ereignis [Bedingung] / Effekt ,

wobei Bedingung und Effekt optional sind.

- Ereignis: Signal oder Nachricht, die die entsprechende Transition auslösen



Zustandsdiagramme: Elemente

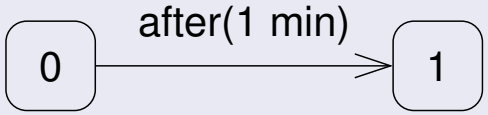
Transition (= Zustandsübergang)

Eine Transition ist eine Kante/Pfeil, beschriftet mit

Ereignis [Bedingung] / Effekt ,

wobei Bedingung und Effekt optional sind.

- Ereignis: Signal oder Nachricht, die die entsprechende Transition auslösen



- Bedingung: Überwachungsbedingung (auch Guard genannt)

Zustandsdiagramme: Elemente

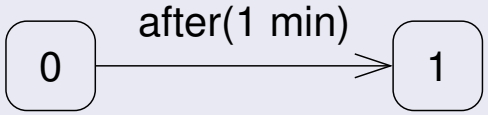
Transition (= Zustandsübergang)

Eine Transition ist eine Kante/Pfeil, beschriftet mit

Ereignis [Bedingung] / Effekt ,

wobei Bedingung und Effekt optional sind.

- Ereignis: Signal oder Nachricht, die die entsprechende Transition auslösen



- Bedingung: Überwachungsbedingung (auch Guard genannt)
- Effekt: Effekt, der durch die Transition ausgelöst wird

Zustandsdiagramme: Elemente

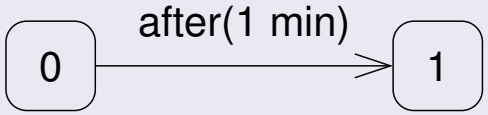
Transition (= Zustandsübergang)

Eine Transition ist eine Kante/Pfeil, beschriftet mit

Ereignis [Bedingung] / Effekt ,

wobei Bedingung und Effekt optional sind.

- Ereignis: Signal oder Nachricht, die die entsprechende Transition auslösen



- Bedingung: Überwachungsbedingung (auch Guard genannt)
- Effekt: Effekt, der durch die Transition ausgelöst wird

In unserem Beispiel (Minutenanzeige) gibt es nur Ereignisse, die eine Zeitspanne angeben, nach der die Transition ausgelöst wird.

Zustandsdiagramme: Elemente

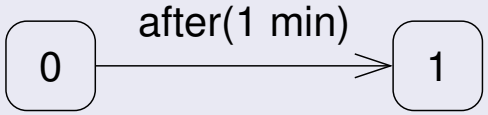
Transition (= Zustandsübergang)

Eine Transition ist eine Kante/Pfeil, beschriftet mit

Ereignis [Bedingung] / Effekt ,

wobei Bedingung und Effekt optional sind.

- Ereignis: Signal oder Nachricht, die die entsprechende Transition auslösen



- Bedingung: Überwachungsbedingung (auch Guard genannt)
- Effekt: Effekt, der durch die Transition ausgelöst wird

In unserem Beispiel (Minutenanzeige) gibt es nur Ereignisse, die eine Zeitspanne angeben, nach der die Transition ausgelöst wird. Allgemein gibt es auch andere Ereignisse, beispielsweise Methodenaufrufe oder durch Broadcast-Effekte.

Zustandsdiagramme: Aktionen

Neben Effekten, die durch Transitionen ausgelöst werden, können in einem Zustand selbst weitere Aktivitäten bei Eintritt, Verweilen oder Verlassen ausgelöst werden.

Zustandsdiagramme: Aktionen

Neben Effekten, die durch Transitionen ausgelöst werden, können in einem Zustand selbst weitere Aktivitäten bei Eintritt, Verweilen oder Verlassen ausgelöst werden.

Diese haben den gleichen Aufbau wie die Beschriftung einer Transition: Ereignis [Bedingung] / Effekt.

Zustandsdiagramme: Aktionen

Neben Effekten, die durch Transitionen ausgelöst werden, können in einem Zustand selbst weitere Aktivitäten bei Eintritt, Verweilen oder Verlassen ausgelöst werden.

Diese haben den gleichen Aufbau wie die Beschriftung einer Transition: Ereignis [Bedingung] / Effekt.

Dabei kann Ereignis dann unter anderem folgendes sein:

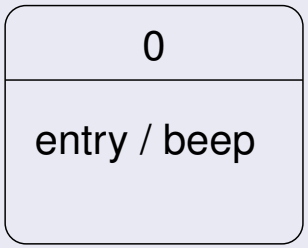
- entry: Der entsprechende Effekt wird bei Eintritt in den Zustand ausgelöst.
- do: Der Effekt ist eine Aktivität, die nach Betreten des Zustands ausgeführt wird und die spätestens dann endet, wenn der Zustand verlassen wird.
- exit: Der entsprechende Effekt wird bei Verlassen des Zustands ausgelöst.

Zustandsdiagramme: Beispiel

Beispiel:

Die Uhr soll zu jeder vollen Stunden ein Signal von sich geben. Daher wird die Aktivität beep bei Eintritt in den Zustand 0 ausgelöst.

Notation:



Zustandsdiagramme: Elemente

Was ist mit der Stundenanzeige?

Zustandsdiagramme: Elemente

Was ist mit der Stundenanzeige?

Diese soll parallel zur Minutenanzeige laufen, das heißt, die Uhr ist eigentlich immer in zwei Zuständen gleichzeitig: ein Zustand für die Stunden, der andere für die Minuten.

Zustandsdiagramme: Elemente

Was ist mit der Stundenanzeige?

Diese soll parallel zur Minutenanzeige laufen, das heißt, die Uhr ist eigentlich immer in zwei Zuständen gleichzeitig: ein Zustand für die Stunden, der andere für die Minuten.

Solch eine Situation wird durch Regionen modelliert.

Region

Regionen unterteilen ein Zustandsdiagramm in mindestens zwei Bereiche, die parallel zueinander ausgeführt werden.

Zustandsdiagramme: Elemente

Was ist mit der Stundenanzeige?

Diese soll parallel zur Minutenanzeige laufen, das heißt, die Uhr ist eigentlich immer in zwei Zuständen gleichzeitig: ein Zustand für die Stunden, der andere für die Minuten.

Solch eine Situation wird durch Regionen modelliert.

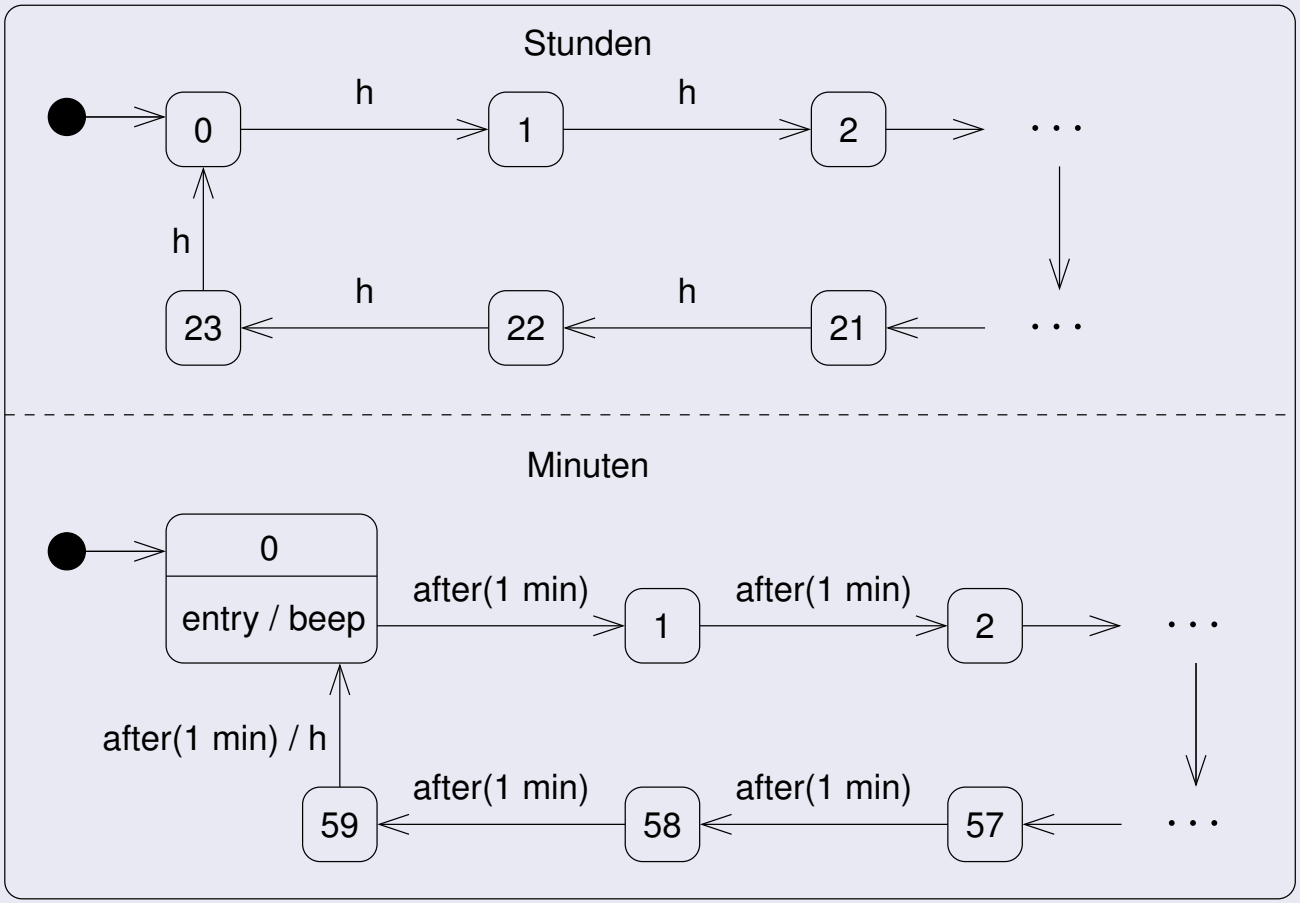
Region

Regionen unterteilen ein Zustandsdiagramm in mindestens zwei Bereiche, die parallel zueinander ausgeführt werden.

Im Beispiel erlaubt uns dies, insgesamt nur $24 + 60 = 84$ Zustände, statt $24 \cdot 60 = 1440$ Zustände, zu zeichnen.

Zustandsdiagramme: Beispiel

So sieht das modifizierte Zustandsdiagramm für die Uhr jetzt aus:



Zustandsdiagramme: Beispiel

Bemerkungen:

- Es gibt jetzt zwei Startzustände, die beide zu Beginn betreten werden (Uhrzeit: 0:00).

  (Allgemein: höchstens ein Startzustand pro Region.)

- Da Stunden- und Minutenanzeige nicht vollkommen unabhängig voneinander arbeiten, ist eine Synchronisation eingebaut: Die Transition in den Minutenzustand 0 löst einen Effekt h aus (h für „hour“).

Zustandsdiagramme: Beispiel

Bemerkungen:

- Es gibt jetzt zwei Startzustände, die beide zu Beginn betreten werden (Uhrzeit: 0:00).

  (Allgemein: höchstens ein Startzustand pro Region.)

- Da Stunden- und Minutenanzeige nicht vollkommen unabhängig voneinander arbeiten, ist eine Synchronisation eingebaut: Die Transition in den Minutenzustand 0 löst einen Effekt h aus (h für „hour“).

  Dieser Effekt dient dann als Ereignis, das den Übergang in den nächsten Stundenzustand verursacht.

  Die beiden Transitionen finden gleichzeitig statt.

Zustandsdiagramme: Hinweise

Weitere Bemerkungen (für uns nicht so relevant):

- Transitionen verbrauchen im Allgemeinen keine Zeit (im Gegensatz zum Aufenthalt in Zuständen).

Zustandsdiagramme: Hinweise

Weitere Bemerkungen (für uns nicht so relevant):

- Transitionen verbrauchen im Allgemeinen keine Zeit (im Gegensatz zum Aufenthalt in Zuständen).
- Neben Effekten/Ereignissen, die direkt ausgeführt werden, gibt es auch solche, die zunächst in einer Event Queue (Warteschlange) abgelegt werden. Diese wird dann schrittweise abgearbeitet, wobei die entsprechenden Ereignisse ausgelöst werden. Damit kann asynchrone Kommunikation beschrieben werden.

Zustandsdiagramme: Hinweise

Weitere Bemerkungen (für uns nicht so relevant):

- Transitionen verbrauchen im Allgemeinen keine Zeit (im Gegensatz zum Aufenthalt in Zuständen).
- Neben Effekten/Ereignissen, die direkt ausgeführt werden, gibt es auch solche, die zunächst in einer Event Queue (Warteschlange) abgelegt werden. Diese wird dann schrittweise abgearbeitet, wobei die entsprechenden Ereignisse ausgelöst werden. Damit kann asynchrone Kommunikation beschrieben werden.
- Effekte können Broadcasts (= Ausstrahlungen) sein, die überall im Zustandsdiagramm sichtbar sind, oder können alternativ direkte Kommunikation bedeuten (beispielsweise Methodenaufrufe).
  (Die ursprüngliche Statecharts-Semantik von Harel verwendete nur Broadcasts.)

Zustandsdiagramme: Elemente

Nun soll die Möglichkeit hinzugefügt werden, das Stunden-Alarmsignal aus- und wieder einzuschalten.

Zustandsdiagramme: Elemente

Nun soll die Möglichkeit hinzugefügt werden, das Stunden-Alarmsignal aus- und wieder einzuschalten.

Dazu führen wir zunächst weitere Zustände (Alarm on, off) ein, zu denen man durch Drücken von a gelangt (und zwischen denen man mit b wechselt).

Zustandsdiagramme: Elemente

Nun soll die Möglichkeit hinzugefügt werden, das Stunden-Alarmsignal aus- und wieder einzuschalten.

Dazu führen wir zunächst weitere Zustände (Alarm on, off) ein, zu denen man durch Drücken von a gelangt (und zwischen denen man mit b wechselt).

Problem: Wir bräuchten ziemlich viele mit a beschriftete Transitionen, die von den Stunden-/Minutenzuständen ausgehen!

Zustandsdiagramme: Elemente

Nun soll die Möglichkeit hinzugefügt werden, das Stunden-Alarmsignal aus- und wieder einzuschalten.

Dazu führen wir zunächst weitere Zustände (Alarm on, off) ein, zu denen man durch Drücken von a gelangt (und zwischen denen man mit b wechselt).

Problem: Wir bräuchten ziemlich viele mit a beschriftete Transitionen, die von den Stunden-/Minutenzuständen ausgehen!

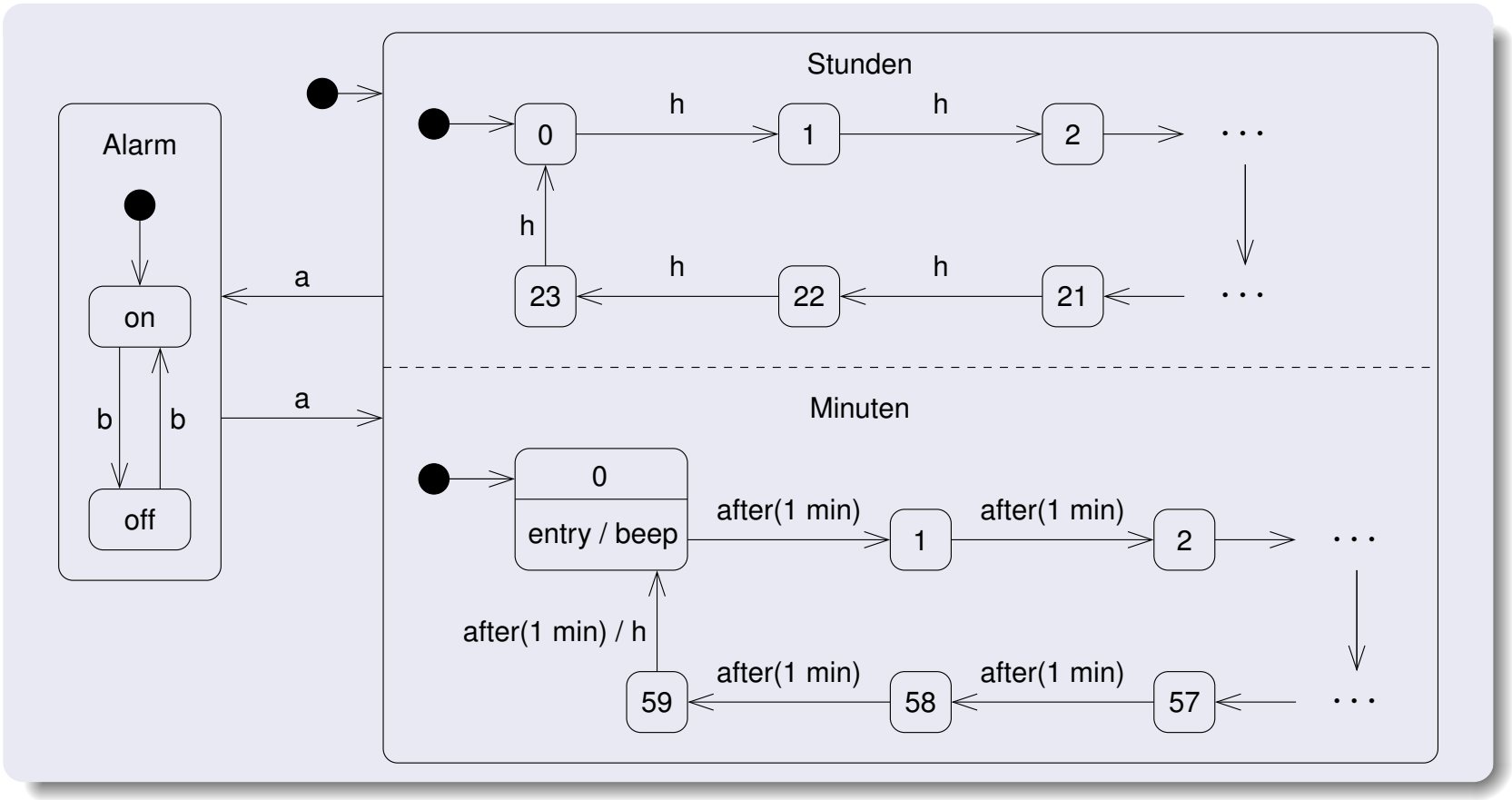
Dafür bieten Zustandsdiagramme folgende Lösung:

Zusammengesetzte Zustände

Zusammengesetzte Zustände dienen dazu, Hierarchien von Zuständen zu modellieren und dadurch ein- und ausgehende Transitionen zusammenzufassen.

Zustandsdiagramme: Beispiel

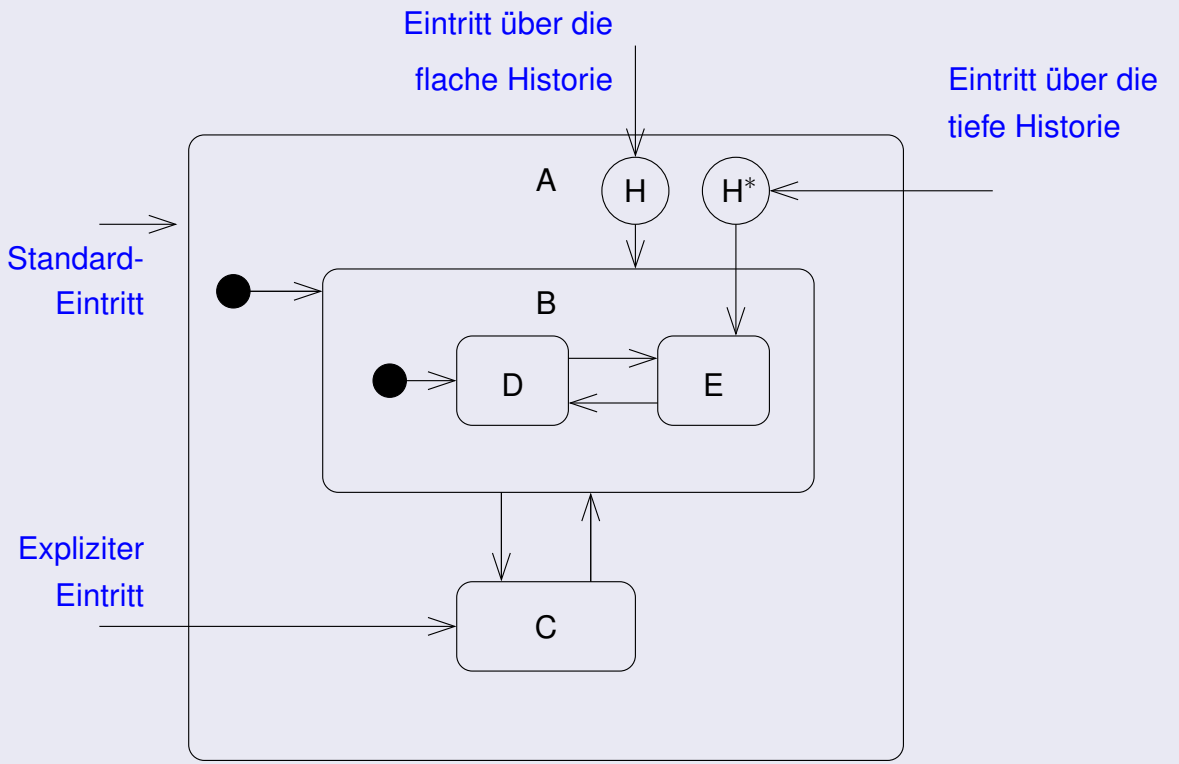
Damit ergibt sich folgendes Zustandsdiagramm:



Zustandsdiagramme: Eintrittsmöglichkeiten

Um möglichst viel Flexibilität bei der Modellierung zu haben, werden für zusammengesetzte Zustände verschiedene Eintritts- und Austrittsmöglichkeiten bereitgestellt.

Eintrittsmöglichkeiten in einen Zustand (grafisch)



Zustandsdiagramme: Eintrittsmöglichkeiten

Beschreibung der Eintrittsmöglichkeiten:

- Standard-Eintritt: Dabei wird der Startzustand des zusammengesetzten Zustands angesprungen.

  (Fortsetzung bei B, was schließlich zu einer Fortsetzung bei D führt.)

Zustandsdiagramme: Eintrittsmöglichkeiten

Beschreibung der Eintrittsmöglichkeiten:

- Standard-Eintritt: Dabei wird der Startzustand des zusammengesetzten Zustands angesprungen.

  (Fortsetzung bei B, was schließlich zu einer Fortsetzung bei D führt.)

- Expliziter Eintritt: Es wird bei dem explizit angegebenen Folgezustand fortgesetzt.

  (Fortsetzung bei C.)

Zustandsdiagramme: Eintrittsmöglichkeiten

Beschreibung der Eintrittsmöglichkeiten (Fortsetzung):

- Eintritt über die tiefe Historie: Wurde der zusammengesetzte Zustand bereits früher besucht, so wird der letzte vor dem Verlassen des Gesamtzustands aktive Unterzustand der tiefstmöglichen Ebene betreten.

  (Falls also der zusammengesetzte Zustand A das letzte Mal von D aus verlassen wurde, so wird jetzt wieder bei D fortgesetzt.)

Zustandsdiagramme: Eintrittsmöglichkeiten

Beschreibung der Eintrittsmöglichkeiten (Fortsetzung):

- Eintritt über die tiefe Historie: Wurde der zusammengesetzte Zustand bereits früher besucht, so wird der letzte vor dem Verlassen des Gesamtzustands aktive Unterzustand der tiefstmöglichen Ebene betreten.

  (Falls also der zusammengesetzte Zustand A das letzte Mal von D aus verlassen wurde, so wird jetzt wieder bei D fortgesetzt.)

  Falls man noch niemals zuvor diesen zusammengesetzten Zustand betreten hat, so wird der Zustand betreten, der mit der Kante gekennzeichnet ist, die von dem H^*-Knoten ausgeht.

Zustandsdiagramme: Eintrittsmöglichkeiten

Beschreibung der Eintrittsmöglichkeiten (Fortsetzung):

- Eintritt über die flache Historie: Wurde der zusammengesetzte Zustand bereits früher besucht, so wird der letzte vor dem Verlassen des Gesamtzustands aktive Unterzustand der obersten Ebene betreten.

  (Falls also der zusammengesetzte Zustand A das letzte Mal von E aus verlassen wurde, so wird jetzt bei B fortgesetzt, was letztendlich zu einer Fortsetzung bei D führt.)

Zustandsdiagramme: Eintrittsmöglichkeiten

Beschreibung der Eintrittsmöglichkeiten (Fortsetzung):

- Eintritt über die flache Historie: Wurde der zusammengesetzte Zustand bereits früher besucht, so wird der letzte vor dem Verlassen des Gesamtzustands aktive Unterzustand der obersten Ebene betreten.

  (Falls also der zusammengesetzte Zustand A das letzte Mal von E aus verlassen wurde, so wird jetzt bei B fortgesetzt, was letztendlich zu einer Fortsetzung bei D führt.)

  Falls man noch niemals zuvor diesen zusammengesetzten Zustand betreten hat, so wird analog wie bei der tiefen Historie verfahren.

Zustandsdiagramme: Eintrittsmöglichkeiten

Beschreibung der Eintrittsmöglichkeiten (Fortsetzung):

- Eintritt über die flache Historie: Wurde der zusammengesetzte Zustand bereits früher besucht, so wird der letzte vor dem Verlassen des Gesamtzustands aktive Unterzustand der obersten Ebene betreten.

  (Falls also der zusammengesetzte Zustand A das letzte Mal von E aus verlassen wurde, so wird jetzt bei B fortgesetzt, was letztendlich zu einer Fortsetzung bei D führt.)

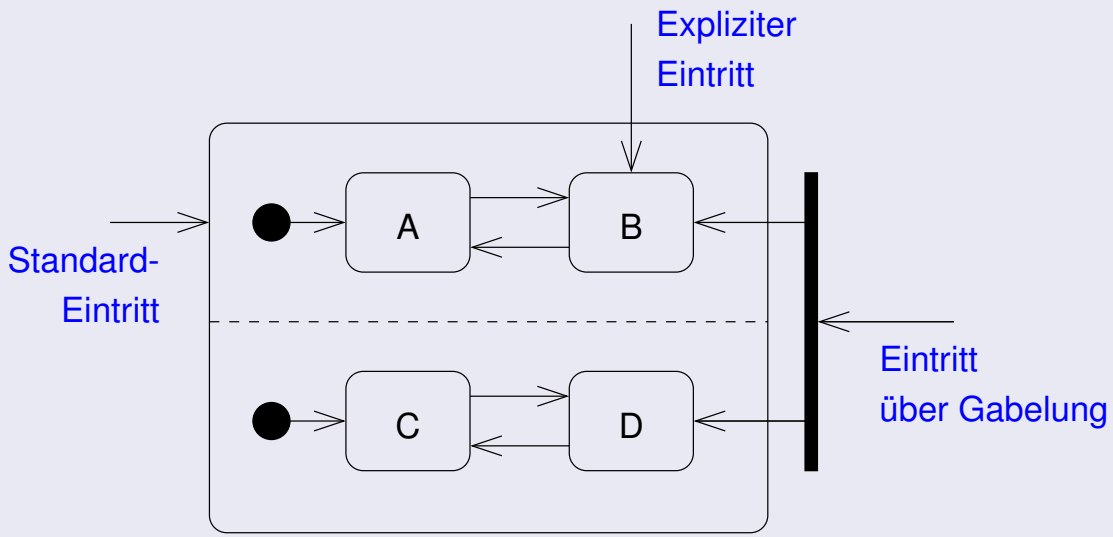
  Falls man noch niemals zuvor diesen zusammengesetzten Zustand betreten hat, so wird analog wie bei der tiefen Historie verfahren.

Außerdem: Eintritt über einen Eintrittspunkt (wird hier nicht behandelt).

Zustandsdiagramme: Eintrittsmöglichkeiten

Wenn ein zusammengesetzter Zustand in mehrere Regionen unterteilt ist, so ergeben sich bei den Eintrittsmöglichkeiten einige Besonderheiten.

Eintrittsmöglichkeiten bei Regionen (grafisch)



Es gäbe zusätzlich auch wieder die Fälle zum Eintritt über flache oder tiefe Historie zu diskutieren, darauf verzichten wir hier jedoch.

Zustandsdiagramme: Eintrittsmöglichkeiten

Beschreibung der Eintrittsmöglichkeiten bei Regionen:

- Standard-Eintritt: Dabei werden die jeweiligen Startzustände der Regionen angesprungen.

  (Fortsetzung bei A und C.)

Zustandsdiagramme: Eintrittsmöglichkeiten

Beschreibung der Eintrittsmöglichkeiten bei Regionen:

- Standard-Eintritt: Dabei werden die jeweiligen Startzustände der Regionen angesprungen.

  (Fortsetzung bei A und C.)

- Expliziter Eintritt: Ein Zustand einer Region wird direkt angesprungen. In der anderen Region wird beim Startzustand fortgesetzt.

  (Fortsetzung bei B und C.)

Zustandsdiagramme: Eintrittsmöglichkeiten

Beschreibung der Eintrittsmöglichkeiten bei Regionen:

- Standard-Eintritt: Dabei werden die jeweiligen Startzustände der Regionen angesprungen.

  (Fortsetzung bei A und C.)

- Expliziter Eintritt: Ein Zustand einer Region wird direkt angesprungen. In der anderen Region wird beim Startzustand fortgesetzt.

  (Fortsetzung bei B und C.)

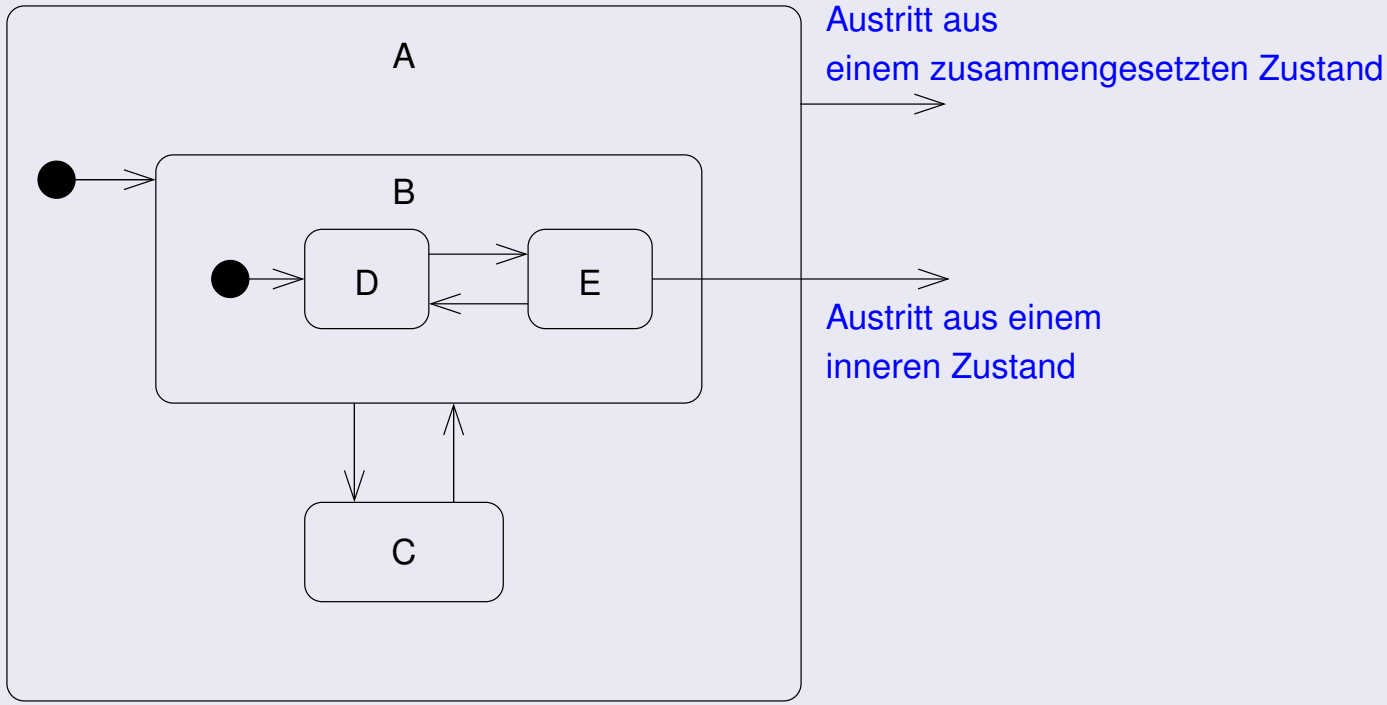
- Eintritt über Gabelung: Die beiden anzuspringenden Zustände in den Regionen werden ähnlich zur Gabelung bei Aktivitätsdiagrammen gekennzeichnet.

  (Fortsetzung bei B und D.)

Zustandsdiagramme: Austrittsmöglichkeiten

Auch für den Austritt aus zusammengesetzten Zuständen gibt es verschiedene Möglichkeiten.

Austrittsmöglichkeiten aus einem Zustand (grafisch)



Zustandsdiagramme: Austrittsmöglichkeiten

Beschreibung der Austrittsmöglichkeiten:

- Austritt aus einem zusammengesetzten Zustand: Sobald das mit der Transition assoziierte Ereignis stattfindet, wird jeder beliebige Unterzustand verlassen.

Zustandsdiagramme: Austrittsmöglichkeiten

Beschreibung der Austrittsmöglichkeiten:

- Austritt aus einem zusammengesetzten Zustand: Sobald das mit der Transition assoziierte Ereignis stattfindet, wird jeder beliebige Unterzustand verlassen.
- Austritt aus einem inneren Zustand: Die Transition wird nur genommen, wenn man sich gerade im entsprechenden Unterzustand (hier: Zustand E) befindet, und das entsprechende Ereignis stattfindet.

Zustandsdiagramme: Austrittsmöglichkeiten

Beschreibung der Austrittsmöglichkeiten:

- Austritt aus einem zusammengesetzten Zustand: Sobald das mit der Transition assoziierte Ereignis stattfindet, wird jeder beliebige Unterzustand verlassen.
- Austritt aus einem inneren Zustand: Die Transition wird nur genommen, wenn man sich gerade im entsprechenden Unterzustand (hier: Zustand E) befindet, und das entsprechende Ereignis stattfindet.

Außerdem: Austritt über einen Austrittspunkt, Endzustand oder Terminator (hier nicht behandelt).

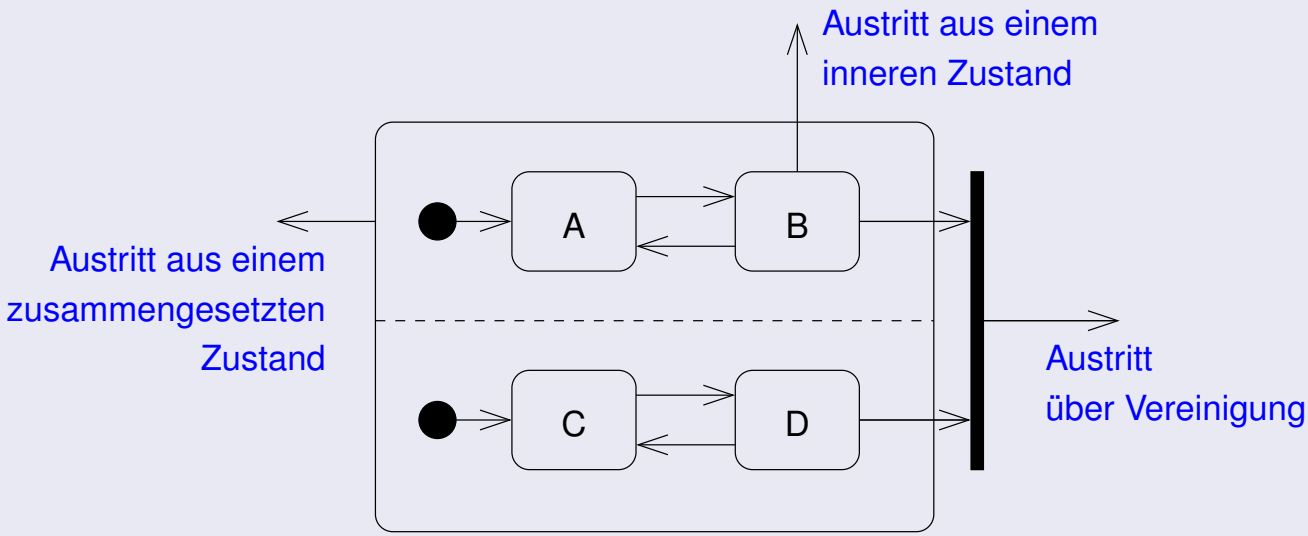
Zustandsdiagramme: Austrittsmöglichkeiten

Beispiel zu Austrittsmöglichkeiten:

Zustandsdiagramme: Austrittsmöglichkeiten

Auch für Austrittsmöglichkeiten müssen wir untersuchen, welche Besonderheiten sich bei Regionen ergeben.

Austrittsmöglichkeiten bei Regionen (grafisch)



Zustandsdiagramme: Austrittsmöglichkeiten

Beschreibung der Austrittsmöglichkeiten bei Regionen:

- Austritt aus einem zusammengesetzten Zustand: Dabei wird der zusammengesetzte Zustand verlassen, egal in welchen Unterzuständen man sich gerade befindet.

Zustandsdiagramme: Austrittsmöglichkeiten

Beschreibung der Austrittsmöglichkeiten bei Regionen:

- Austritt aus einem zusammengesetzten Zustand: Dabei wird der zusammengesetzte Zustand verlassen, egal in welchen Unterzuständen man sich gerade befindet.
- Austritt aus einem inneren Zustand: Der zusammengesetzte Zustand wird nur verlassen, wenn man sich in der entsprechenden Region in dem Zustand befindet, der durch den Pfeil verlassen wird. In den anderen Regionen kann man sich in beliebigen Zuständen befinden.

  (Hier Austritt nur, wenn man sich in der ersten Region in B befindet.)

Zustandsdiagramme: Austrittsmöglichkeiten

Beschreibung der Austrittsmöglichkeiten bei Regionen (Fortsetzung):

- Austritt über Vereinigung: Der zusammengesetzte Zustand kann nur verlassen werden, wenn man sich in den Regionen in den Zuständen befindet, von denen die Pfeile in die Vereinigung hineinführen.

  (Hier Austritt nur, wenn man sich in B und D befindet.)

Zustandsdiagramme: Beispiel

Wieder zurück zur Armbanduhr.

Zustandsdiagramme: Beispiel

Wieder zurück zur Armbanduhr.

Es gibt weitere Probleme:

Wenn man aus der Alarmeinstellung zurückkehrt, ist die Zeit auf 0:00 zurückgesetzt!

Und umgekehrt, wenn man zur Alarmeinstellung wechselt, wird der Alarm immer auf „on“ gesetzt!

Zustandsdiagramme: Beispiel

Wieder zurück zur Armbanduhr.

Es gibt weitere Probleme:

Wenn man aus der Alarmeinstellung zurückkehrt, ist die Zeit auf 0:00 zurückgesetzt!

Und umgekehrt, wenn man zur Alarmeinstellung wechselt, wird der Alarm immer auf „on“ gesetzt!

Lösung:

Jeweils Verwendung des Eintritts über die (flache) Historie.

Zustandsdiagramme: Beispiel

Wieder zurück zur Armbanduhr.

Es gibt weitere Probleme:

Wenn man aus der Alarmeinstellung zurückkehrt, ist die Zeit auf 0:00 zurückgesetzt!

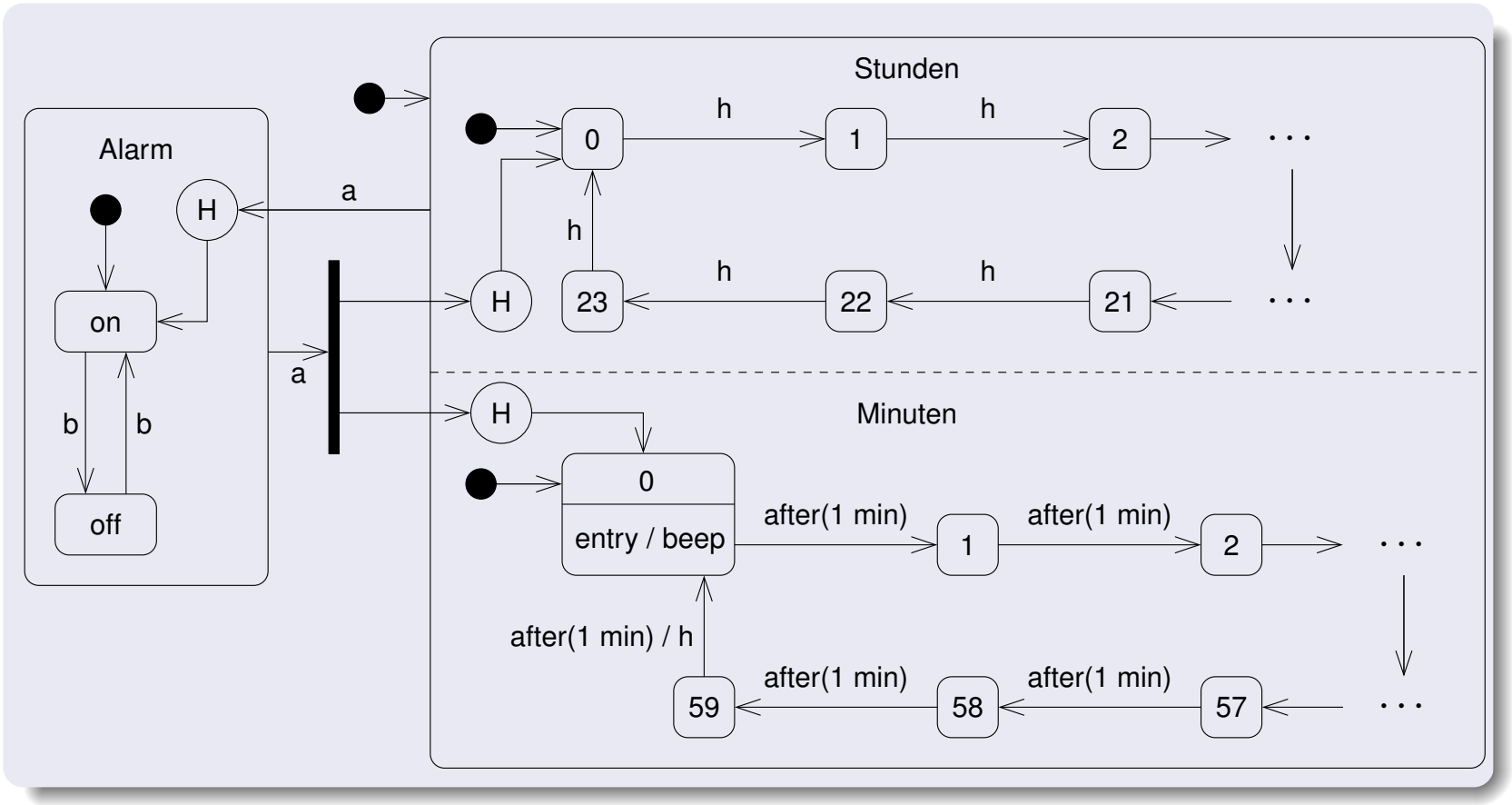
Und umgekehrt, wenn man zur Alarmeinstellung wechselt, wird der Alarm immer auf „on“ gesetzt!

Lösung:

Jeweils Verwendung des Eintritts über die (flache) Historie.

Da man im Fall der Zeitanzeige in zwei Regionen gleichzeitig eintreten muss, verwenden wir dort eine Gabelung.

Zustandsdiagramme: Beispiel



Zustandsdiagramme: Beispiel

Weiteres Problem: Wenn man längere Zeit in der Alarmanzeige verbringt, so wird in dieser Zeit die Minuten-/Stundenanzeige nicht entsprechend aktualisiert. Denn das Ereignis „after(1 min)“ bezieht sich nur auf die Zeit, die seit dem Eintritt in den entsprechenden Zustand vergangen ist.

Die Zeitanzeige müsste deshalb entsprechend aktualisiert werden. Dieses Problem lösen wir hier nicht.

Zustandsdiagramme: Beispiel

Was fehlt ansonsten noch?

Zustandsdiagramme: Beispiel

Was fehlt ansonsten noch?

Beim Wechseln zwischen den Alarm-Zuständen (on, off) muss ein Flag (hier al genannt) gesetzt werden, um damit den beep-Effekt zu kontrollieren.

Zustandsdiagramme: Beispiel

Was fehlt ansonsten noch?

Beim Wechseln zwischen den Alarm-Zuständen (on, off) muss ein Flag (hier al genannt) gesetzt werden, um damit den beep-Effekt zu kontrollieren.

Dieses Flag muss dann mit Hilfe einer Bedingung im Minutenzustand 0 abgefragt werden.

Zustandsdiagramme: Beispiel

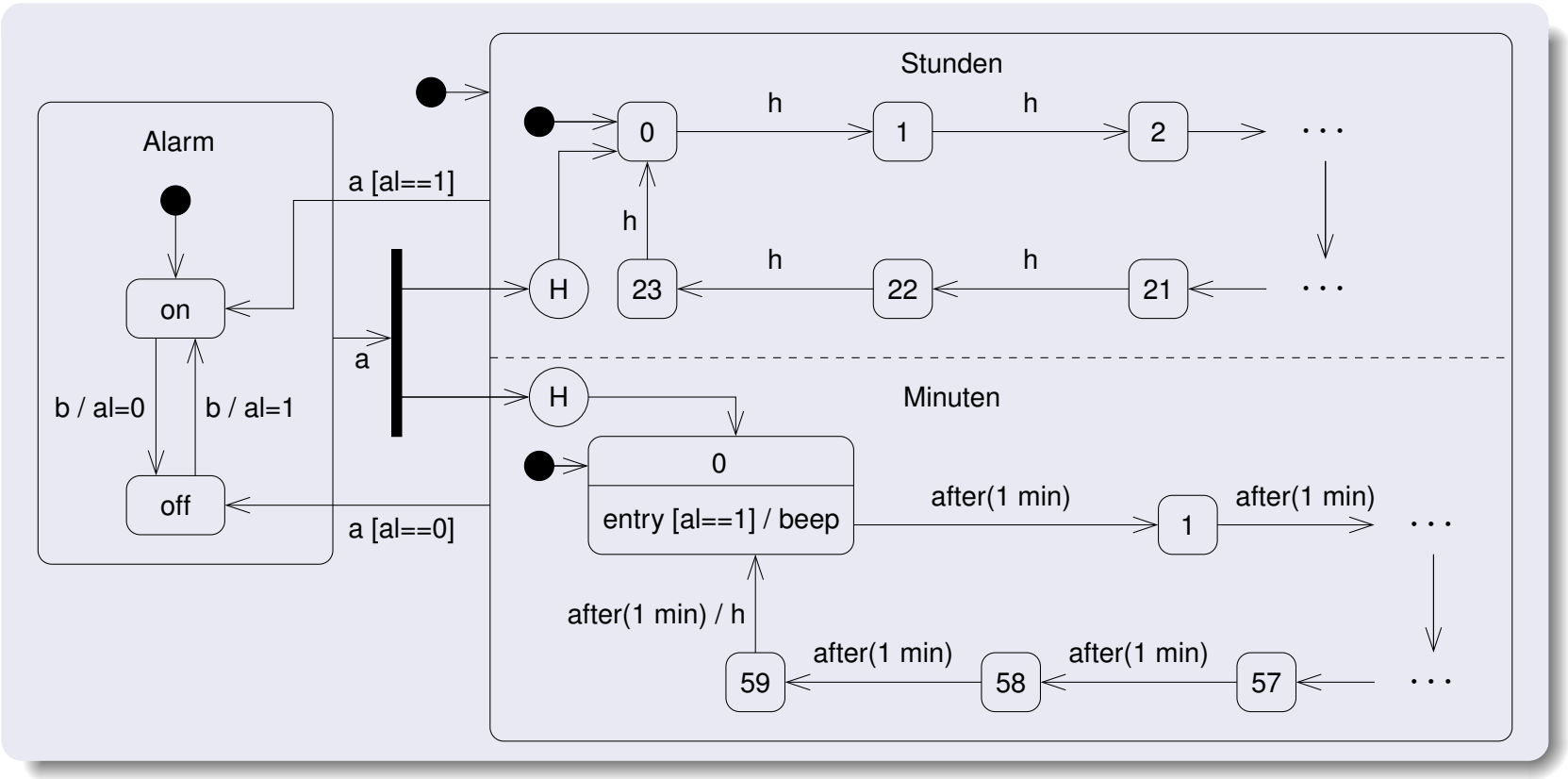
Was fehlt ansonsten noch?

Beim Wechseln zwischen den Alarm-Zuständen (on, off) muss ein Flag (hier al genannt) gesetzt werden, um damit den beep-Effekt zu kontrollieren.

Dieses Flag muss dann mit Hilfe einer Bedingung im Minutenzustand 0 abgefragt werden.

Außerdem betreten wir den Zustand Alarm nun nicht mehr über die flache Historie, sondern fragen mit Hilfe von Bedingungen ab, wie al belegt ist.

Zustandsdiagramme: Beispiel



Zustandsdiagramme: Beispiel

Schließlich modellieren wir noch, dass die Batterie der Uhr leer werden kann und gewechselt werden muss.

Zustandsdiagramme: Beispiel

Schließlich modellieren wir noch, dass die Batterie der Uhr leer werden kann und gewechselt werden muss.

In diesem Fall will man den zusammengesetzten Zustand nicht über die flache oder tiefe Historie betreten! Es wird also dann tatsächlich die Zeit auf 0:00 zurückgesetzt.

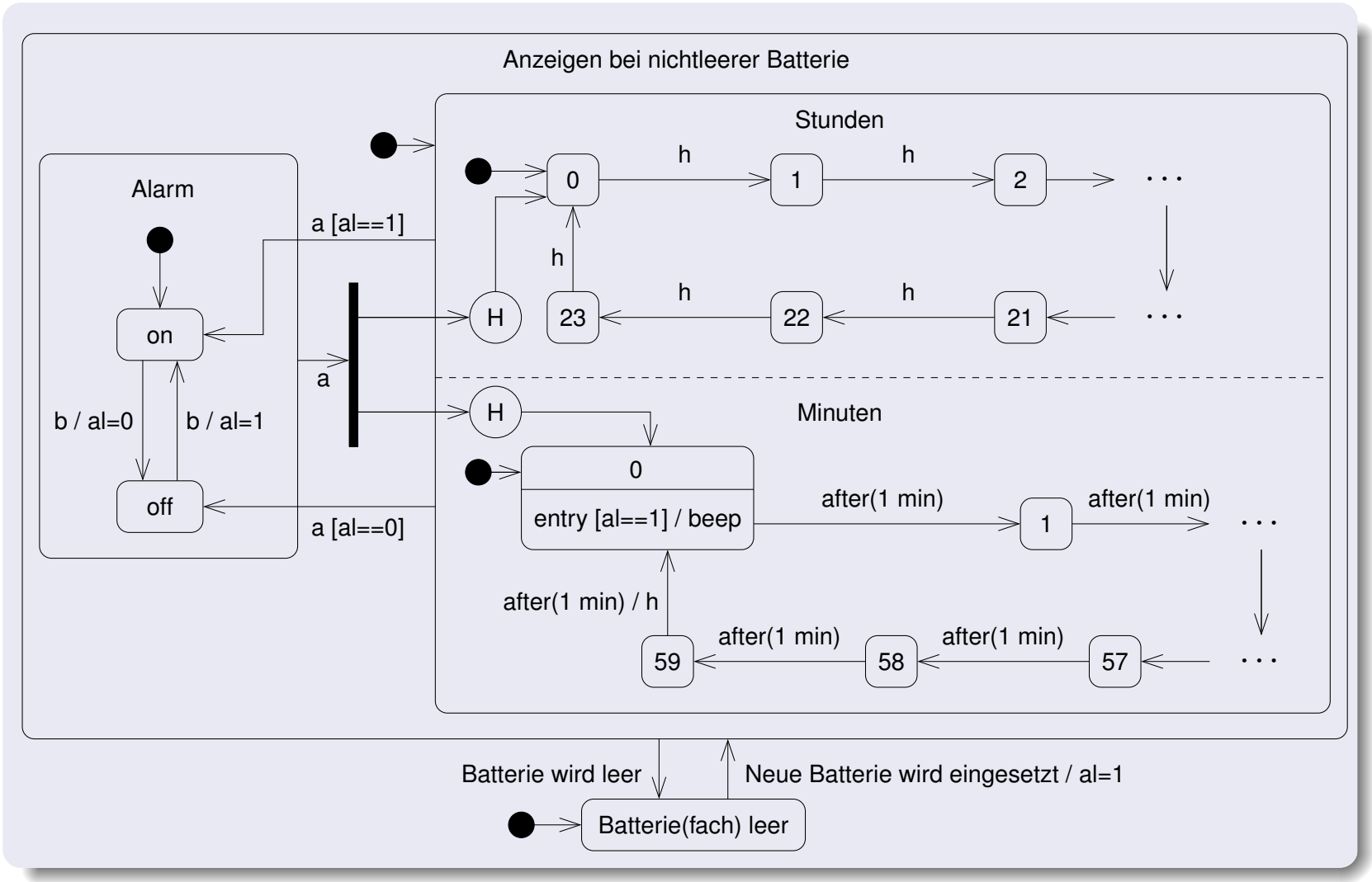
Zustandsdiagramme: Beispiel

Schließlich modellieren wir noch, dass die Batterie der Uhr leer werden kann und gewechselt werden muss.

In diesem Fall will man den zusammengesetzten Zustand nicht über die flache oder tiefe Historie betreten! Es wird also dann tatsächlich die Zeit auf 0:00 zurückgesetzt.

Außerdem setzen wir das Flag al beim Einsetzen der Batterie (zurück) auf den Anfangswert 1.

Zustandsdiagramme: Beispiel



Zustandsdiagramme: „Flachklopfen“

Ein großer Teil der Modellierungsmöglichkeiten von Zustandsdiagrammen dient dazu, diese kompakt und übersichtlich zu notieren.

Zustandsdiagramme: „Flachklopfen“

Ein großer Teil der Modellierungsmöglichkeiten von Zustandsdiagrammen dient dazu, diese kompakt und übersichtlich zu notieren.

Umgekehrt kann man Zustandsdiagramme oft „flachklopfen“ und zusammengesetzte Zustände auflösen, wodurch man äquivalente Zustandsdiagramme erhält, die die gleichen Übergänge erlauben. Dabei erhält man jedoch im Allgemeinen mehr Zustände und/oder Transitionen.

Zustandsdiagramme: „Flachklopfen“

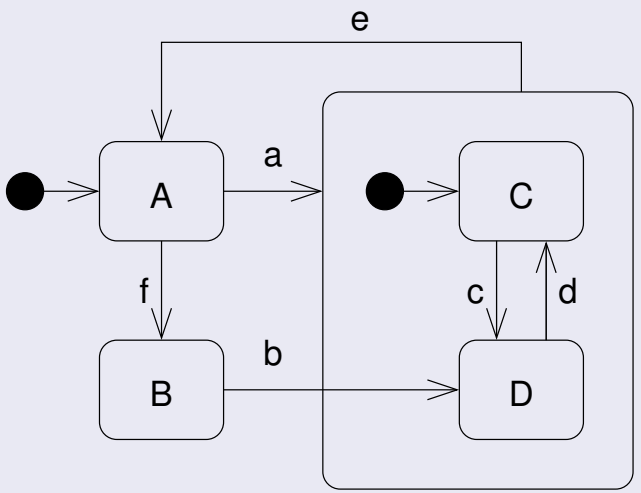
Ein großer Teil der Modellierungsmöglichkeiten von Zustandsdiagrammen dient dazu, diese kompakt und übersichtlich zu notieren.

Umgekehrt kann man Zustandsdiagramme oft „flachklopfen“ und zusammengesetzte Zustände auflösen, wodurch man äquivalente Zustandsdiagramme erhält, die die gleichen Übergänge erlauben. Dabei erhält man jedoch im Allgemeinen mehr Zustände und/oder Transitionen.

Wir sehen uns einige Beispiele an (und werden bestimmte Features dabei bewusst nicht betrachten) . . .

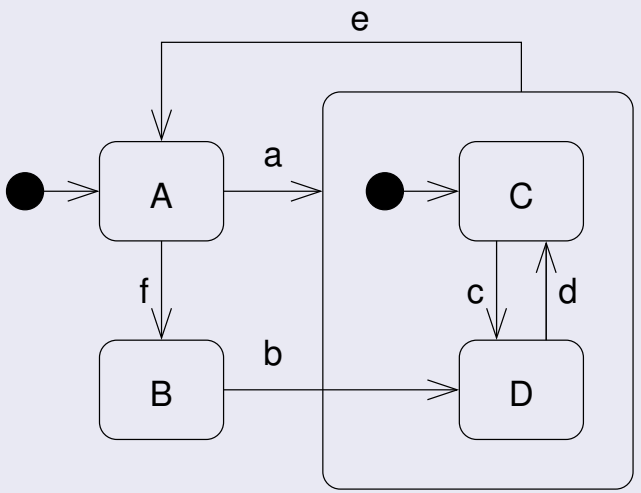
Zustandsdiagramme: „Flachklopfen“

Beispiel 1: Wandeln Sie folgendes Zustandsdiagramm in ein „flaches“ Zustandsdiagramm um:



Zustandsdiagramme: „Flachklopfen“

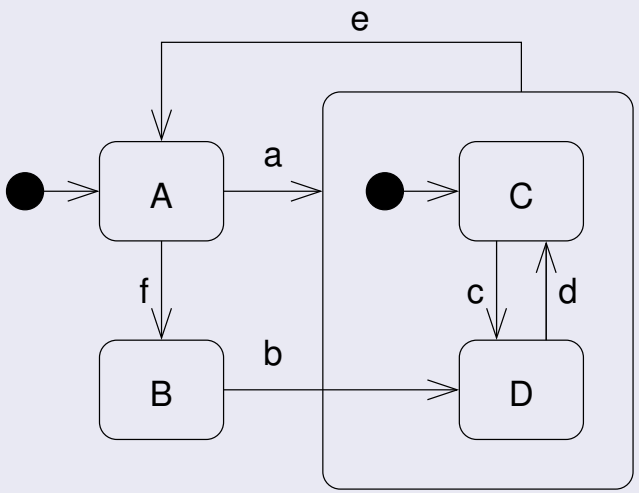
Beispiel 1: Wandeln Sie folgendes Zustandsdiagramm in ein „flaches“ Zustandsdiagramm um:



Charakteristika dieses Beispiels: keine Regionen, keine Historie

Zustandsdiagramme: „Flachklopfen“

Beispiel 1: Wandeln Sie folgendes Zustandsdiagramm in ein „flaches“ Zustandsdiagramm um:

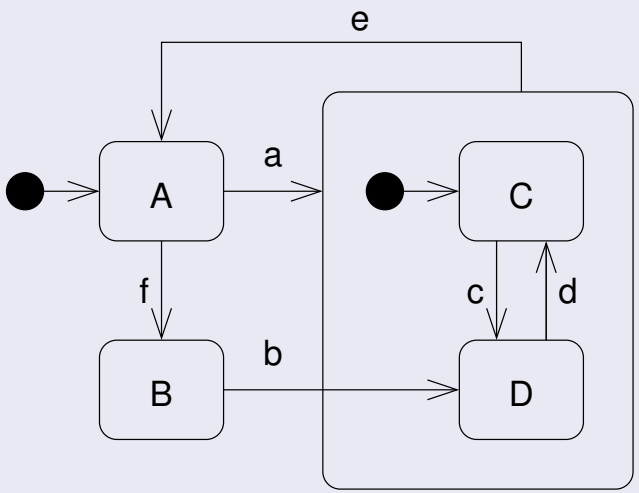


Charakteristika dieses Beispiels: keine Regionen, keine Historie

Ansatz: einfache Zustände behalten,

Zustandsdiagramme: „Flachklopfen“

Beispiel 1: Wandeln Sie folgendes Zustandsdiagramm in ein „flaches“ Zustandsdiagramm um:

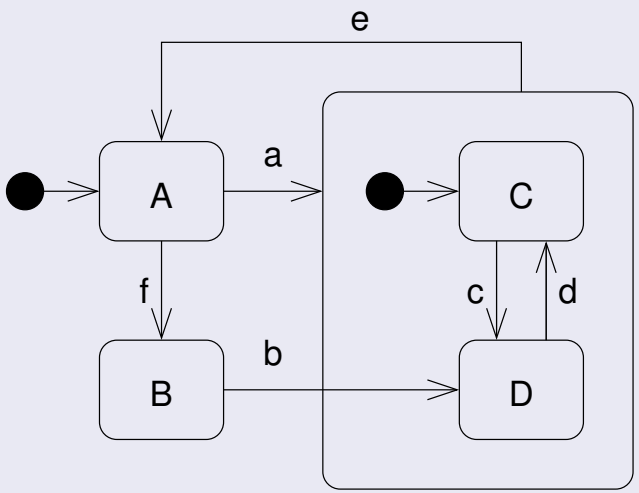


Charakteristika dieses Beispiels: keine Regionen, keine Historie

Ansatz: einfache Zustände behalten, Eintritte/Austritte an Rand zusammengesetzter Zustände übersetzen,

Zustandsdiagramme: „Flachklopfen“

Beispiel 1: Wandeln Sie folgendes Zustandsdiagramm in ein „flaches“ Zustandsdiagramm um:

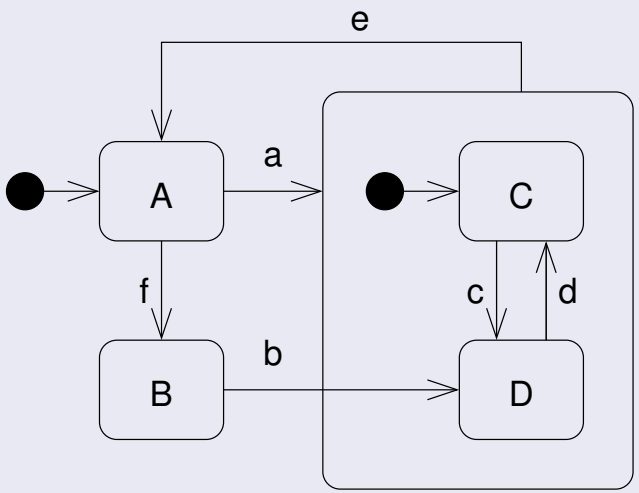


Charakteristika dieses Beispiels: keine Regionen, keine Historie

Ansatz: einfache Zustände behalten, Eintritte/Austritte an Rand zusammengesetzter Zustände übersetzen, andere Übergänge einfach behalten,

Zustandsdiagramme: „Flachklopfen“

Beispiel 1: Wandeln Sie folgendes Zustandsdiagramm in ein „flaches“ Zustandsdiagramm um:

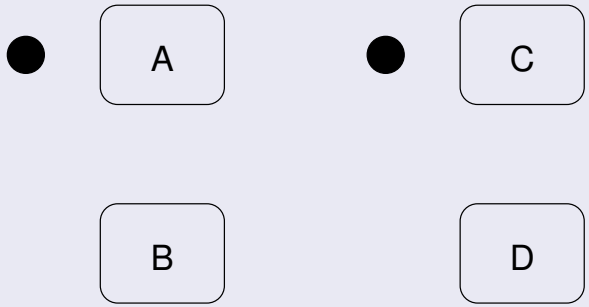


Charakteristika dieses Beispiels: keine Regionen, keine Historie

Ansatz: einfache Zustände behalten, Eintritte/Austritte an Rand zusammengesetzter Zustände übersetzen, andere Übergänge einfach behalten, und nur äußerster Startzustand bleibt solcher

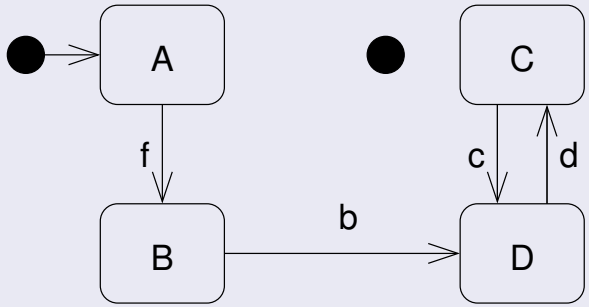
Zustandsdiagramme: „Flachklopfen“

Lösung zu Beispiel 1:



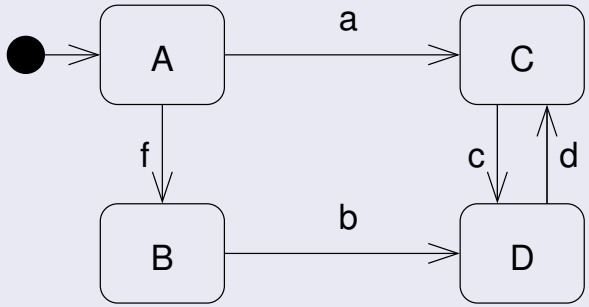
Zustandsdiagramme: „Flachklopfen“

Lösung zu Beispiel 1:



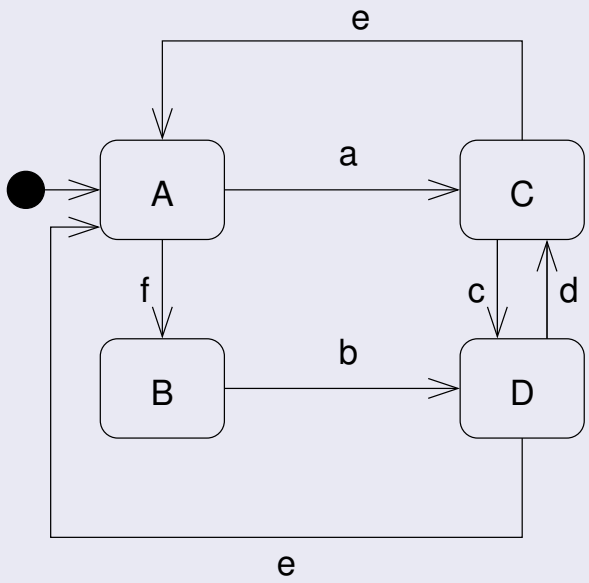
Zustandsdiagramme: „Flachklopfen“

Lösung zu Beispiel 1:



Zustandsdiagramme: „Flachklopfen“

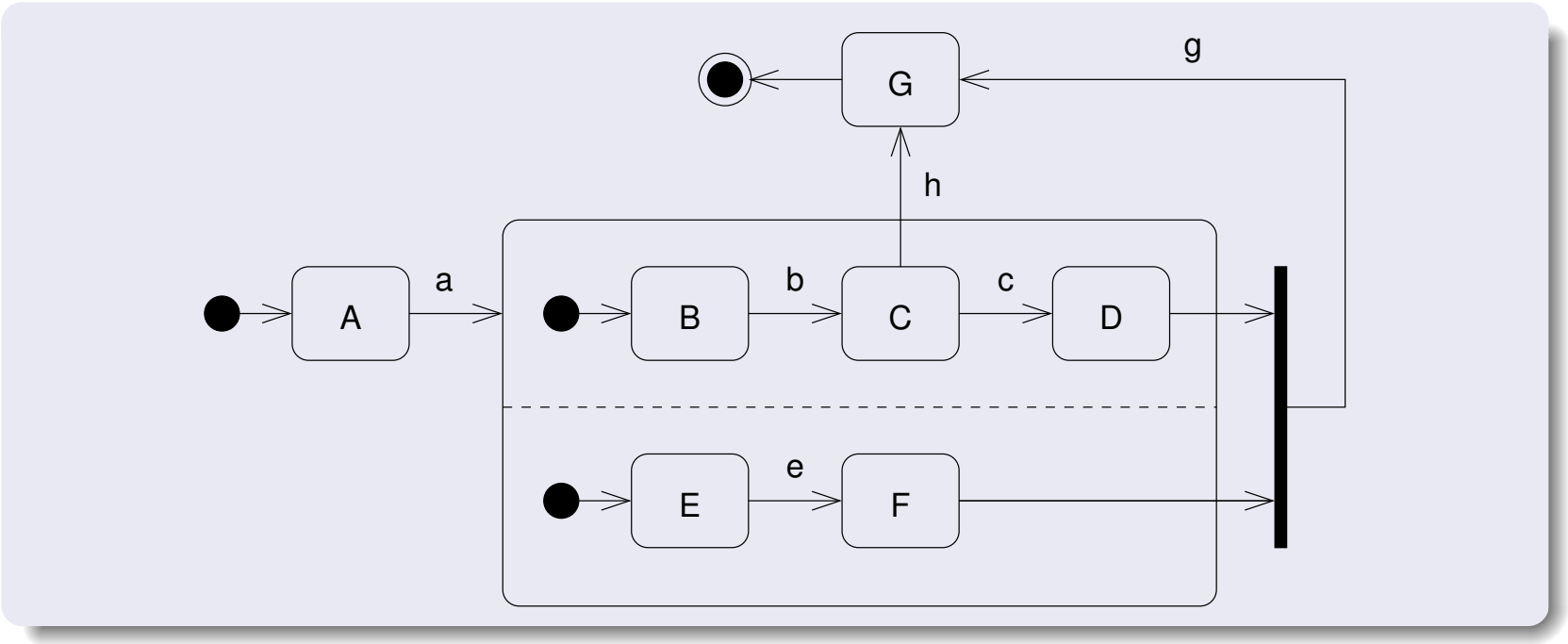
Lösung zu Beispiel 1:



Hauptschritt: Die Transition, die von dem zusammengesetzten Zustand wegführt, wurde durch mehrere Transitionen ersetzt, die von den inneren Zuständen ausgehen.

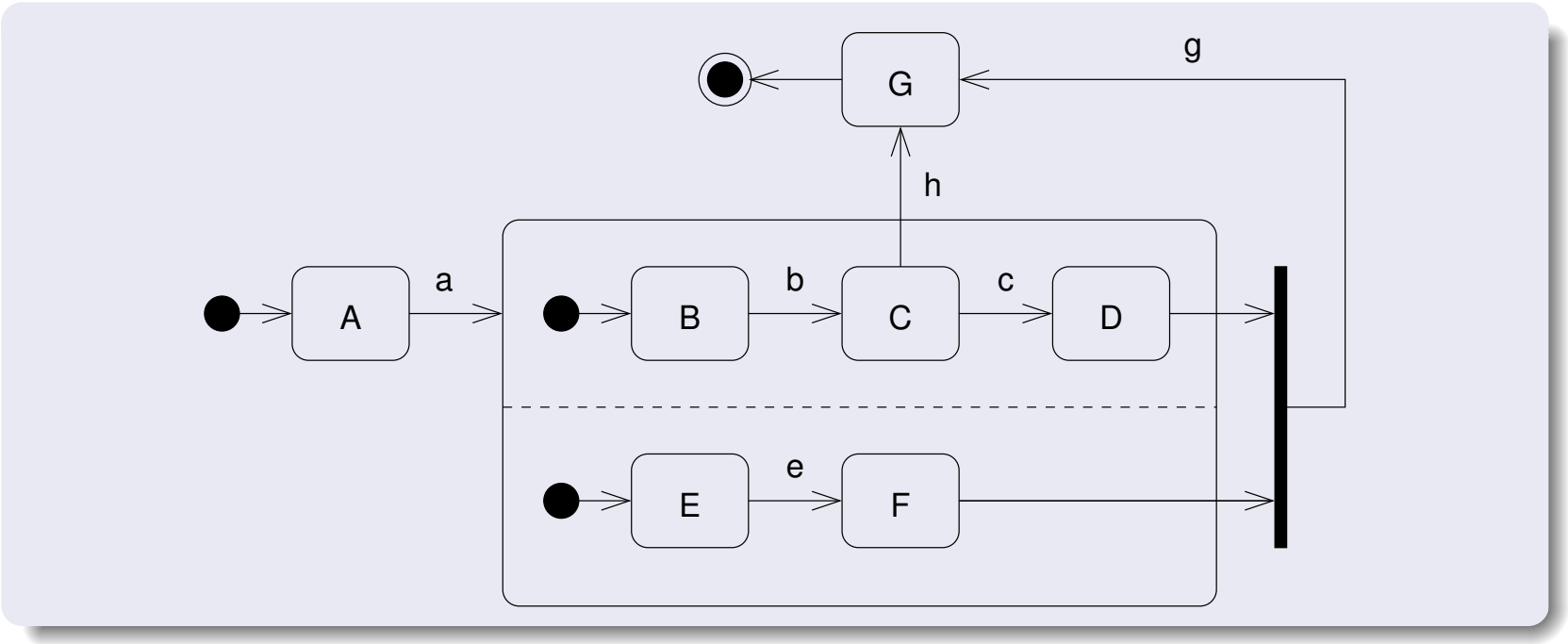
Zustandsdiagramme: „Flachklopfen“

Beispiel 2: Wandeln Sie folgendes Zustandsdiagramm in ein „flaches“ Zustandsdiagramm um:



Zustandsdiagramme: „Flachklopfen“

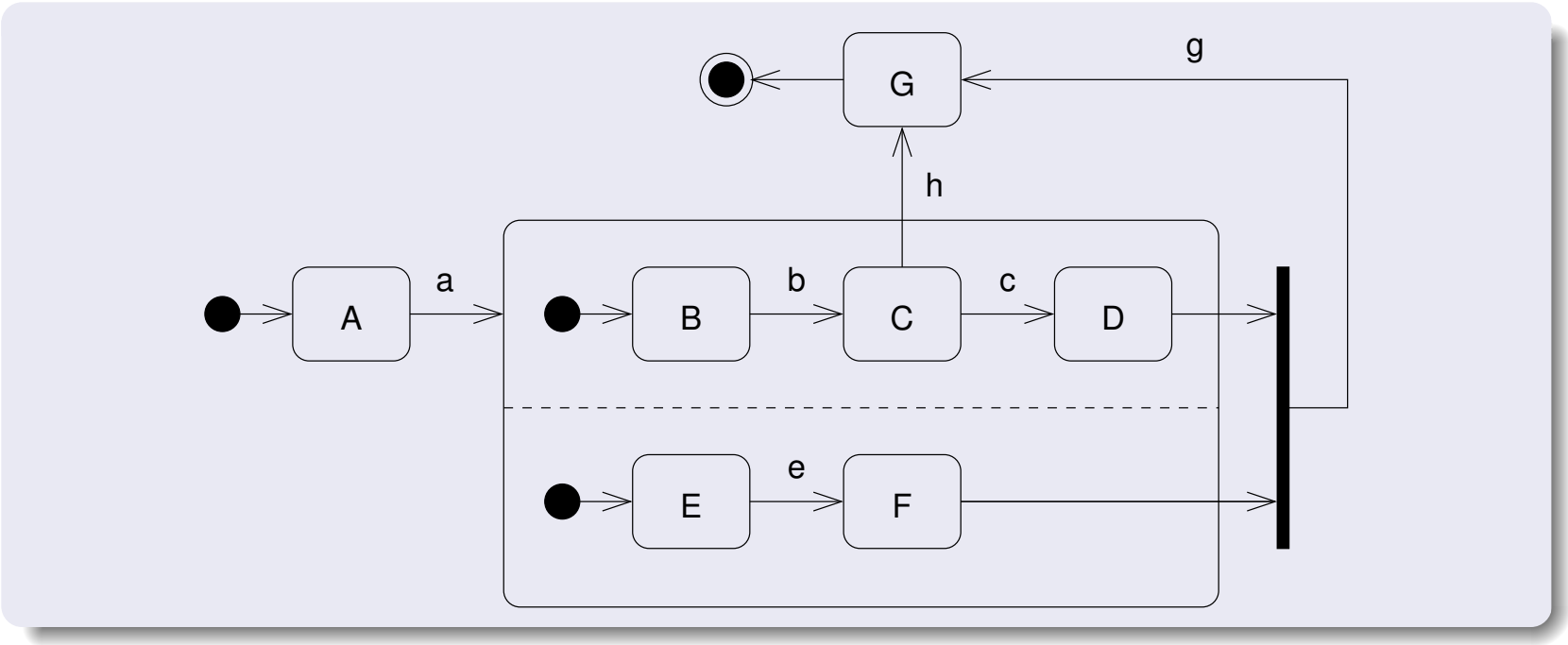
Beispiel 2: Wandeln Sie folgendes Zustandsdiagramm in ein „flaches“ Zustandsdiagramm um:



Charakteristika dieses Beispiels: Regionen, aber keine Historie

Zustandsdiagramme: „Flachklopfen“

Beispiel 2: Wandeln Sie folgendes Zustandsdiagramm in ein „flaches“ Zustandsdiagramm um:

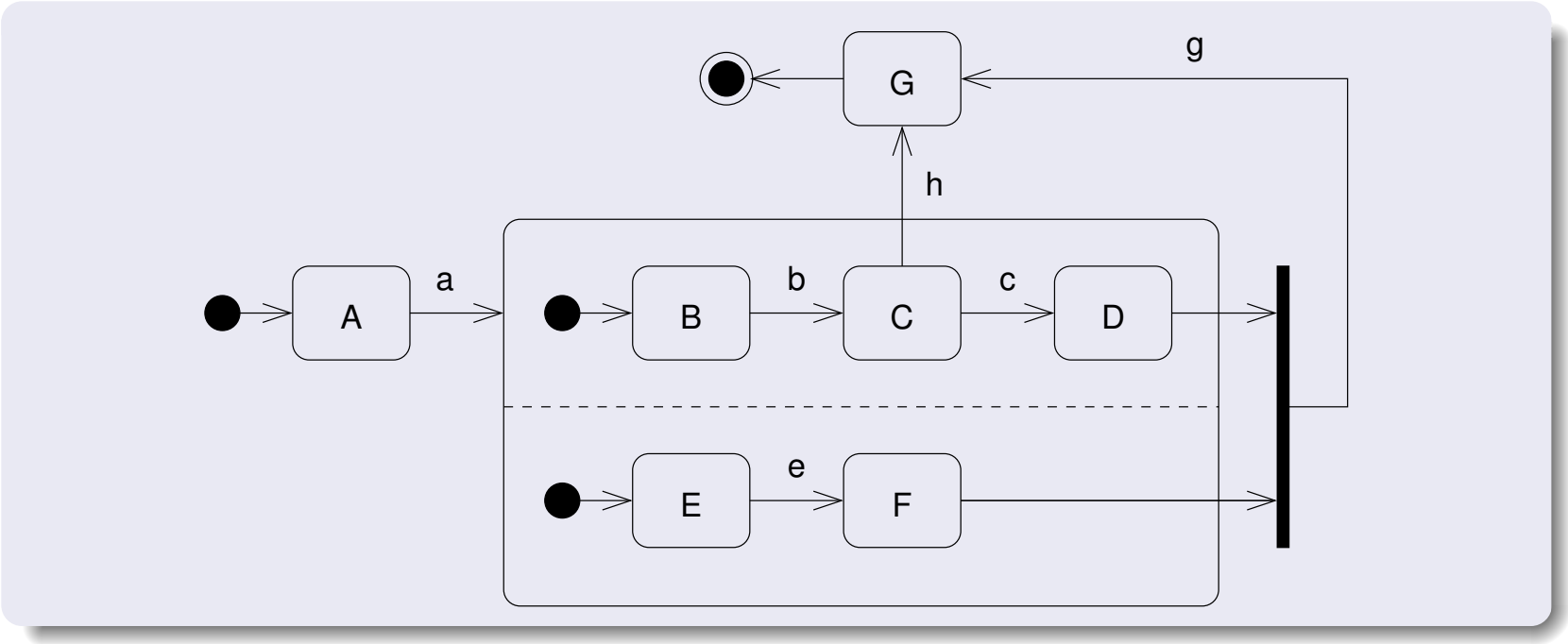


Charakteristika dieses Beispiels: Regionen, aber keine Historie

Ansatz außerhalb Regionen wie bisher,

Zustandsdiagramme: „Flachklopfen“

Beispiel 2: Wandeln Sie folgendes Zustandsdiagramm in ein „flaches“ Zustandsdiagramm um:

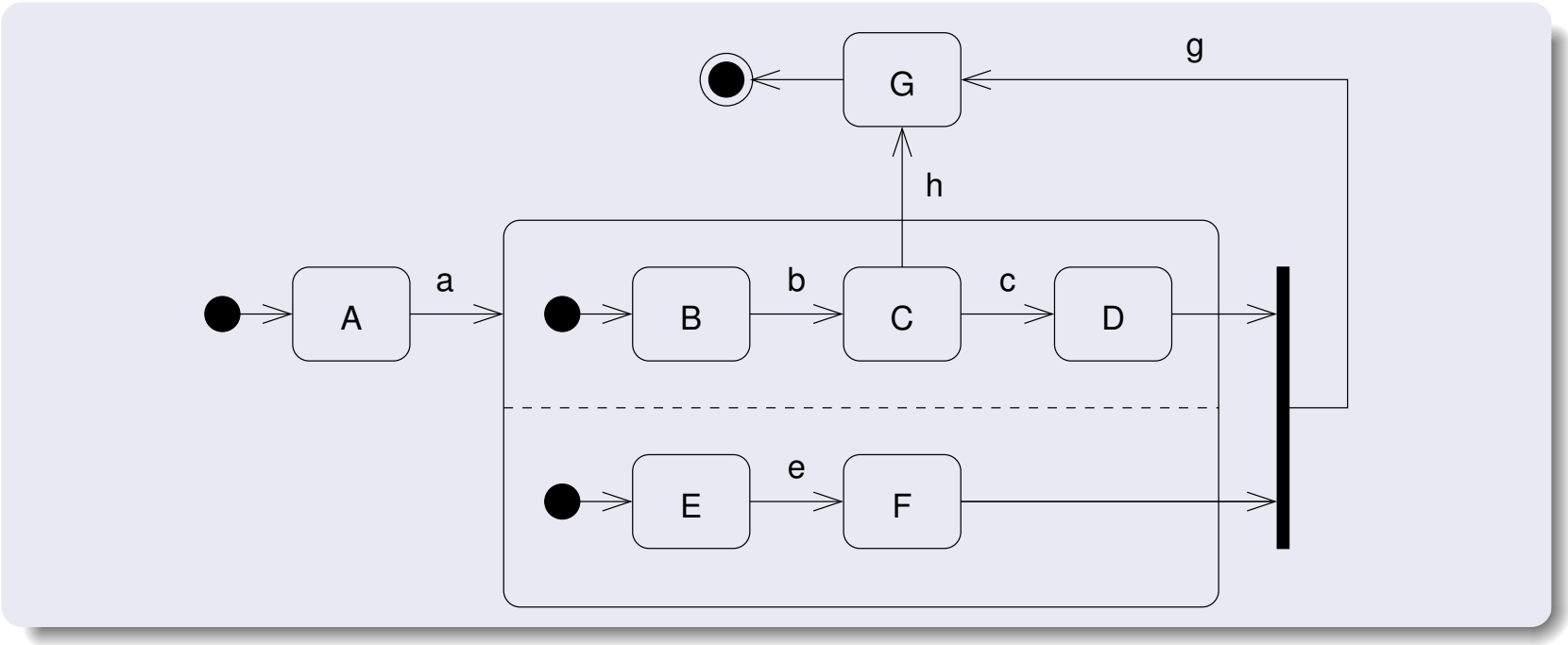


Charakteristika dieses Beispiels: Regionen, aber keine Historie

Ansatz außerhalb Regionen wie bisher, zusätzlich: Kreuzprodukt;

Zustandsdiagramme: „Flachklopfen“

Beispiel 2: Wandeln Sie folgendes Zustandsdiagramm in ein „flaches“ Zustandsdiagramm um:

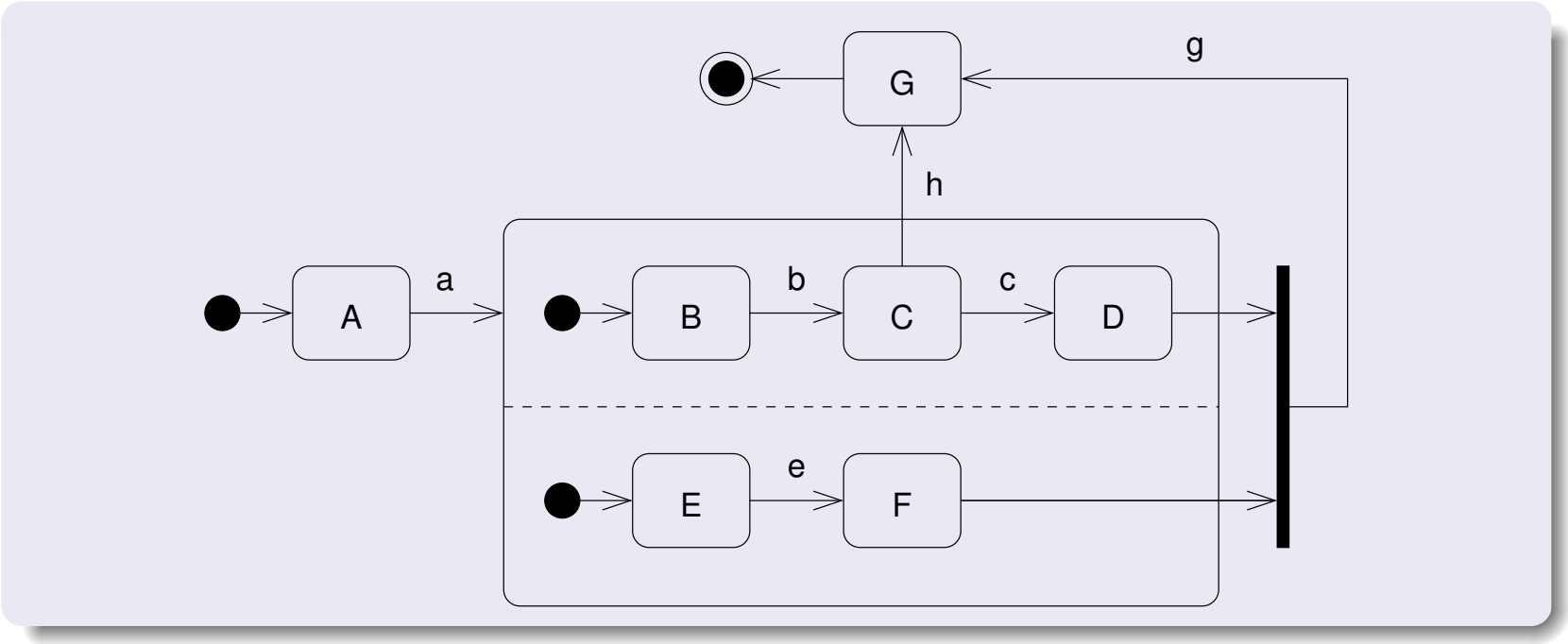


Charakteristika dieses Beispiels: Regionen, aber keine Historie

Ansatz außerhalb Regionen wie bisher, zusätzlich: Kreuzprodukt; parallele Übergänge entsprechend den Regionen;

Zustandsdiagramme: „Flachklopfen“

Beispiel 2: Wandeln Sie folgendes Zustandsdiagramm in ein „flaches“ Zustandsdiagramm um:

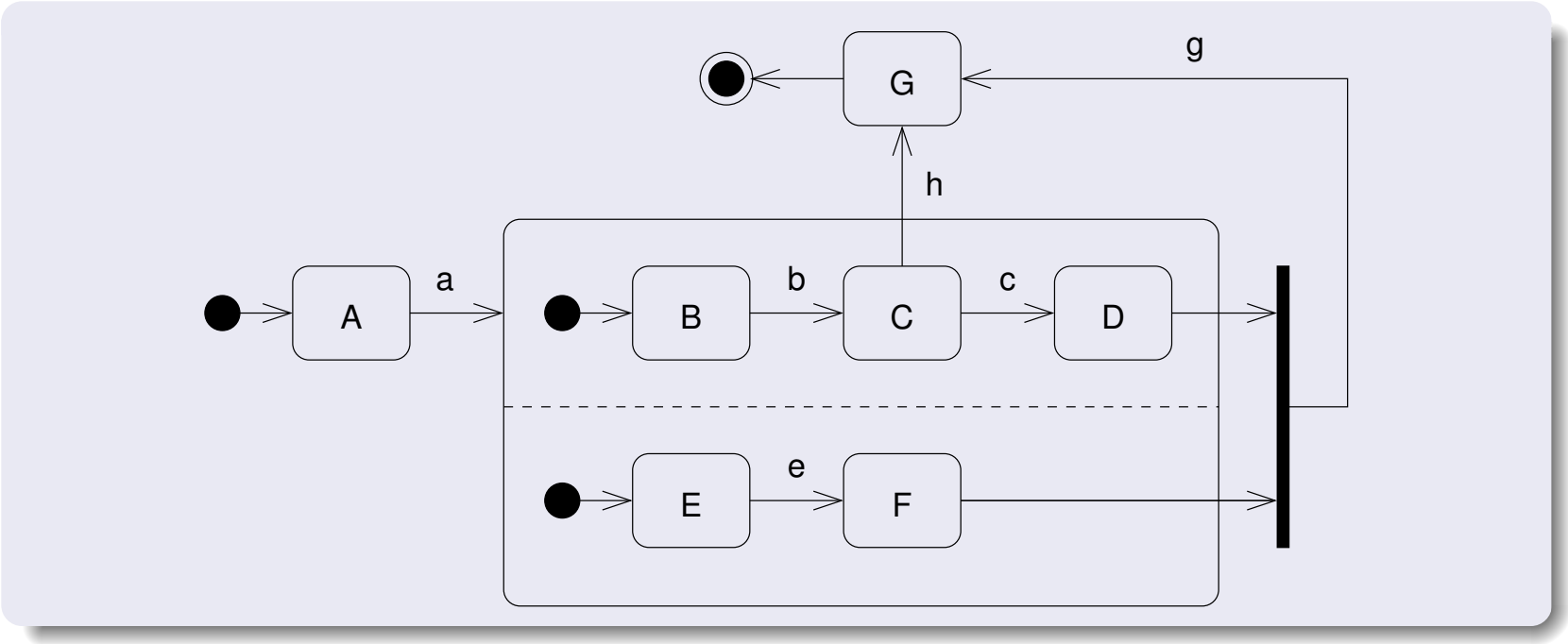


Charakteristika dieses Beispiels: Regionen, aber keine Historie

Ansatz außerhalb Regionen wie bisher, zusätzlich: Kreuzprodukt; parallele Übergänge entsprechend den Regionen; Ein-/Austritte an Rand von Regionen übersetzen

Zustandsdiagramme: „Flachklopfen“

Beispiel 2: Wandeln Sie folgendes Zustandsdiagramm in ein „flaches“ Zustandsdiagramm um:

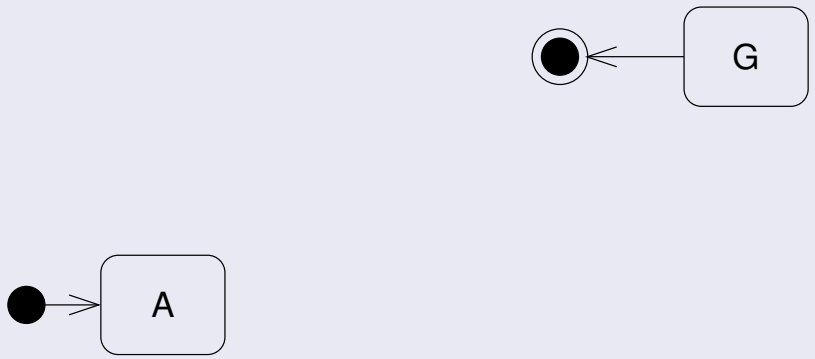


Charakteristika dieses Beispiels: Regionen, aber keine Historie

Ansatz außerhalb Regionen wie bisher, zusätzlich: Kreuzprodukt; parallele Übergänge entsprechend den Regionen; Ein-/Austritte an Rand von – aber auch hinein und heraus aus – Regionen übersetzen

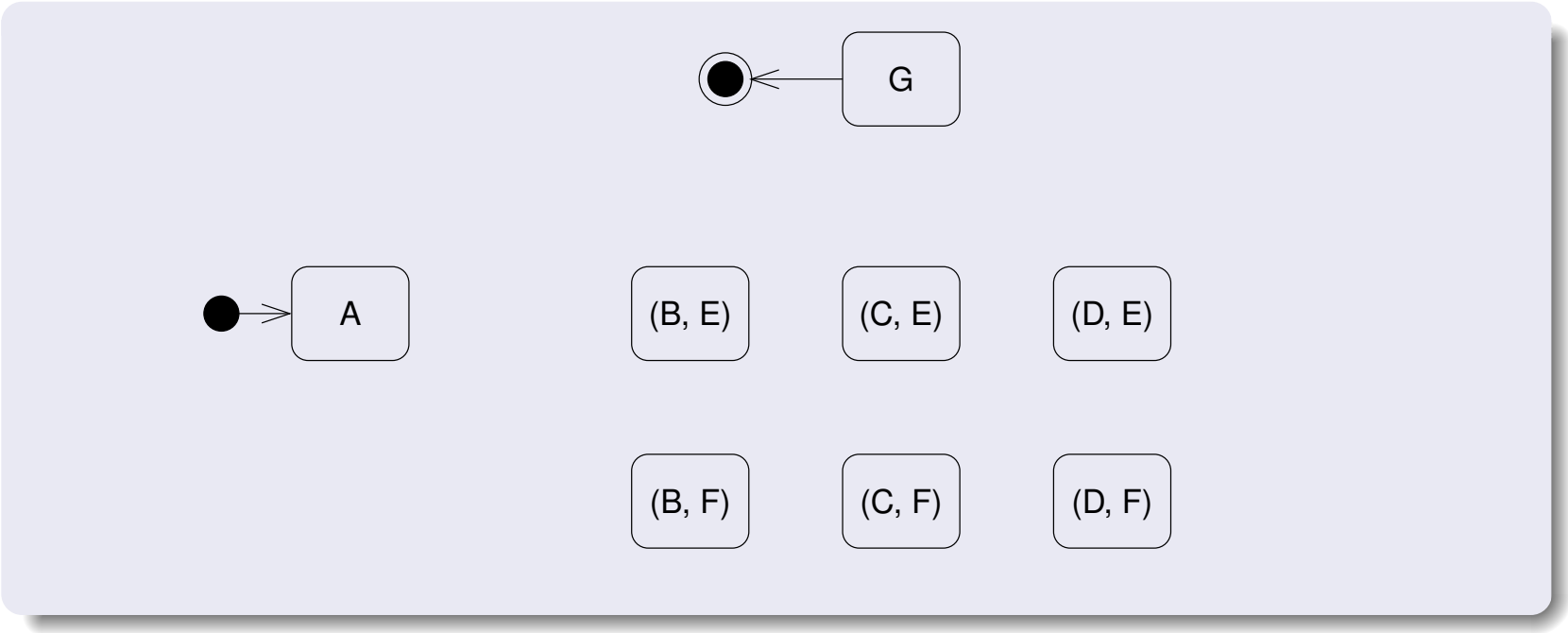
Zustandsdiagramme: „Flachklopfen“

Lösung zu Beispiel 2:



Zustandsdiagramme: „Flachklopfen“

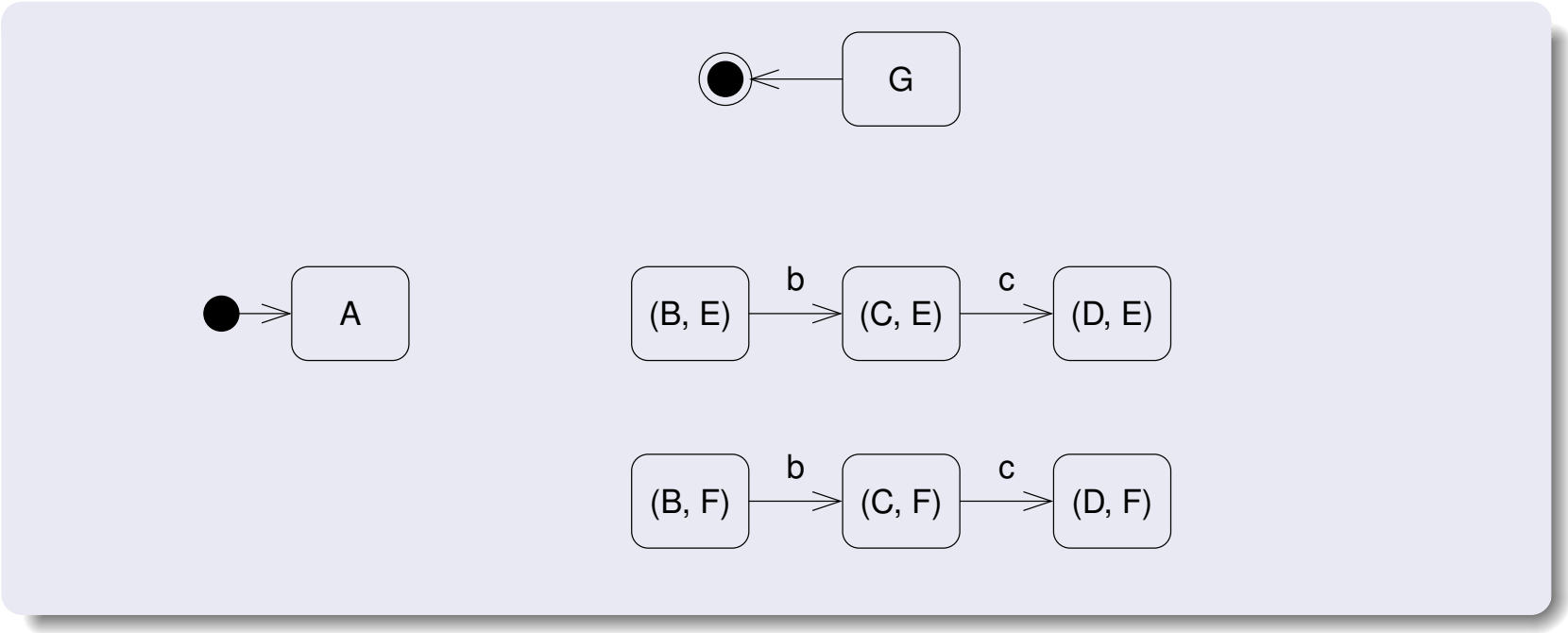
Lösung zu Beispiel 2:



Hauptidee: Kreuzprodukt der Zustandsmengen der Regionen bilden.

Zustandsdiagramme: „Flachklopfen“

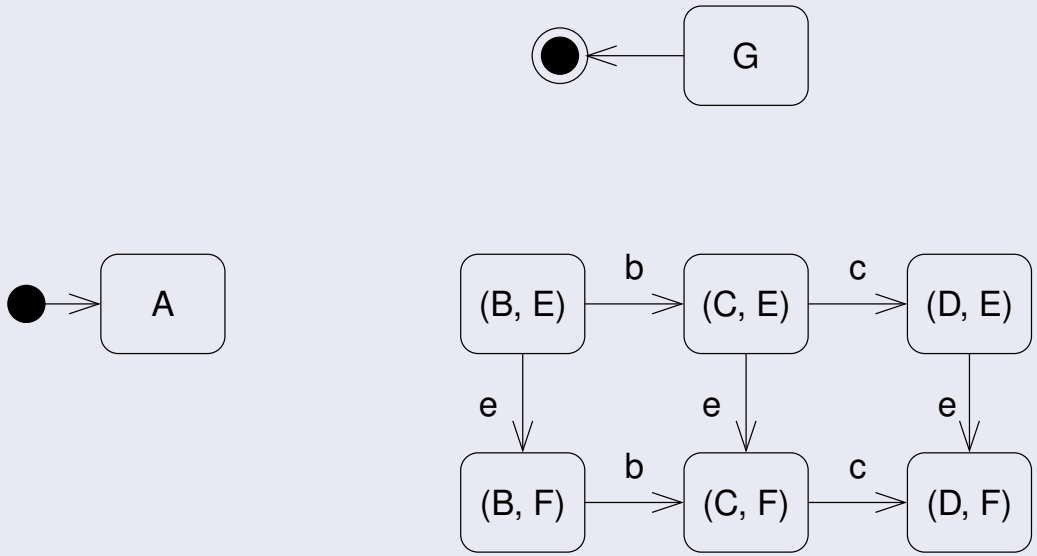
Lösung zu Beispiel 2:



Hauptidee: Kreuzprodukt der Zustandsmengen der Regionen bilden.

Zustandsdiagramme: „Flachklopfen“

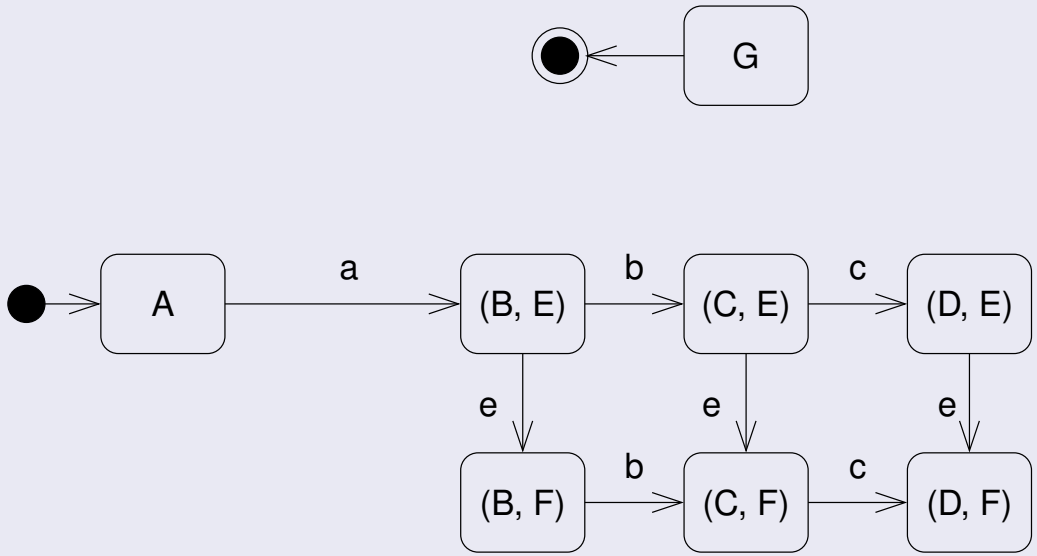
Lösung zu Beispiel 2:



Hauptidee: Kreuzprodukt der Zustandsmengen der Regionen bilden.

Zustandsdiagramme: „Flachklopfen“

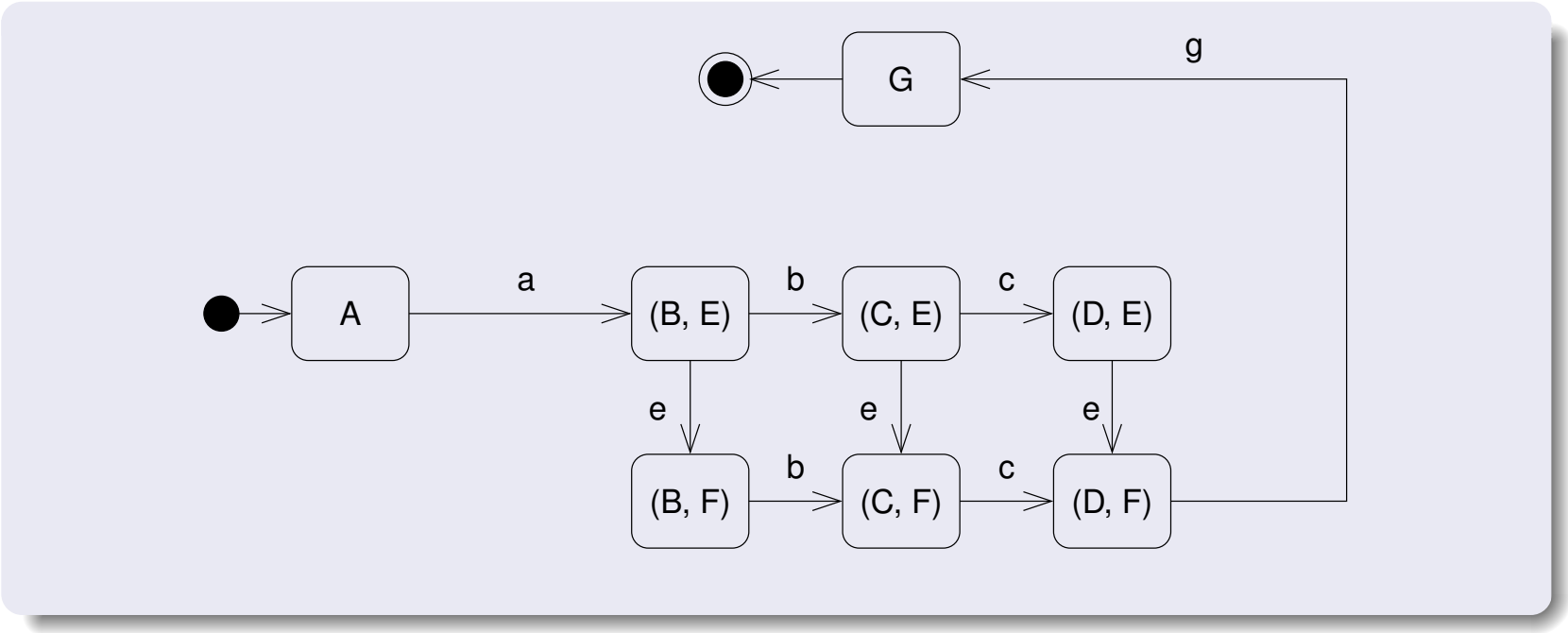
Lösung zu Beispiel 2:



Hauptidee: Kreuzprodukt der Zustandsmengen der Regionen bilden.

Zustandsdiagramme: „Flachklopfen“

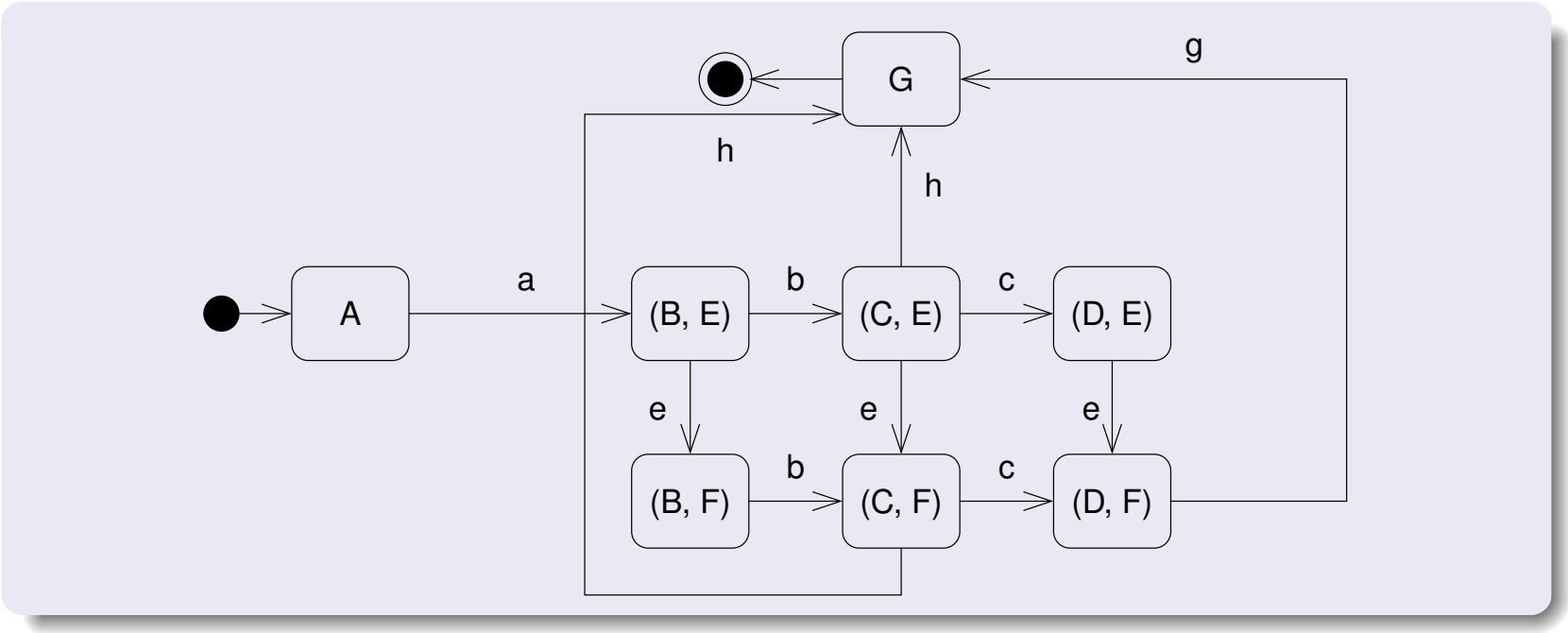
Lösung zu Beispiel 2:



Hauptidee: Kreuzprodukt der Zustandsmengen der Regionen bilden.

Zustandsdiagramme: „Flachklopfen“

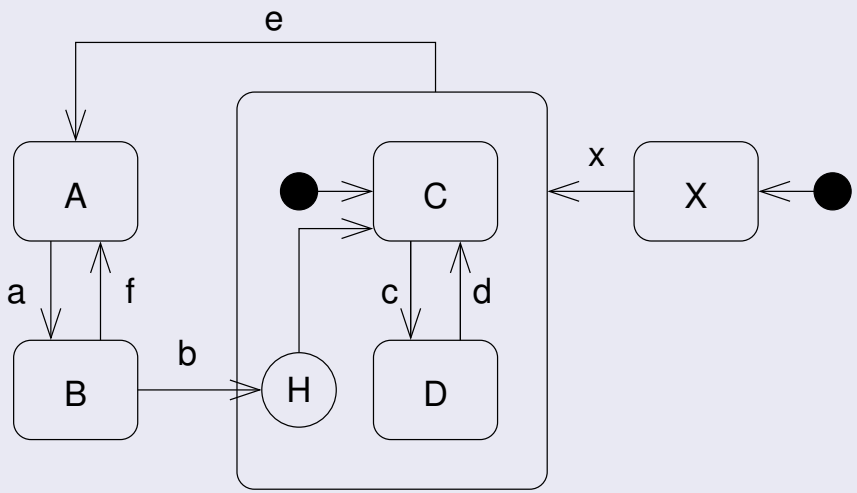
Lösung zu Beispiel 2:



Hauptidee: Kreuzprodukt der Zustandsmengen der Regionen bilden.

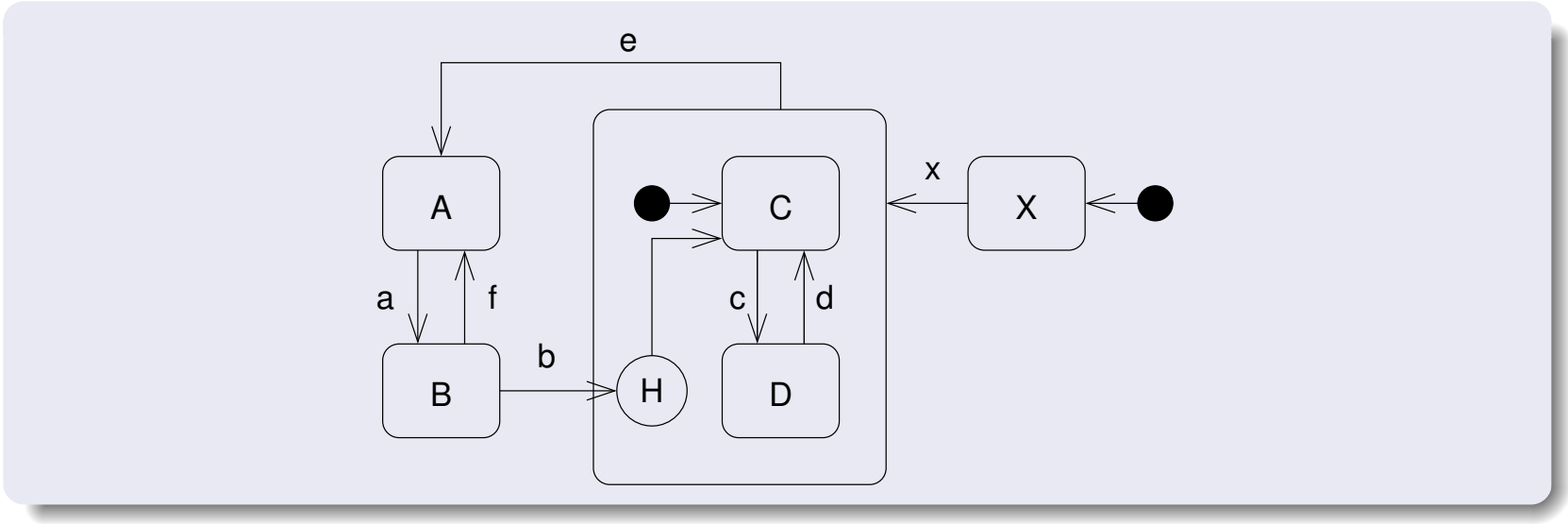
Zustandsdiagramme: „Flachklopfen“

Beispiel 3: Wandeln Sie folgendes Zustandsdiagramm in ein „flaches“ Zustandsdiagramm um:



Zustandsdiagramme: „Flachklopfen“

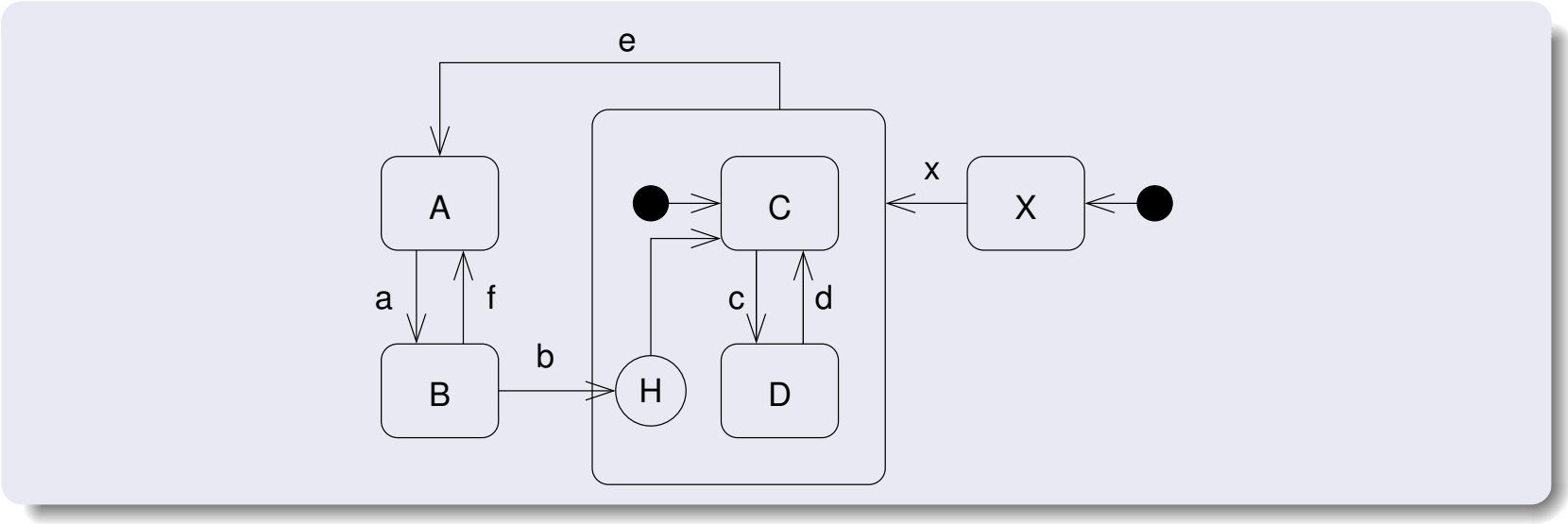
Beispiel 3: Wandeln Sie folgendes Zustandsdiagramm in ein „flaches“ Zustandsdiagramm um:



Charakteristika hier: keine Regionen, aber (flache) Historie

Zustandsdiagramme: „Flachklopfen“

Beispiel 3: Wandeln Sie folgendes Zustandsdiagramm in ein „flaches“ Zustandsdiagramm um:

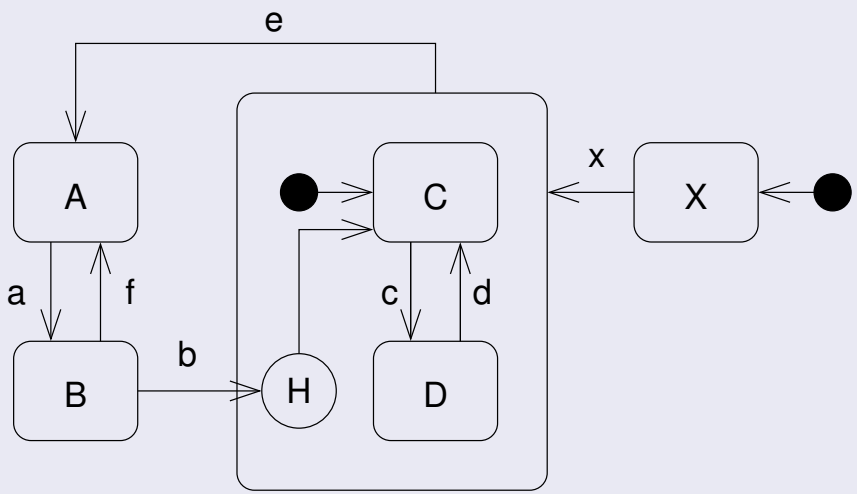


Charakteristika hier: keine Regionen, aber (flache) Historie

Ansatz jetzt: zunächst wie im einfachen Fall,

Zustandsdiagramme: „Flachklopfen“

Beispiel 3: Wandeln Sie folgendes Zustandsdiagramm in ein „flaches“ Zustandsdiagramm um:

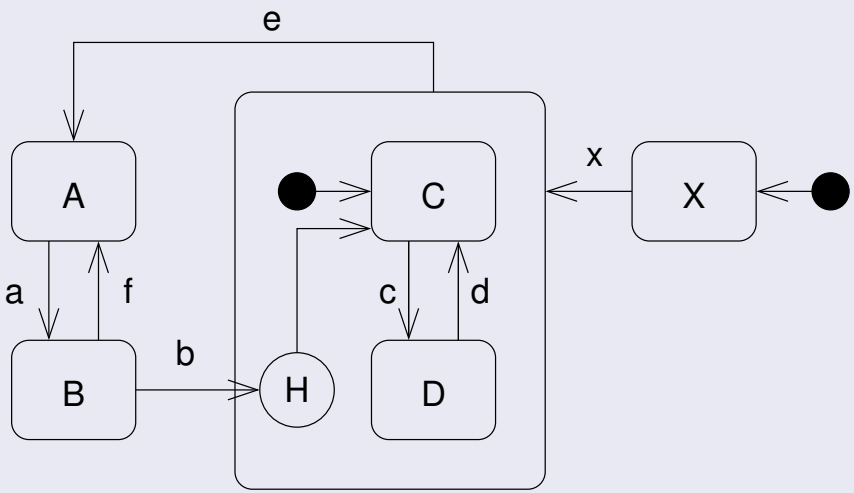


Charakteristika hier: keine Regionen, aber (flache) Historie

Ansatz jetzt: zunächst wie im einfachen Fall, aber für Verlassen eines zusammengesetzten Zustands mit Historien-Knoten gegebenenfalls Kopien äußerer Zustände (und ihrer Übergänge) zum Merken des letzten inneren Zustands,

Zustandsdiagramme: „Flachklopfen“

Beispiel 3: Wandeln Sie folgendes Zustandsdiagramm in ein „flaches“ Zustandsdiagramm um:

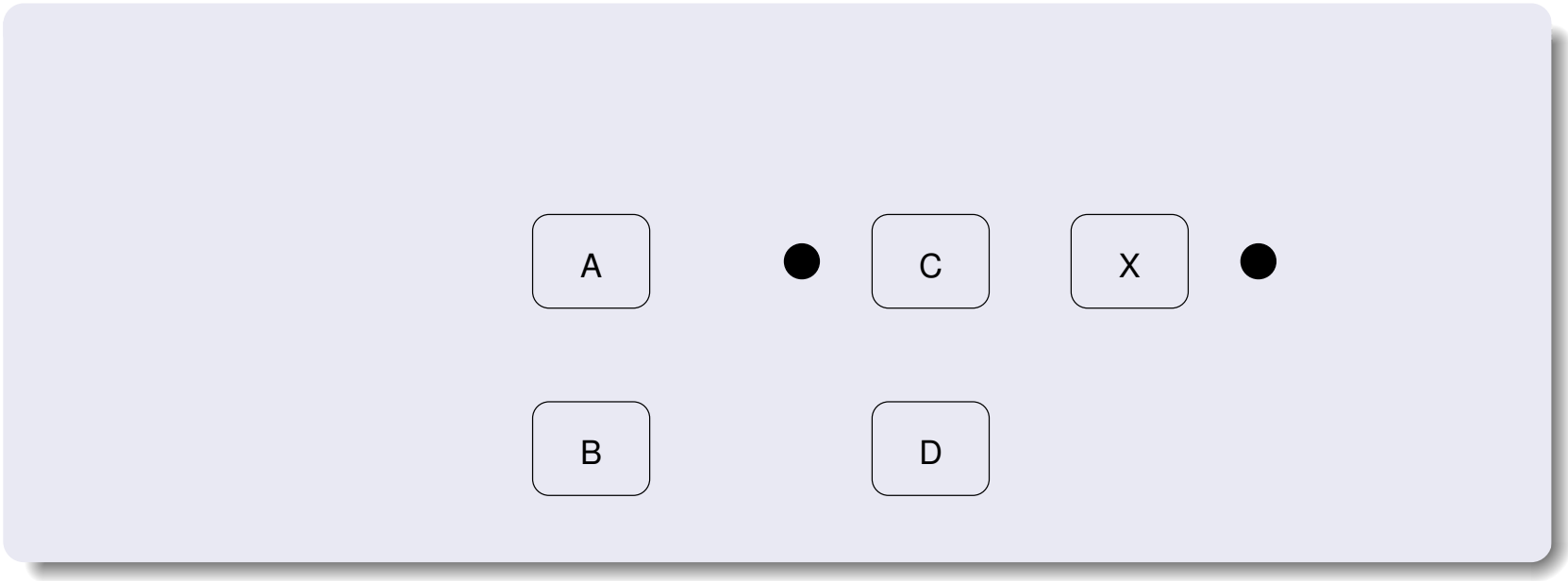


Charakteristika hier: keine Regionen, aber (flache) Historie

Ansatz jetzt: zunächst wie im einfachen Fall, aber für Verlassen eines zusammengesetzten Zustands mit Historien-Knoten gegebenenfalls Kopien äußerer Zustände (und ihrer Übergänge) zum Merken des letzten inneren Zustands, und Verwendung dieser Information bei Wiedereintritt über die Historie

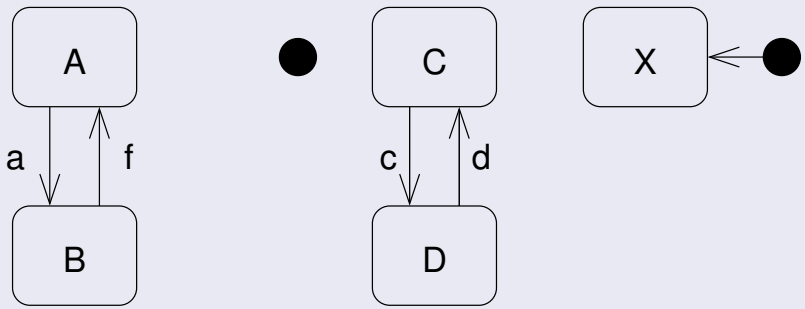
Zustandsdiagramme: „Flachklopfen“

Lösung zu Beispiel 3:



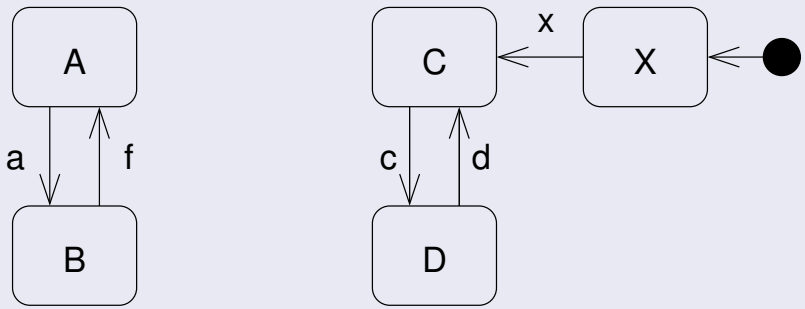
Zustandsdiagramme: „Flachklopfen“

Lösung zu Beispiel 3:



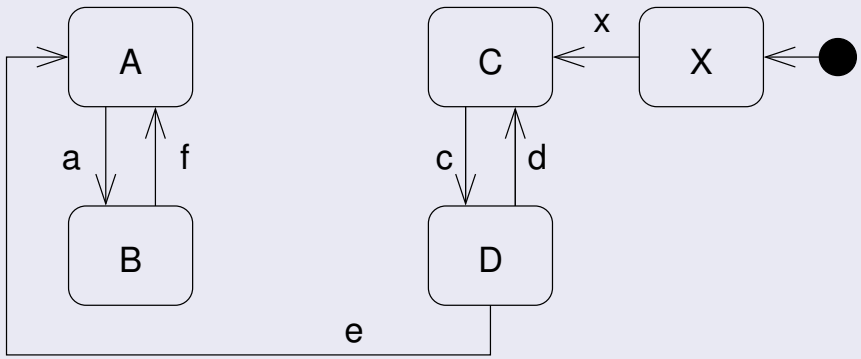
Zustandsdiagramme: „Flachklopfen“

Lösung zu Beispiel 3:



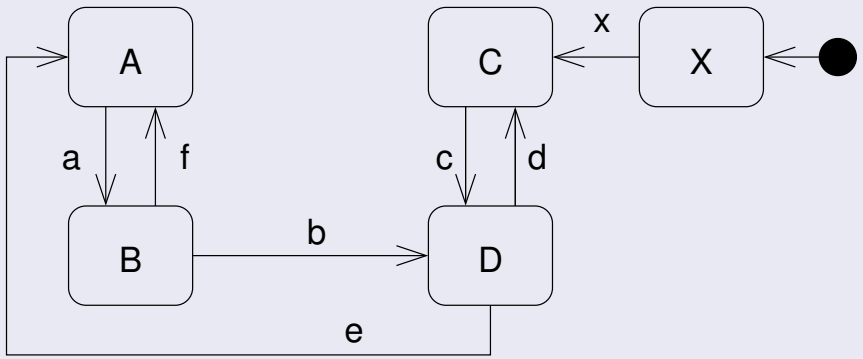
Zustandsdiagramme: „Flachklopfen“

Lösung zu Beispiel 3:



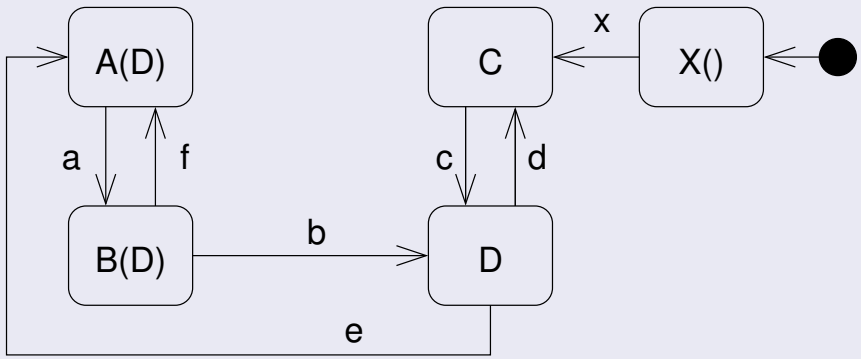
Zustandsdiagramme: „Flachklopfen“

Lösung zu Beispiel 3:



Zustandsdiagramme: „Flachklopfen“

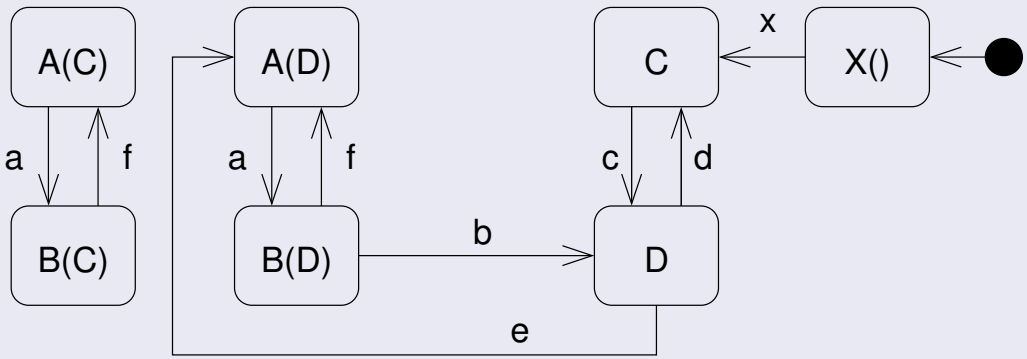
Lösung zu Beispiel 3:



Hauptidee: In den äußeren Zuständen merkt man sich, ob/aus welchem Unterzustand man den zusammengesetzten Zustand zuletzt verlassen hat. Dies führt zu zusätzlichen Zuständen.

Zustandsdiagramme: „Flachklopfen“

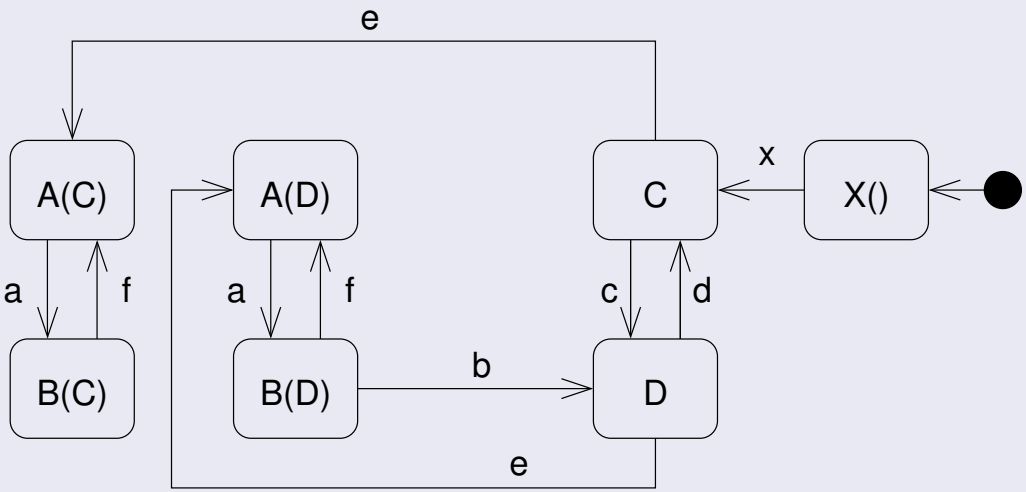
Lösung zu Beispiel 3:



Hauptidee: In den äußeren Zuständen merkt man sich, ob/aus welchem Unterzustand man den zusammengesetzten Zustand zuletzt verlassen hat. Dies führt zu zusätzlichen Zuständen.

Zustandsdiagramme: „Flachklopfen“

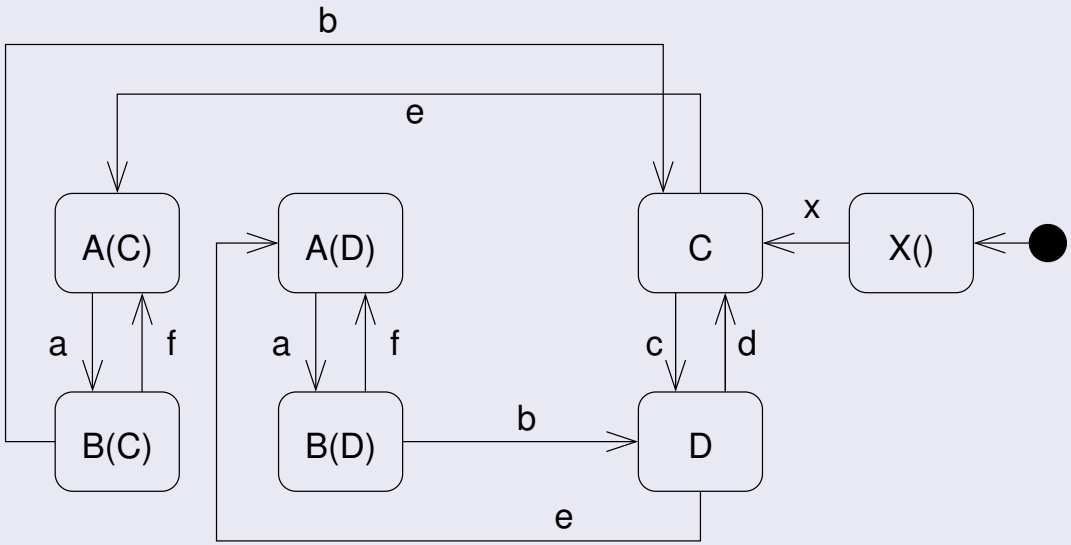
Lösung zu Beispiel 3:



Hauptidee: In den äußeren Zuständen merkt man sich, ob/aus welchem Unterzustand man den zusammengesetzten Zustand zuletzt verlassen hat. Dies führt zu zusätzlichen Zuständen.

Zustandsdiagramme: „Flachklopfen“

Lösung zu Beispiel 3:



Hauptidee: In den äußeren Zuständen merkt man sich, ob/aus welchem Unterzustand man den zusammengesetzten Zustand zuletzt verlassen hat. Dies führt zu zusätzlichen Zuständen.

Zustandsdiagramme: weitere Features

Weitere Features von UML-Zustandsdiagrammen:

- Unterscheidung zwischen verschiedenen Arten von Ereignissen: call event, signal event, change event, time event, any receive event
- Verzögern und Ignorieren von Ereignissen
- Entscheidungen und Kreuzungen
- Eintritts- und Austrittspunkte, Terminator
- Rahmen und Wiederverwendung von Zustandsdiagrammen

Außerdem: Generalisierung und Spezialisierung von Zustandsdiagrammen

Sequenzdiagramme

Sequenzdiagramme

Sequenzdiagramme (sequence diagrams) sind die bekanntesten Vertreter von Interaktionsdiagrammen in UML.

Sie dienen dazu, Kommunikation und Interaktion zwischen mehreren Kommunikationspartnern zu modellieren, und beruhen auf dem Basiskonzept der Interaktion:

Interaktion

Eine Interaktion ist das Zusammenspiel von mehreren Kommunikationspartnern.

Typische Beispiele: Versenden von Nachrichten, Datenaustausch, Methodenaufruf

Sequenzdiagramme

Sequenzdiagramme waren bereits vor Aufnahme in die UML unter dem Namen message sequence charts bekannt.

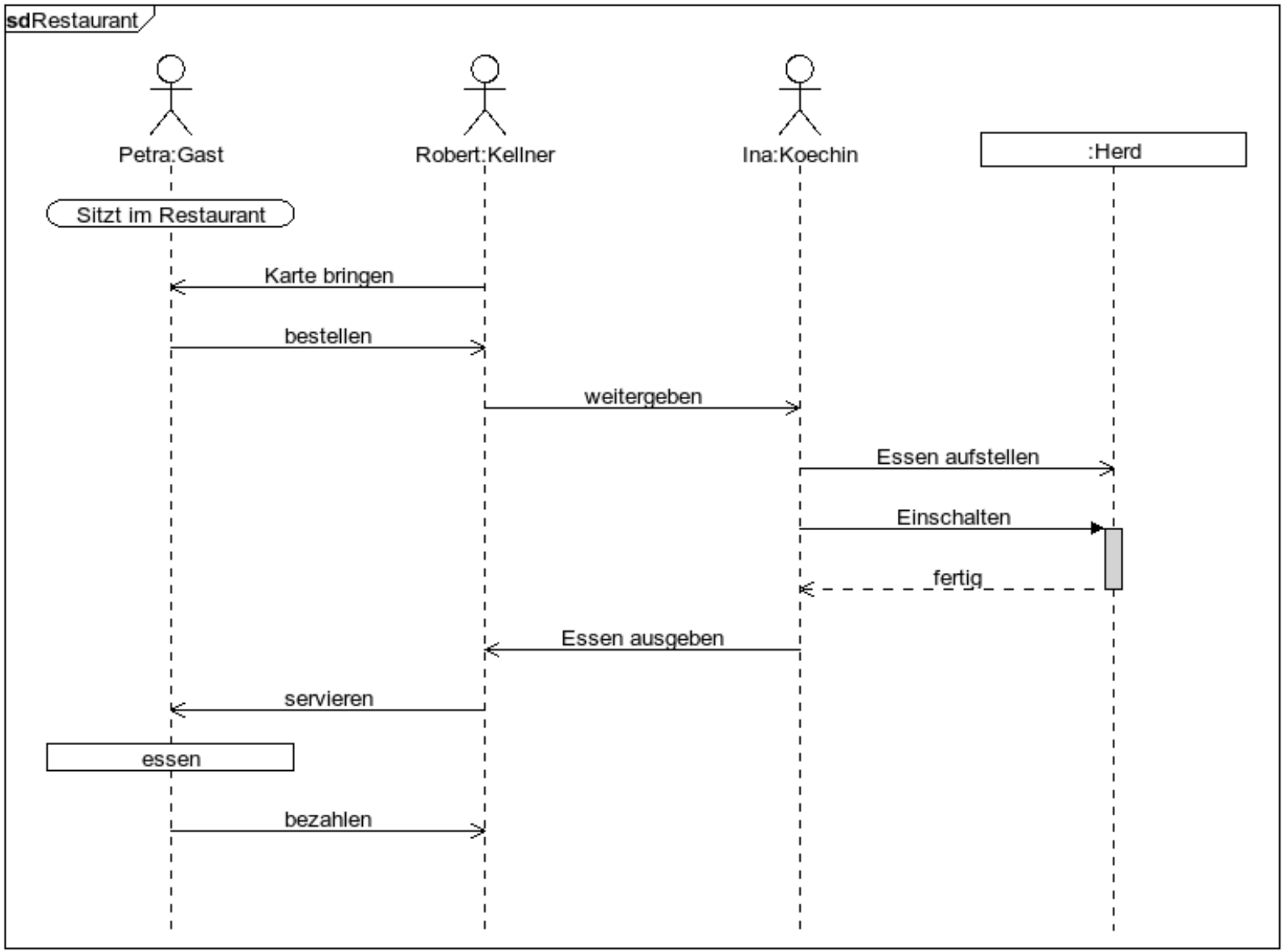
Im Gegensatz zu Aktivitätsdiagrammen oder Zustandsdiagrammen beschreiben sie im Allgemeinen nicht alle Abläufe eines Systems, sondern nur einen oder mehrere mögliche Abläufe.

Sequenzdiagramme

Sequenzdiagramme beschreiben Interaktionen in zwei Dimensionen:

- von links nach rechts: Anordnung der Kommunikationspartner als Lebenslinien. Oft wird der Partner, der den Ablauf initiiert, ganz links angegeben.
- von oben nach unten: Zeitachse

Sequenzdiagramme: Beispiel (Restaurant)

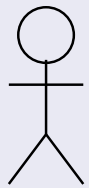


Sequenzdiagramme: Kommunikationspartner

Kommunikationspartner

Die Kommunikationspartner in einem Sequenzdiagramm werden ähnlich wie in Objektdiagrammen als Rechtecke notiert.

Manchmal werden die Rechtecke auch weggelassen. Menschliche Partner werden auch durch ein Strichmännchen symbolisiert:

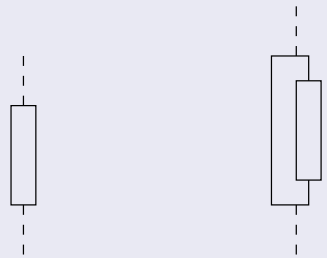


Von jedem Kommunikationspartner führt eine gestrichelte Lebenslinie nach unten.

Sequenzdiagramme: Kommunikationspartner

Ausführungsbalken

Aktivitäten eines Kommunikationspartners werden durch sogenannte Ausführungsbalken dargestellt.



Parallele Tätigkeiten eines Kommunikationspartners werden dabei durch übereinander liegende Ausführungsbalken beschrieben (siehe oben rechts).

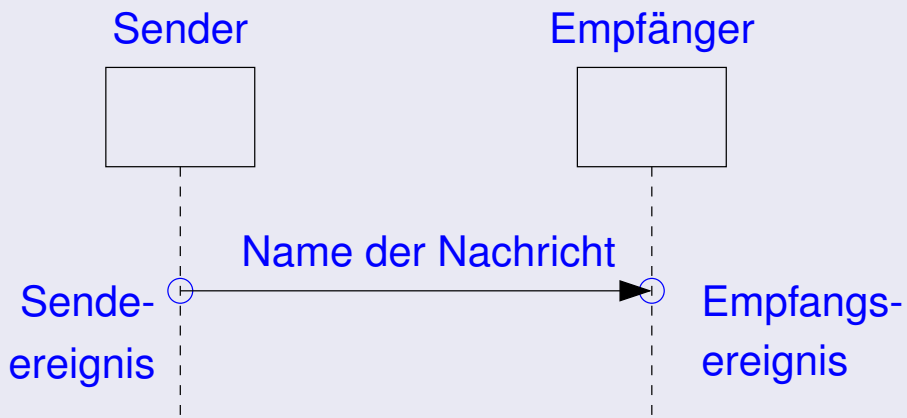
Während die Balken aktive Zeit anzeigen, symbolisieren die gestrichelten Linien passive Zeit.

Sequenzdiagramme: Nachrichten

Nachrichten

Die Nachrichten beschreiben die Kommunikationen bzw. Interaktionen der Kommunikationspartner und werden durch Pfeile dargestellt. Eine Nachricht hat einen Sender und einen Empfänger.

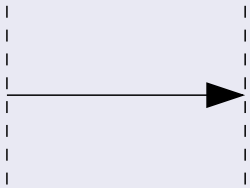
Die Stellen, an denen die Pfeile auf den Lebenslinien auftreffen, nennt man auch Sendeereignis und Empfangsereignis.



Sequenzdiagramme: Nachrichten

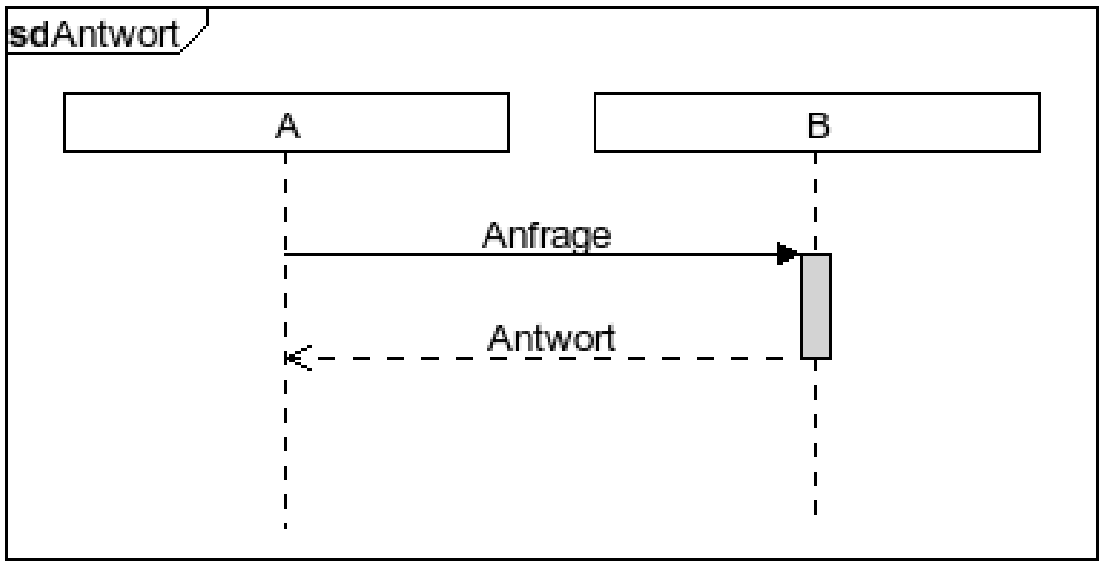
Synchrone Nachrichten

Bei synchroner Kommunikation warten Sender und Empfänger aufeinander. Der Sender macht erst dann weiter, wenn er weiß, dass der Empfänger die Nachricht erhalten hat. Solche Kommunikation wird unter Verwendung einer schwarzen ausgefüllten Pfeilspitze dargestellt.



Sequenzdiagramme: Nachrichten

Um zu signalisieren, dass der Empfänger die Nachricht erhalten hat, bzw. die zugehörige Aktion abgeschlossen wurde, wird eine Antwort gesendet. Diese ist als gestrichelter Pfeil dargestellt.



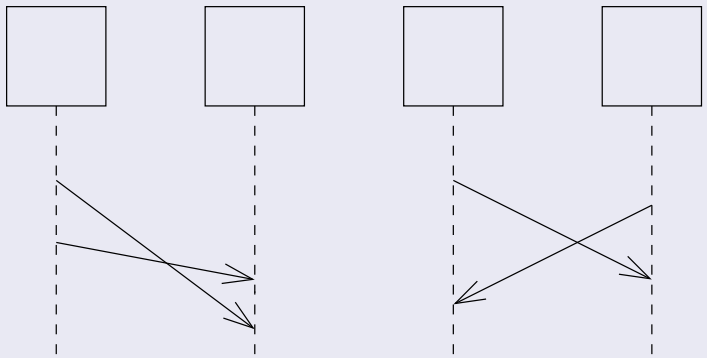
Sequenzdiagramme: Nachrichten

Asynchrone Nachrichten

Bei asynchroner Kommunikation wartet der Sender nicht darauf, dass der Empfänger die Nachricht erhalten hat. Er arbeitet einfach weiter. Solche Kommunikation wird durch eine einfache Pfeilspitze dargestellt.

Sequenzdiagramme: Nachrichten

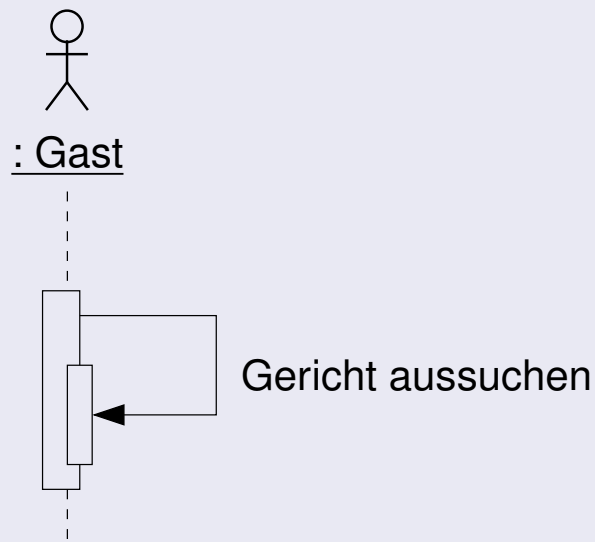
Bei asynchroner Kommunikation (aber nicht bei synchroner Kommunikation) kann auch der Fall eintreten, dass sich Nachrichten überholen oder kreuzen.



Das Überholen von Nachrichten (oben links) ist nur dann nicht möglich, wenn man explizit einen FIFO-Kanal fordert.

Sequenzdiagramme: Nachrichten

Es ist möglich, eine Nachricht an sich selbst zu schicken.



Sequenzdiagramme: Äquivalenz

Äquivalenz von Sequenzdiagrammen

Zwei Sequenzdiagramme sind äquivalent, wenn sie die gleichen Ereignisse enthalten und die Reihenfolge der Ereignisse identisch ist.

Dabei können die Diagramme durchaus verschieden gezeichnet sein (andere Reihenfolge der Kommunikationspartner, verschiedene Abstände zwischen den Ereignissen, . . .).

Durch die Bestimmung der Reihenfolge der Ereignisse kann die Äquivalenz zweier Diagramme nachgewiesen bzw. widerlegt werden.

Sequenzdiagramme: Interaktions-Operatoren

Bisher haben wir gesehen, wie man mit einem Sequenzdiagramm einen möglichen Ablauf beschreiben kann. In manchen Fällen möchte man jedoch mehrere (vielleicht sogar alle) Abläufe beschreiben.

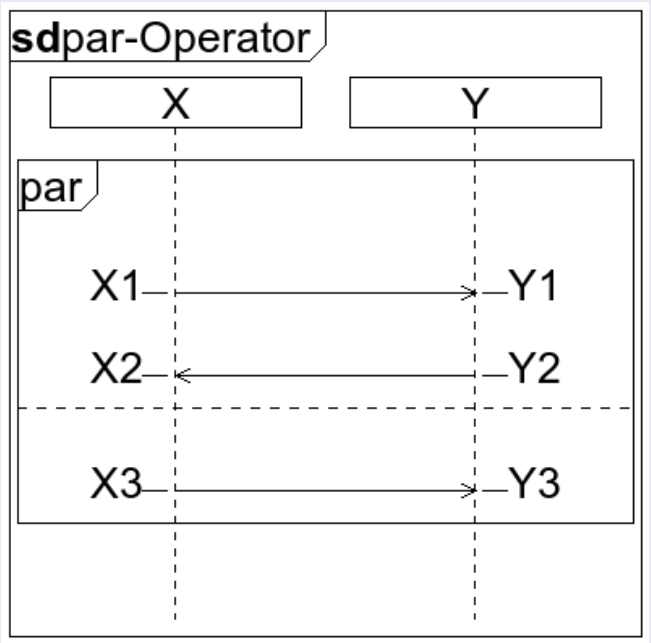
Dazu gibt es die Möglichkeit, kombinierte Fragmente zu verwenden, bei denen mehrere (Interaktions-)Operanden (= Teil-Sequenzdiagramme) mit Hilfe von Interaktions-Operatoren zusammengesetzt werden.

Wir betrachten im Folgenden einige dieser Interaktions-Operatoren.

Sequenzdiagramme: par-Operator

Interaktionsoperator par (Parallelität)

Hier sind die Operanden in beliebiger Reihenfolge ausführbar. Die Reihenfolge der Ereignisse in den Operanden muss aber gewahrt werden. Ansonsten gibt es keine Bedingungen.

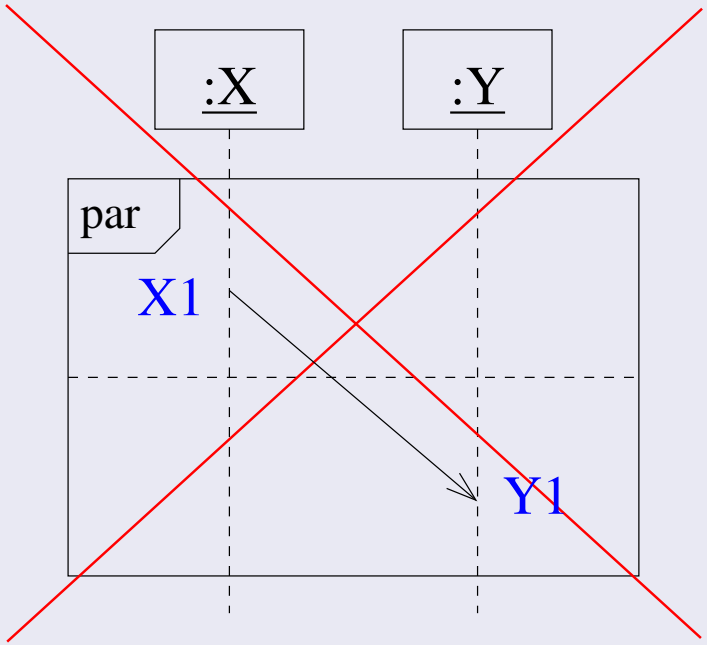


Dabei wird der Operator par links oben in der Ecke angegeben.

Eine gestrichelte waagerechte Linie trennt die verschiedenen Operanden.

Sequenzdiagramme: par-Operator

Nachrichten, die von einem Operanden in einen anderen laufen, sind nicht zugelassen.



Sequenzdiagramme: par-Operator

Ordnung auf den Ereignissen:

- X1 $<$ Y1 $<$ Y2 $<$ X2 (Operand 1)
- X3 $<$ Y3 (Operand 2)

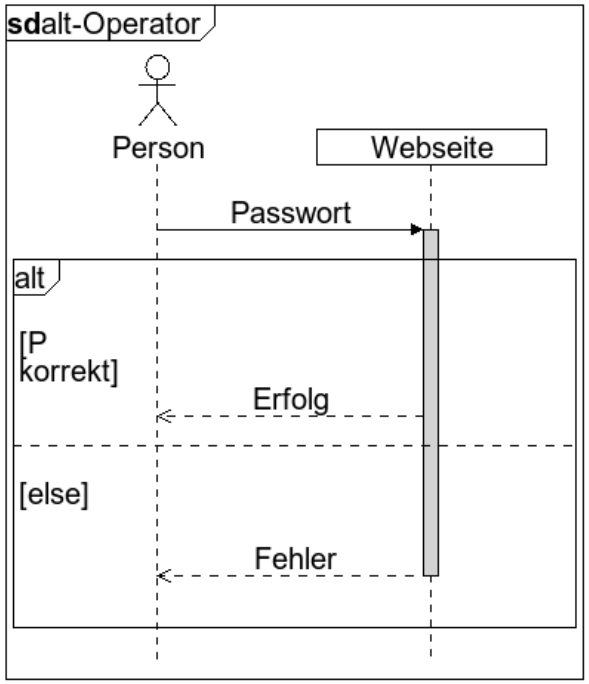
Ansonsten gibt es keine weiteren Einschränkungen.

Dieses Sequenzdiagramm beschreibt insgesamt fünfzehn Abläufe, beispielsweise:

- X1, Y1, X3, Y3, Y2, X2
- X3, X1, Y1, Y2, X2, Y3
- ...

Sequenzdiagramme: alt-Operator

Sequenzdiagramme bieten auch die Möglichkeit, alternative Nachrichtenflüsse darzustellen. Zu diesem Zweck wird der alt-Operator verwendet.
Ein möglicher Anwendungsfall ist die unterschiedliche Rückgabe an eine Person bei der Eingabe eines Passwortes.



Sequenzdiagramme: alt-Operator

Dabei ist es nötig, die Eintrittsfälle für die Alternativen durch Guards zu definieren.

Es ist ebenfalls möglich, eine beliebige Anzahl von Alternativen zu modellieren. Diese sollten aber in einem Kontext stehen und nicht unabhängig voneinander sein. Zudem müssen alle Guards disjunkt sein.

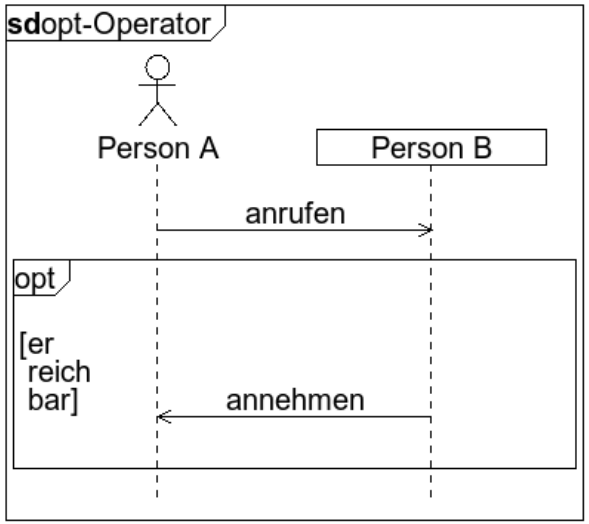
Durch die Verschachtelung der Blöcke ist es aber möglich, auch unabhängige Alternativen zu modellieren.

Es gibt keine Beschränkung der Anzahl Nachrichten, die in einem Block enthalten sein dürfen.

Wichtig: Bei der Nutzung eines alt-Operators müssen alle Fälle abgedeckt sein. Dies kann zum Beispiel durch einen Fall *else* beschrieben werden.

Sequenzdiagramme: opt-Operator

Der opt-Operator dient dazu, optionale Nachrichtenflüsse zu definieren. Im Gegensatz zum bereits vorgestellten alt-Operator ist es damit möglich, diesen Block auch zu überspringen, falls die Nachricht nicht gesendet werden soll.
Ein Beispiel ist ein Telefonanruf, bei dem nur abgehoben wird, wenn die andere Person erreichbar ist.



Sequenzdiagramme: opt-Operator

Auch bei einem opt-Operator ist es nötig, durch Guards die Eintrittsbedingung zu definieren. Ist diese nicht erfüllt, wird der Teil innerhalb des Operators übersprungen.

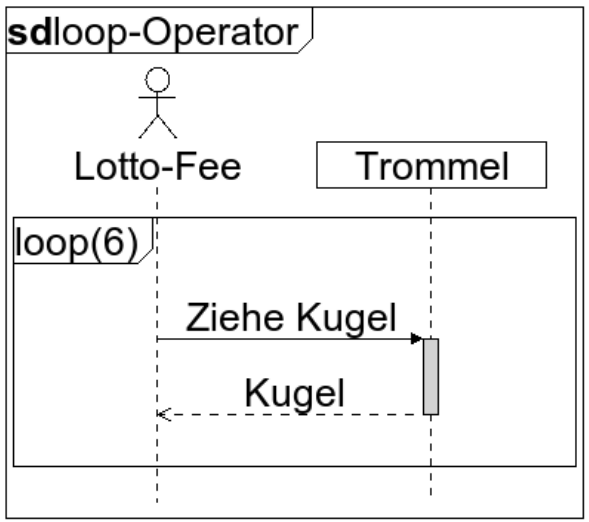
Im Gegensatz zum alt-Operator gibt es keine Möglichkeit, verschiedene Fälle in einem Operator zu definieren. Für mehrere verschiedene Bedingungen muss daher ein eigener opt-Operator eingeführt werden. Innerhalb eines Operators kann ein beliebiger Nachrichtenfluss mit einer beliebigen Anzahl an Nachrichten stattfinden.

Auch dieser Operator kann beliebig verschachtelt und mit anderen Operatoren kombiniert werden.

Sequenzdiagramme: loop-Operator

Der loop-Operator ermöglicht es, einen definierten Nachrichtenfluss zu wiederholen.

Ein Beispiel ist die Ziehung der Lottozahlen *6 aus 49*, bei der die Lottofee 6 mal eine Kugel aus der Trommel zieht.



Sequenzdiagramme: loop-Operator

In dem gezeigten Beispiel ist eine feste Anzahl von Abläufen gegeben. Es ist auch möglich, eine minimale und eine maximale Zahl der Form loop(min,max) zu definieren, wobei „*“ für eine beliebige Zahl einschließlich 0 stehen kann. In diesem Fall ist normalerweise zusätzlich ein Guard angegeben, der angibt, wann abgebrochen wird. Ist die min-Anzahl durchlaufen und die Bedingung erfüllt, wird die Schleife verlassen.

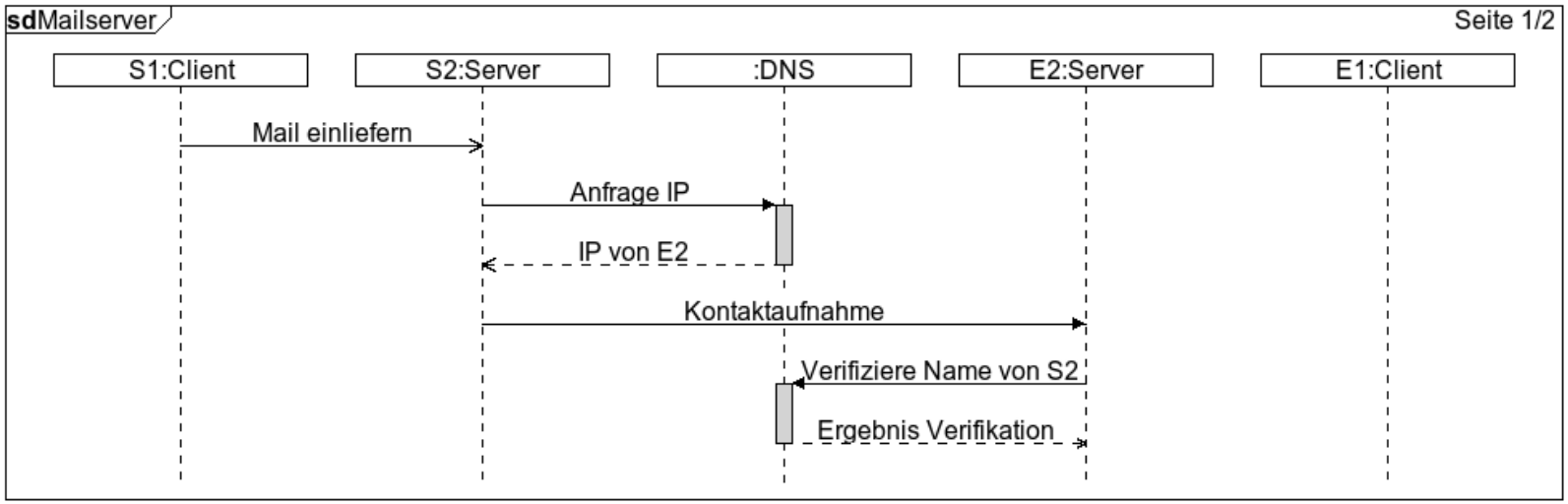
Auch der loop-Operator ist nicht in der Lage, verschiedene Fälle zu definieren. Dies geschieht, indem innerhalb die bereits bekannten Operatoren opt und alt verwendet werden.

Sequenzdiagramme: Mailserver (Beispiel)

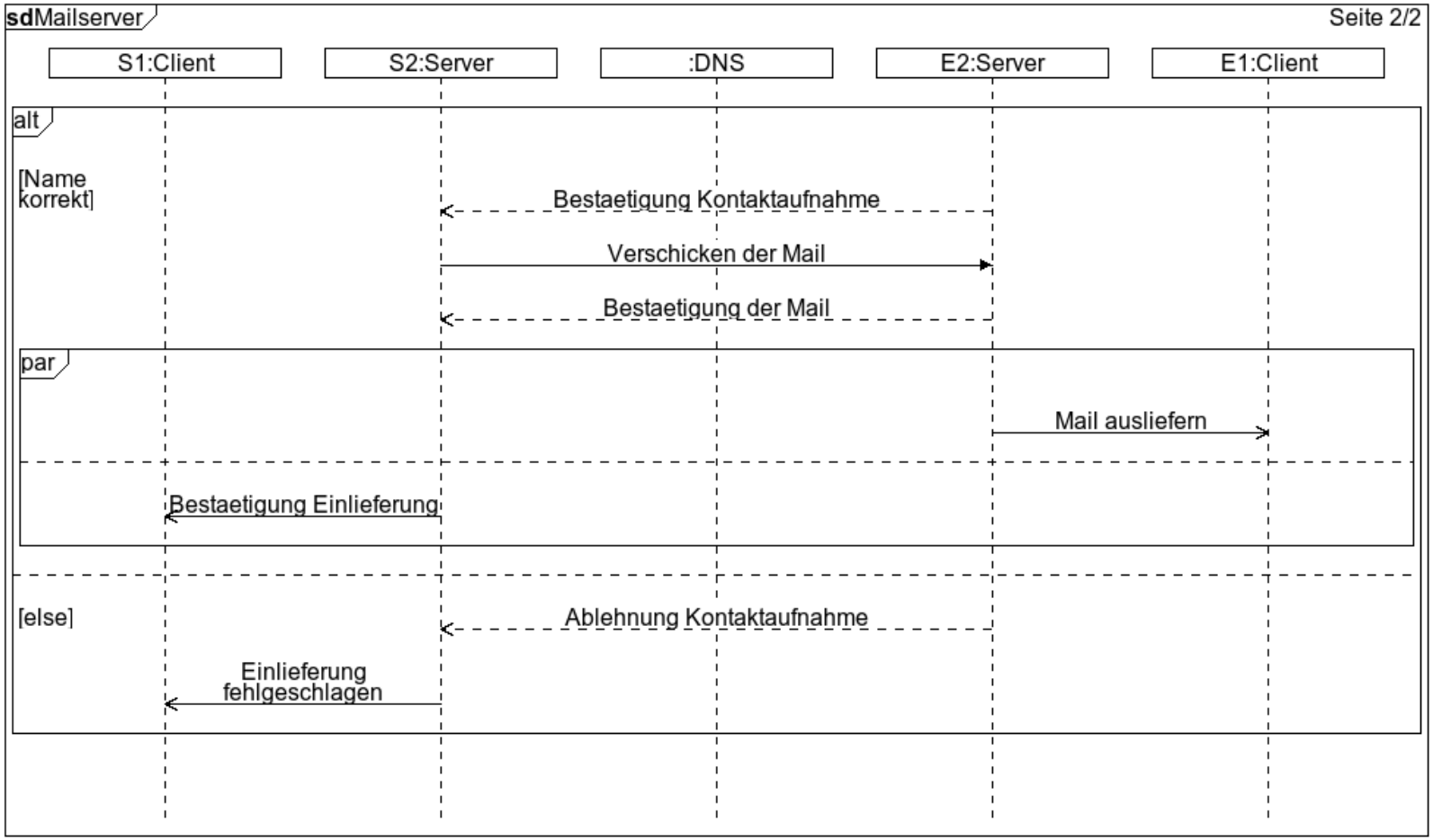
Beispiel: Wir modellieren ein Protokoll, bei dem ein Client-Rechner S eine E-Mail an einen anderen Client R verschickt.

Weitere Beteiligte sind der Mail-Server von S, der Mail-Server von R und der DNS-Server, der benötigt wird, um Mail-Adressen in die IP-Adressen des entsprechenden Servers umzuwandeln (DNS = domain name system).

Sequenzdiagramme: Mailserver (Beispiel)



Sequenzdiagramme: Mailserver (Beispiel)



Sequenzdiagramme: Mailserver (Beispiel)

Bemerkung: Dieses Sequenzdiagramm ist an den Ablauf im SMTP-Protokoll angelehnt (SMTP = send mail transport protocol), ist jedoch erheblich vereinfacht.

Sequenzdiagramme zu vielen TCP/IP-Netzwerkprotokollen findet man unter:

`http://www.eventhelix.com/Realtimemantra/Networking/`

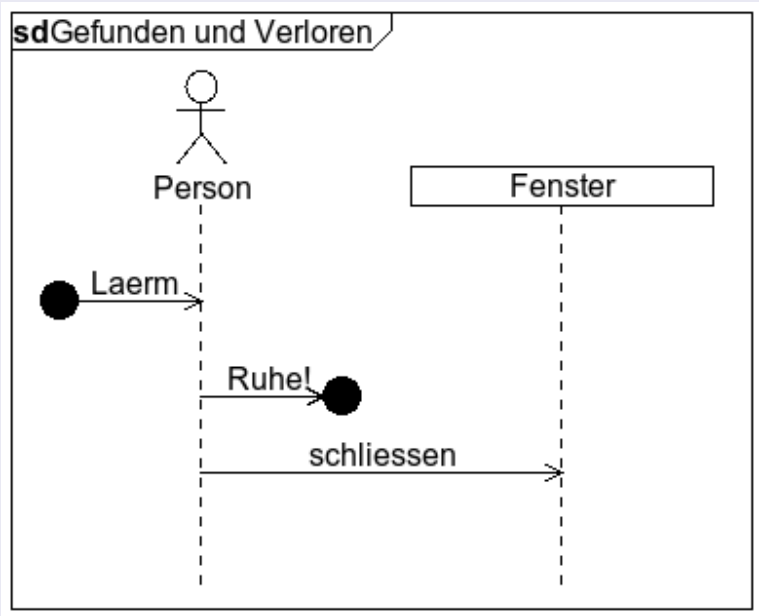
Sequenzdiagramme: weitere Arten von Nachrichten

Es gibt noch einige weitere Arten von Nachrichten:

Gefundene und verlorene Nachrichten

Bei gefundenen und verlorenen Nachrichten werden das Sende- bzw. das Empfangsereignis nicht explizit modelliert. Die Nachrichten tauchen quasi aus der Umgebung auf und verschwinden wieder dorthin.

Solche Nachrichten werden benötigt, wenn die entsprechenden Kommunikationspartner nicht mitmodelliert werden.

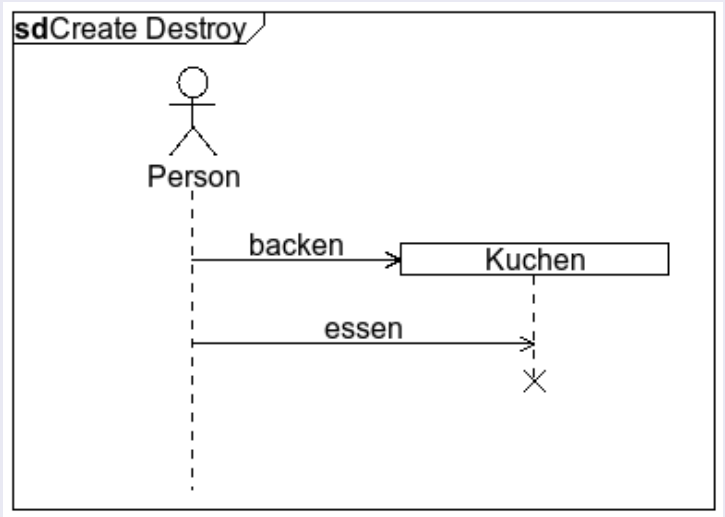


Sequenzdiagramme

Erzeugung und Löschung von Objekten

Außerdem kann es passieren, dass Objekte nicht während des ganzen Ablaufs zur Verfügung stehen. Sie können während des Ablaufs dynamisch erzeugt und wieder gelöscht werden.

Dies erfolgt zumeist durch sogenannte Erzeugungs- und Löschnachrichten und wird folgendermaßen dargestellt:



Anwendungsfalldiagramme

Anwendungsfalldiagramme

In frühen Stadien der Entwicklung und bei der Kommunikation mit dem Auftraggeber spielen auch Anwendungsfalldiagramme (engl. use case diagrams) eine große Rolle.

Anwendungsfalldiagramme

In frühen Stadien der Entwicklung und bei der Kommunikation mit dem Auftraggeber spielen auch Anwendungsfalldiagramme (engl. use case diagrams) eine große Rolle.

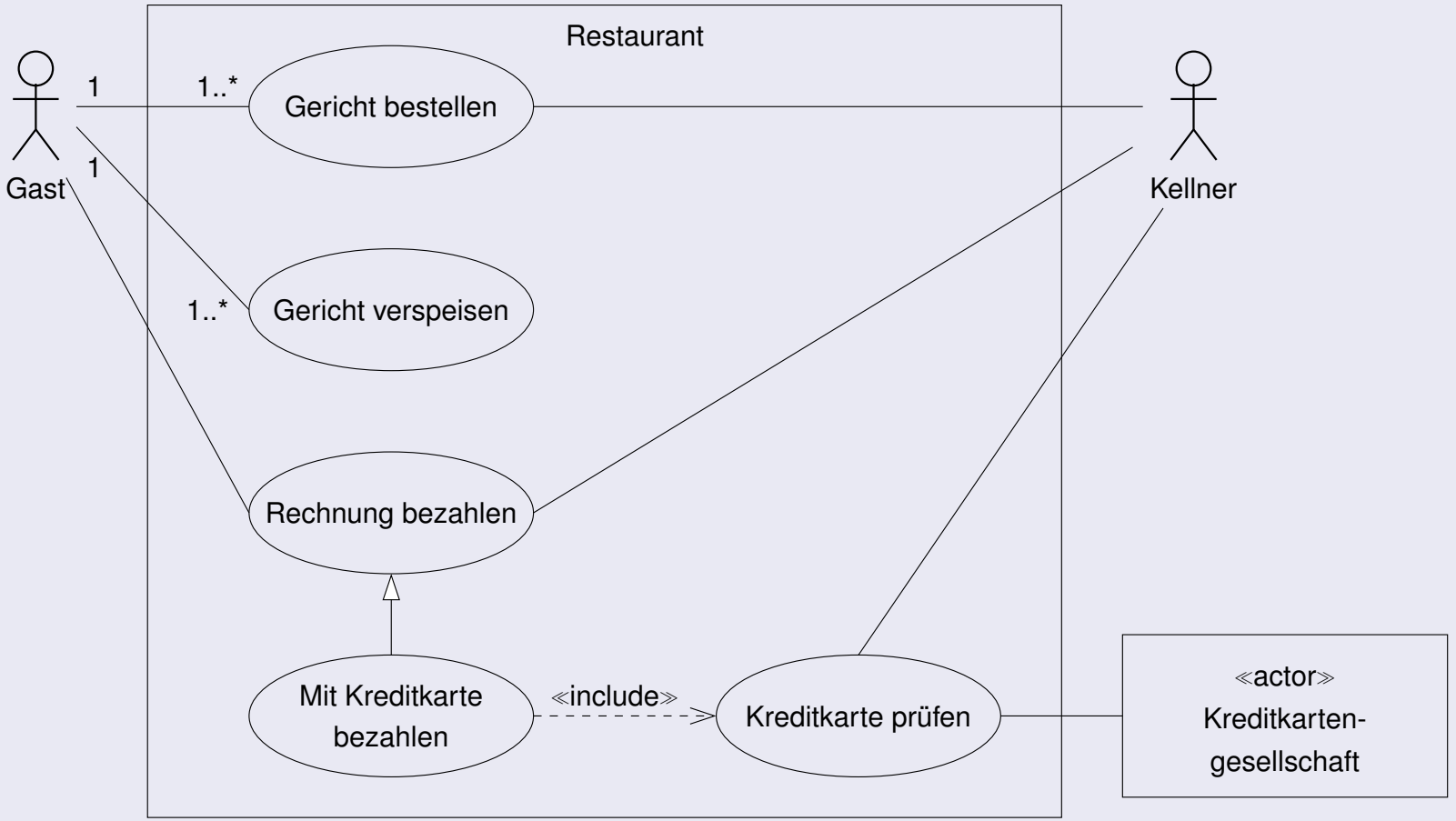
Anwendungsfalldiagramme modellieren die Funktionalität des Systems,

- auf einem hohen Abstraktionsniveau;
- aus der Black-Box-Sicht des Anwenders (das heißt, nur das von außen Sichtbare soll beschrieben werden, nicht die interne Realisierung);
- durch Spezifikation der Schnittstellen.

Die Anwender bzw. Nutzer tauchen als sogenannte Akteure in den Diagrammen auf.

Anwendungsfalldiagramme

Beispiel: Restaurant



Anwendungsfalldiagramme

Anwendungsfalldiagramme bestehen aus im Folgenden beschriebenen Komponenten.

Systemgrenze

Die Systemgrenze ist ein Rechteck, das beschreibt, was sich außerhalb und was sich innerhalb des zu erstellenden Systems befindet.

Anwendungsfalldiagramme

Akteur

Ein Akteur ist ein Typ oder eine Rolle, die ein externer Benutzer oder ein externes System während der Interaktion mit dem System einnimmt. Menschliche Akteure werden durch Strichmännchen symbolisiert, andere Akteure durch ein Rechteck, das mit dem Schlüsselwort «actor» gekennzeichnet ist.

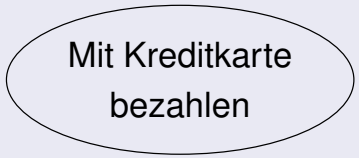
Gast

«actor»
Kreditkarten-
gesellschaft

Anwendungsfalldiagramme

Anwendungsfall

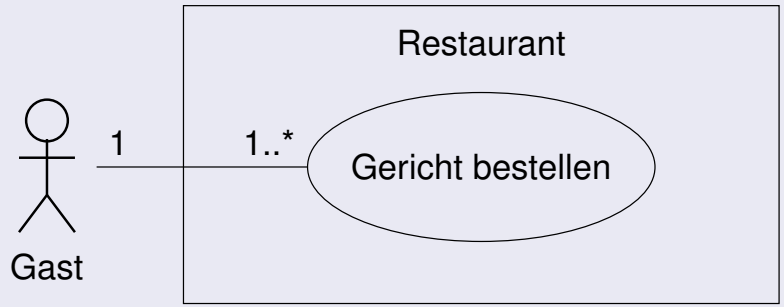
Ein Anwendungsfall ist eine Menge von Interaktionsfolgen, die von dem System bereitgestellt werden und die einen Nutzen für einen oder mehrere Akteure bringen. (Das heißt, es handelt sich dabei um eine Art „Service“ des Systems.) Ein Anwendungsfall wird durch eine Ellipse dargestellt.



Anwendungsfalldiagramme

Assoziation

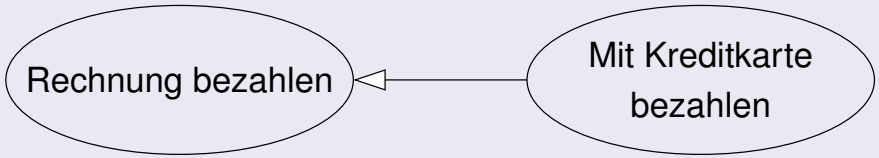
Wie bei Klassendiagrammen gibt es Assoziationen, vor allem zwischen Akteuren und Anwendungsfällen. (Welcher Akteur ist an welchem Anwendungsfall beteiligt?)
Assoziationen dürfen die üblichen Beschriftungen besitzen (Multiplizitäten etc.).



Anwendungsfalldiagramme

Generalisierung/Spezialisierung

Anwendungsfälle können andere Anwendungsfälle spezialisieren, das heißt, sie können von ihnen erben. Dabei werden – wie bei Klassen – auch alle Assoziationen geerbt.

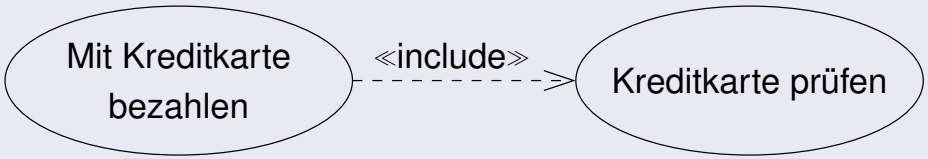


Auch Spezialisierungsbeziehungen zwischen Akteuren sind möglich.

Anwendungsfalldiagramme

Include-Beziehung

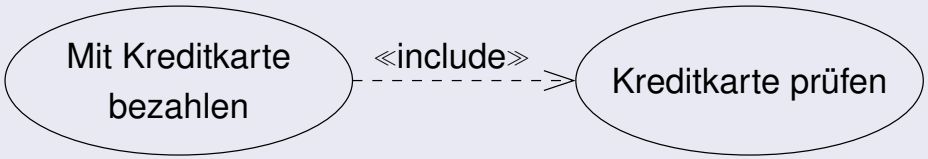
Bei einer Include-Beziehung wird modelliert, dass ein Anwendungsfall die Funktionalität eines anderen Anwendungsfalles auf jeden Fall beinhaltet. Das heißt, der zweite Anwendungsfall wird immer als eine Art „Unterprozedur“ aufgerufen.



Anwendungsfalldiagramme

Include-Beziehung

Bei einer Include-Beziehung wird modelliert, dass ein Anwendungsfall die Funktionalität eines anderen Anwendungsfalles auf jeden Fall beinhaltet. Das heißt, der zweite Anwendungsfall wird immer als eine Art „Unterprozedur“ aufgerufen.



Es gibt auch sogenannte Extend-Beziehungen, bei denen ein Anwendungsfall nur unter bestimmten Bedingungen in einen anderen Anwendungsfall eingebunden wird.

Anwendungsfalldiagramme

Bemerkungen:

- Anwendungsfalldiagramme sollten nicht zu viele Details enthalten.
- Sie sind ein einfaches Mittel, um Anwenderwünsche zu diskutieren, und sollten nur eine grobe Sicht auf die Funktionalität des Systems darstellen.
- Bei Bedarf müssen bestimmte Anwendungsfälle dann noch textuell oder mit Hilfe anderer UML-Diagramme genauer beschrieben werden.

Weitere UML-Diagramme

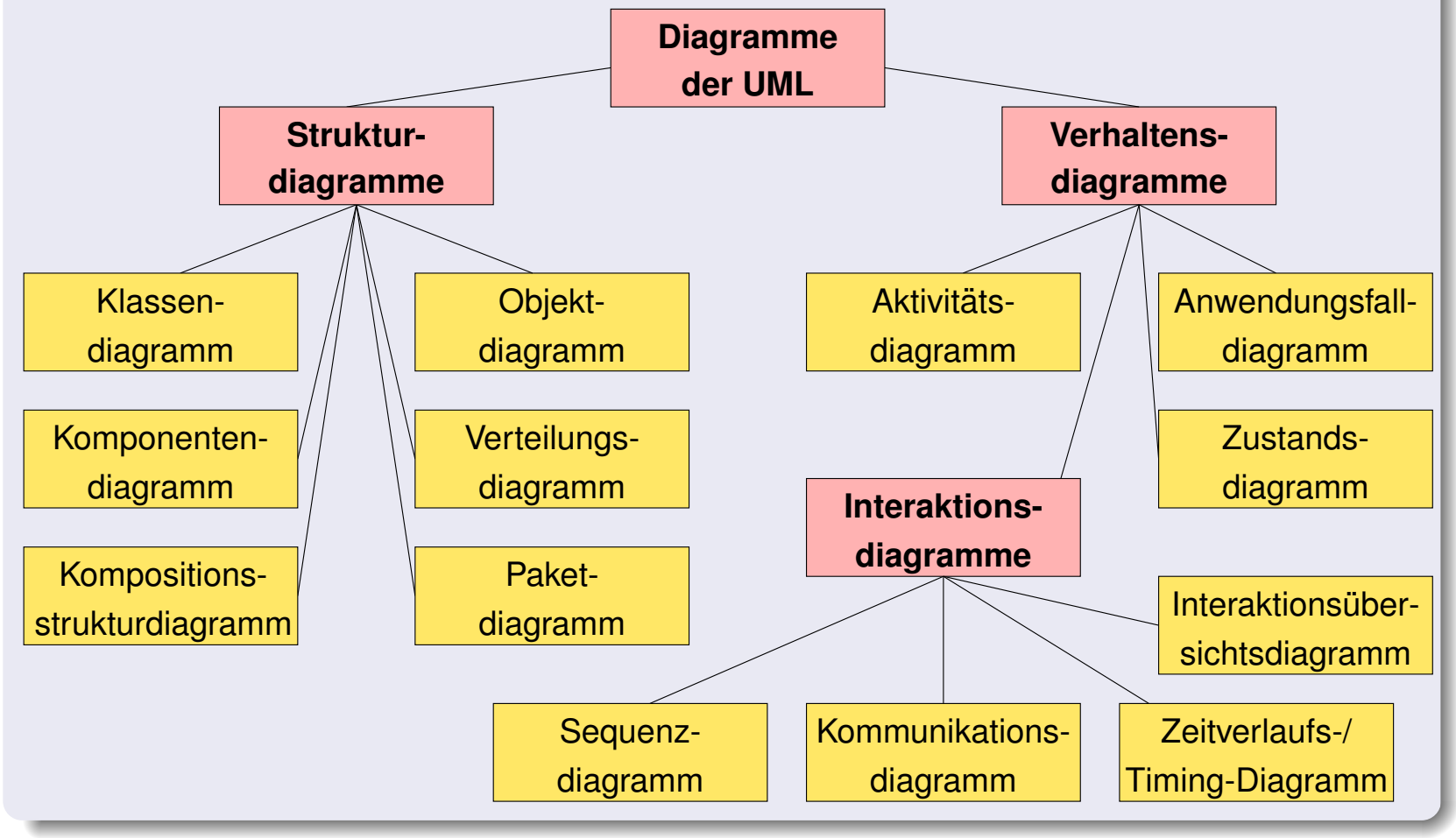
Überblick über weitere UML-Diagramme

Wir schließen die Vorlesung mit einem kurzen Überblick über die noch fehlenden Typen von UML-Diagrammen ab.

Zuletzt gibt es dann noch ein paar Abschlussbemerkungen.

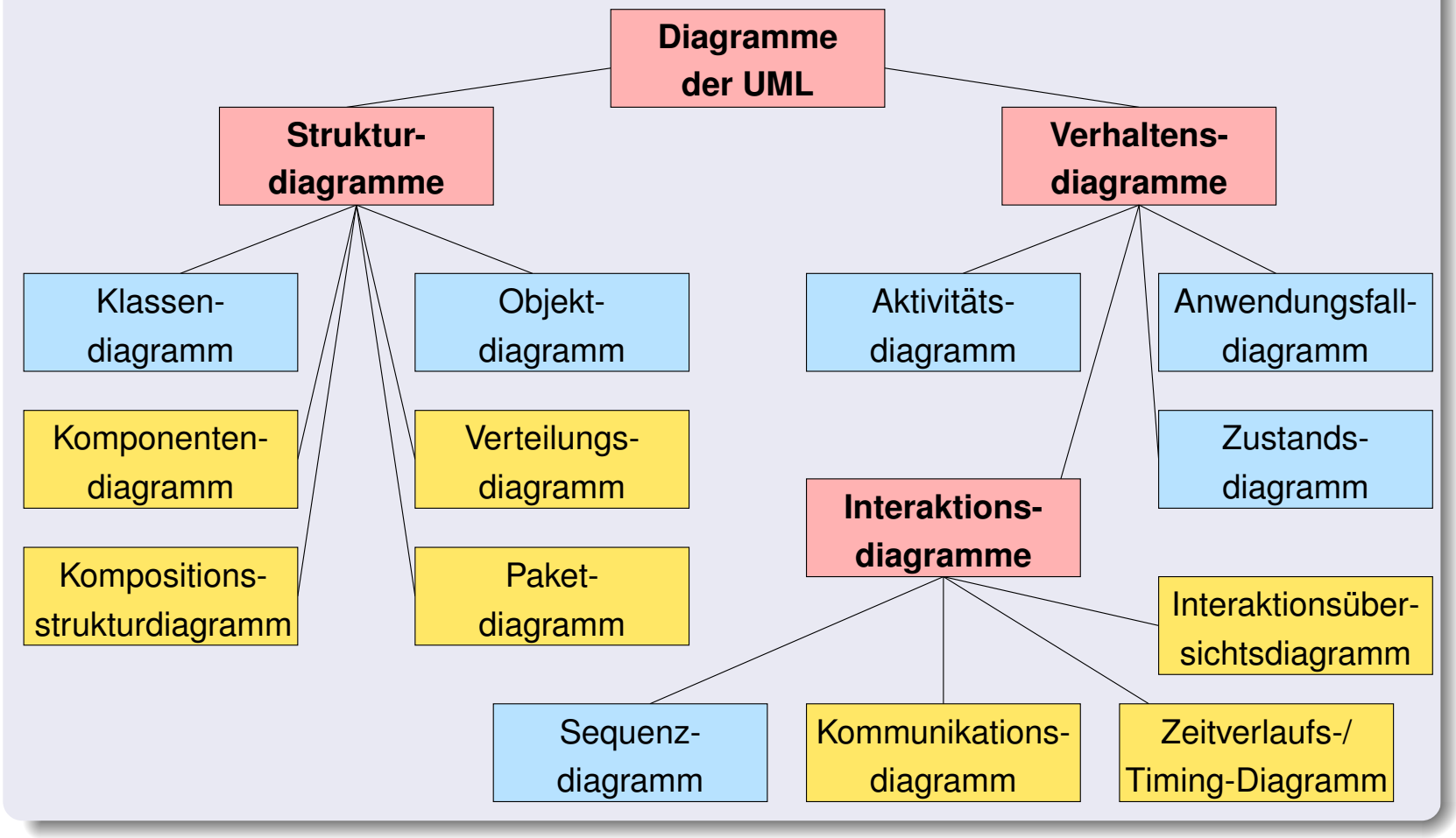
Überblick über weitere UML-Diagramme

UML-Diagramme ▸ Liste



Überblick über weitere UML-Diagramme

UML-Diagramme (blau: bereits behandelte Diagramme)



Überblick über weitere UML-Diagramme

Wir beginnen zunächst mit den drei noch fehlenden Arten von Interaktionsdiagrammen:

- Kommunikationsdiagramme
- Timing-Diagramme bzw. Zeitverlaufsdiagramme
- Interaktionsübersichtsdiagramme

Kommunikationsdiagramme

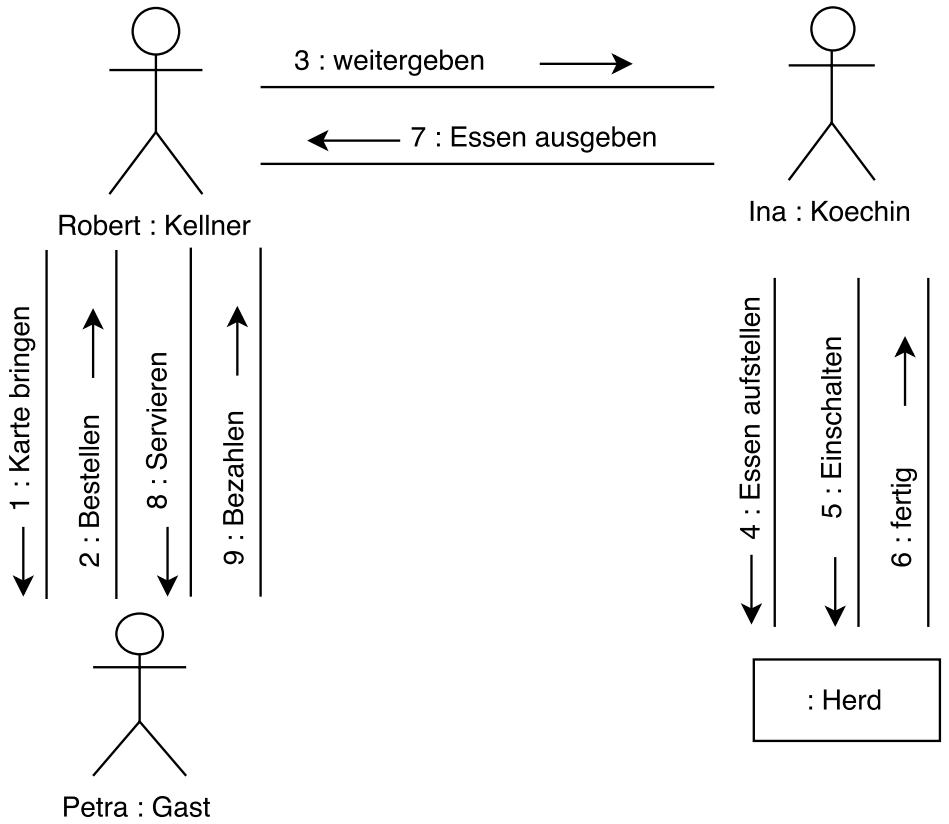
Kommunikationsdiagramme enthalten dieselbe Information wie Sequenzdiagramme, werden jedoch anders dargestellt.

Während bei Sequenzdiagrammen der Fokus eher auf dem zeitlichen Ablauf liegt, heben Kommunikationsdiagramme eher die Kommunikationsbeziehungen der Teilnehmer hervor.

Kommunikationsdiagramme gehören – genau wie Sequenzdiagramme – zur Klasse der Interaktionsdiagramme.

Kommunikationsdiagramme

Das Sequenzdiagramm Restaurantbesuch ▸ Diagramm wird folgendermaßen als Kommunikationsdiagramm dargestellt:



Kommunikationsdiagramme

Dabei werden die Kommunikationspartner wie bisher durch Rechtecke oder Strichmännchen dargestellt. Es wird jedoch kein zeitlicher Ablauf mehr dargestellt.

Die Interaktionen bzw. Nachrichten werden durch Linien notiert, an denen die Namen der Nachrichten und die Senderichtung ($\rightarrow$) stehen.

Die Nummerierung der Nachrichten (1, 2, 3, . . .) gibt die Reihenfolge an. Durch Buchstaben hinter den Nummern (2a, 2b, . . .) beschreibt man parallele Nachrichten, die in beliebiger Reihenfolge angeordnet sein können.

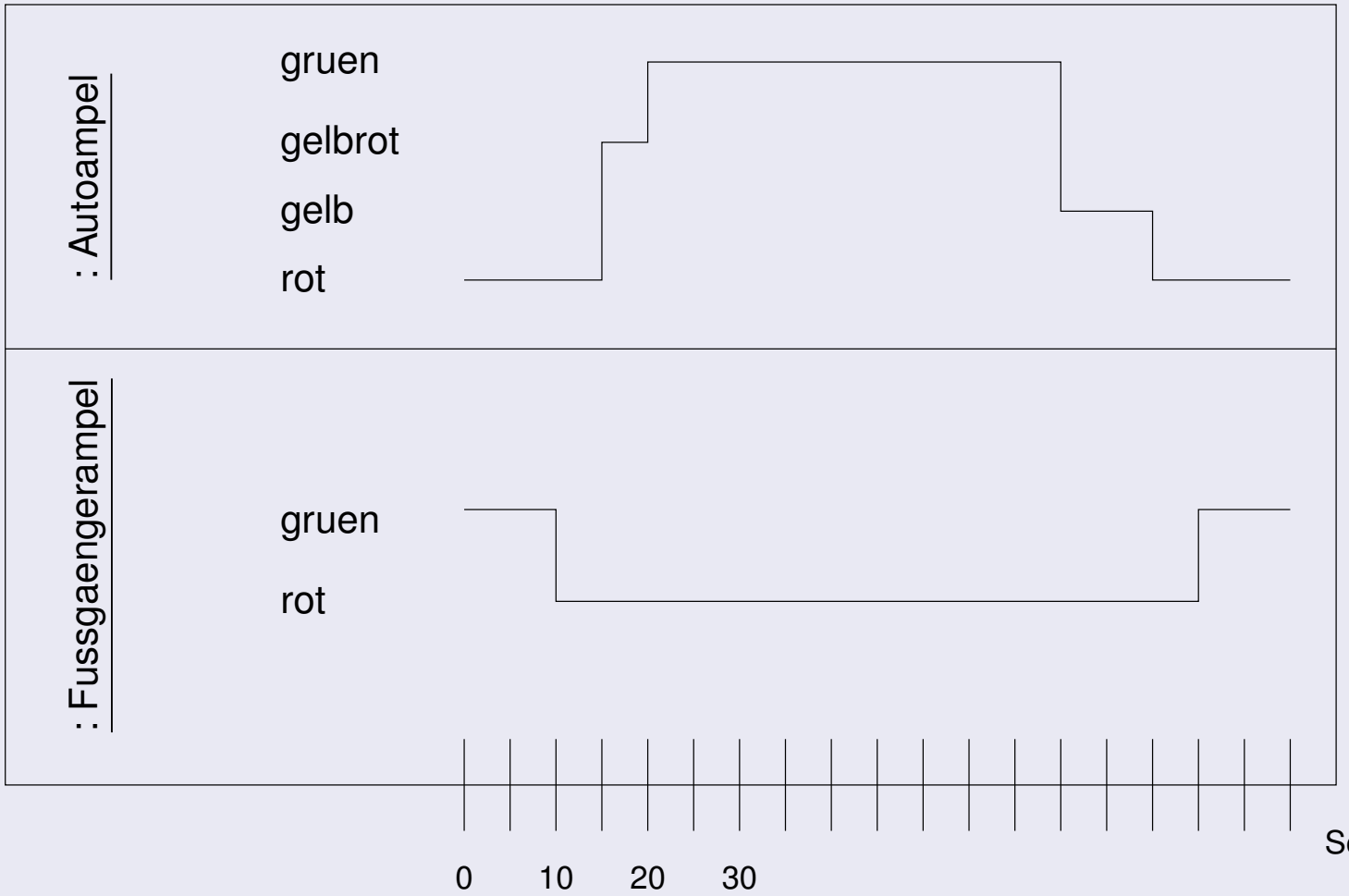
Timing-Diagramme

Timing-Diagramme sind in der Elektrotechnik weit verbreitet und zeigen an, zu welchem Zeitpunkt welcher Kommunikationspartner welchen Zustand einnimmt.

Dabei wird von links nach rechts (horizontal) die Zeit aufgetragen und von oben nach unten werden die Kommunikationspartner und deren Zustände aufgetragen.

Timing-Diagramme

Beispiel: Ampelschaltung



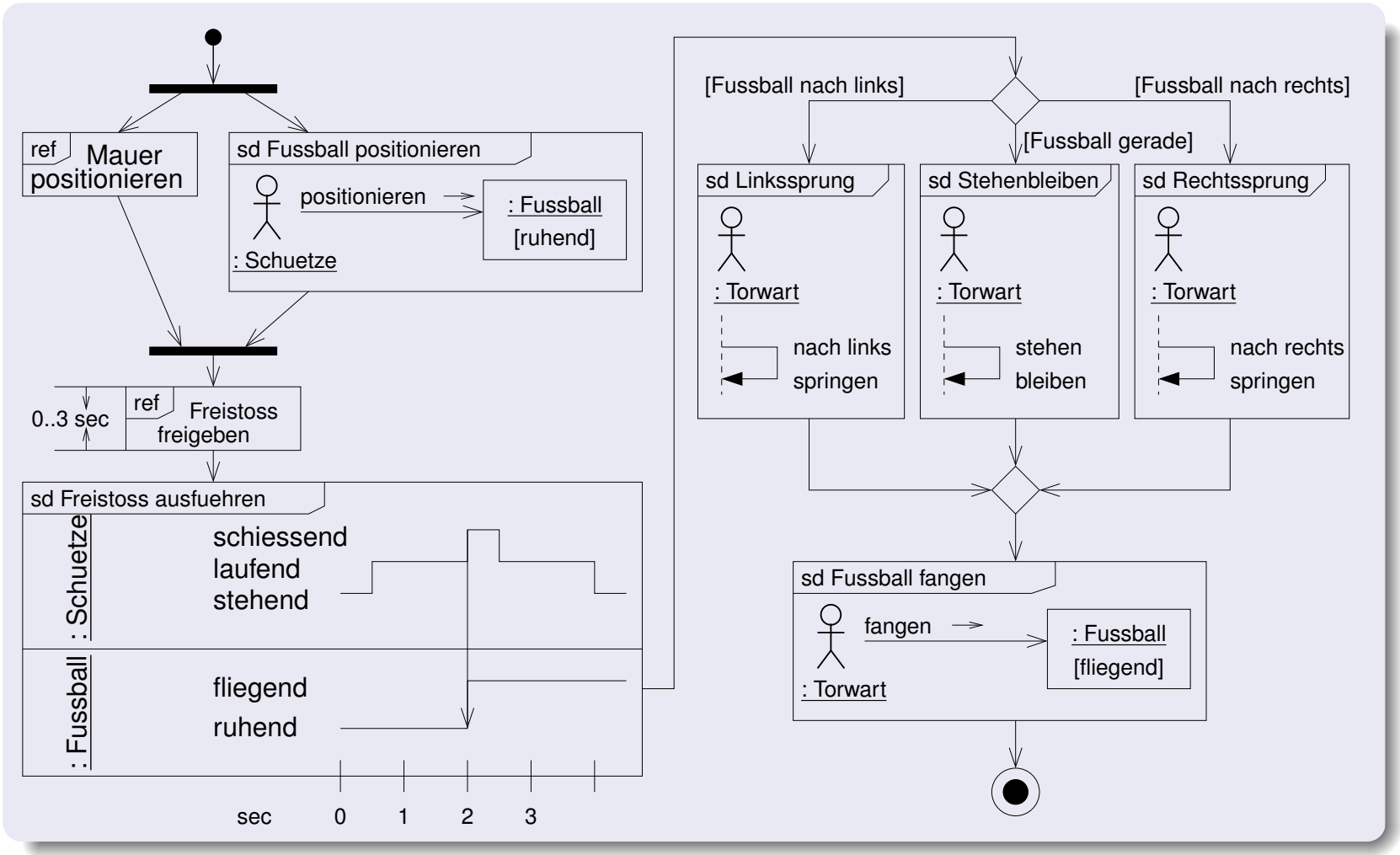
Interaktionsübersichtsdiagramme

Interaktionsübersichtsdiagramme sind im Wesentlichen Aktivitätsdiagramme, die – anstatt der Aktionen – hierarchisch weitere Interaktionsdiagramme (Sequenzdiagramme, Kommunikationsdiagramme, Timing-Diagramme) enthalten können.

Sie werden eingesetzt, wenn es eine größere Menge verschiedener Arten von Aktionen gibt, über die man sonst nur schwer den Überblick behalten kann.

Als Beispiel betrachten wir ein Interaktionsübersichtsdiagramm, das den Ablauf eines Freistoßes in einem Fußballspiel modelliert.

Interaktionsübersichtsdiagramme



Interaktionsübersichtsdiagramme

Bemerkungen:

- Interaktionsreferenzen (Schlüsselwort ref) werden verwendet, um an anderer Stelle definierte Interaktionsdiagramme wiederzuverwenden.
- Anstatt der Aktionen wie in Aktivitätsdiagrammen werden ganze Interaktionsdiagramme verwendet, die alle sd als Diagrammtyp für Interaktionsdiagramme erhalten.

Überblick über weitere UML-Diagramme

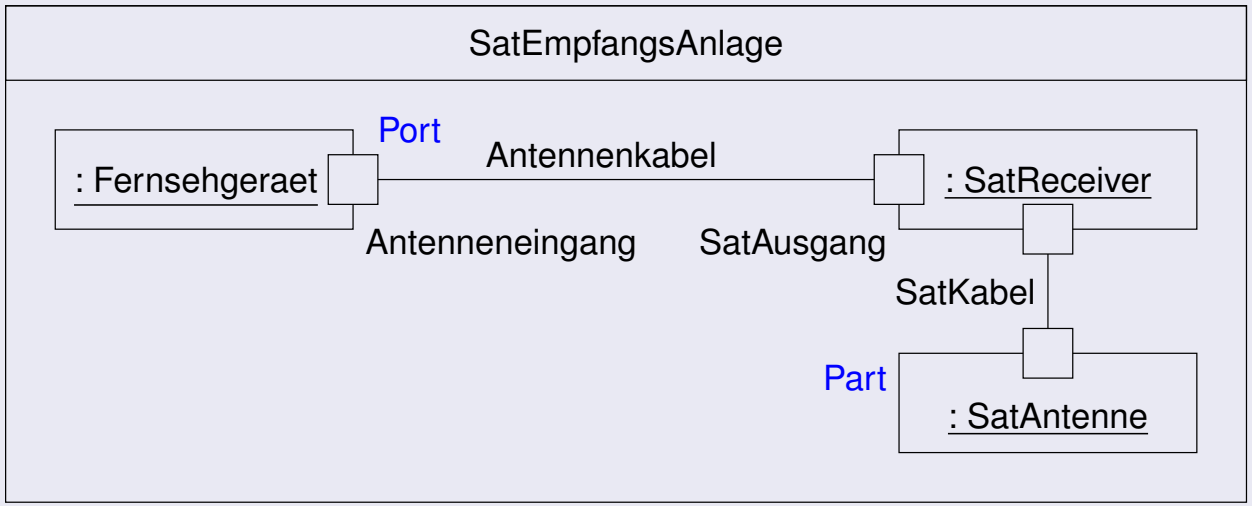
Wir sehen uns nun noch (ganz kurz) die fehlenden vier Typen von Strukturdiagrammen an. ▸ UML-Übersicht

- Kompositionsstrukturdiagramme
- Paketdiagramme
- Verteilungsdiagramme
- Komponentendiagramme

Im weitesten Sinne dienen sie alle dazu, die (übergeordnete) Struktur bzw. Architektur eines Systems darzustellen.

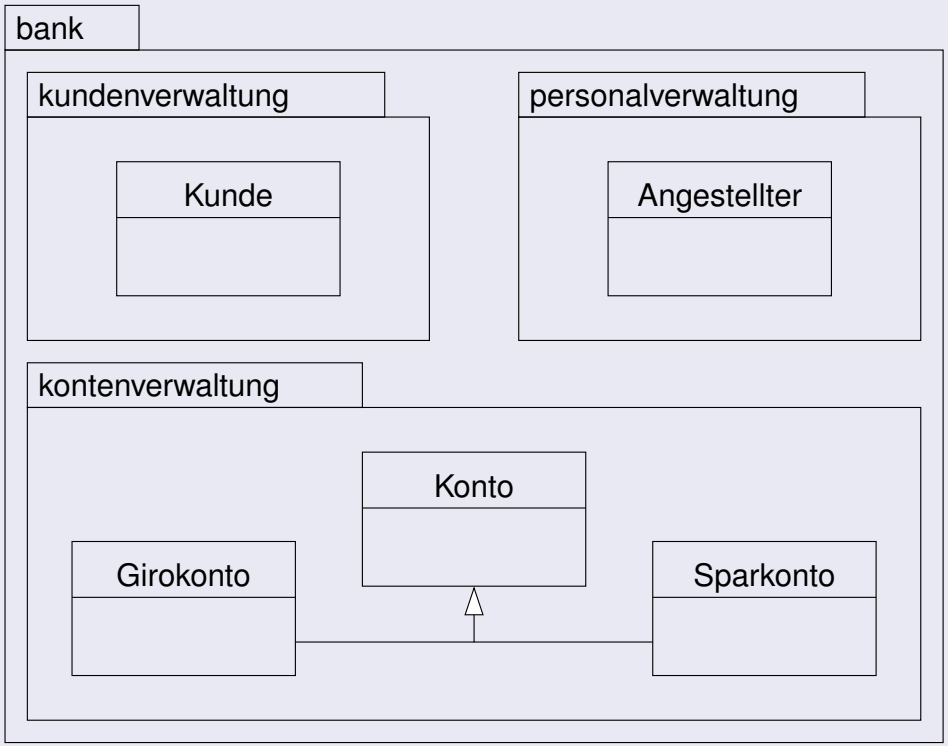
Kompositionsstrukturdiagramme

Kompositionsstrukturdiagramme (engl. composite structure diagrams) beschreiben die interne Struktur von Komponenten (z.B. Klassen) in einer White-Box-Darstellung. Instanzen von durch Komposition verbundenen Klassen werden dabei als sogenannte Parts innerhalb der Klasse dargestellt. Ports, dargestellt als kleine Quadrate, sind Verbindungsstellen zwischen Parts und beinhalten Schnittstellen.



Paketdiagramme

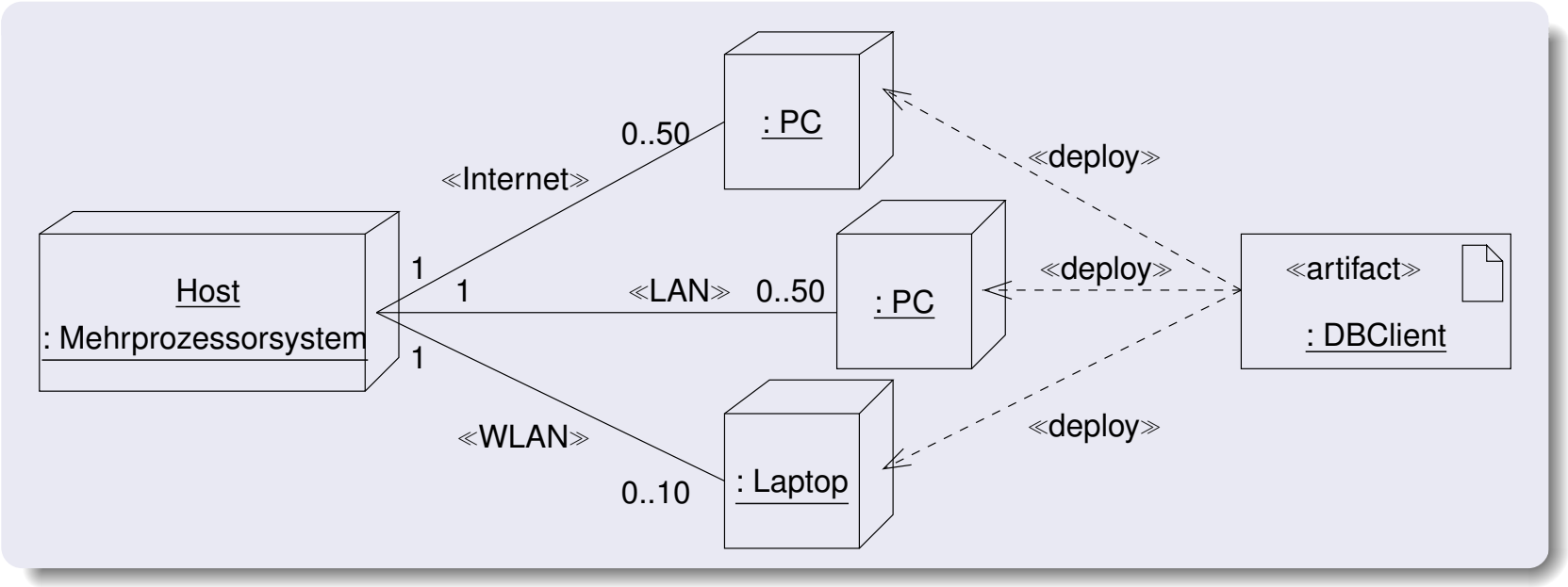
Ein großes Softwaresystem muss in Pakete bzw. Module gegliedert werden. Paketdiagramme (engl. package diagrams) beschreiben die statische Struktur eines großen Systems durch Zusammenfassen von Klassen in Paketen.



Verteilungsdiagramme

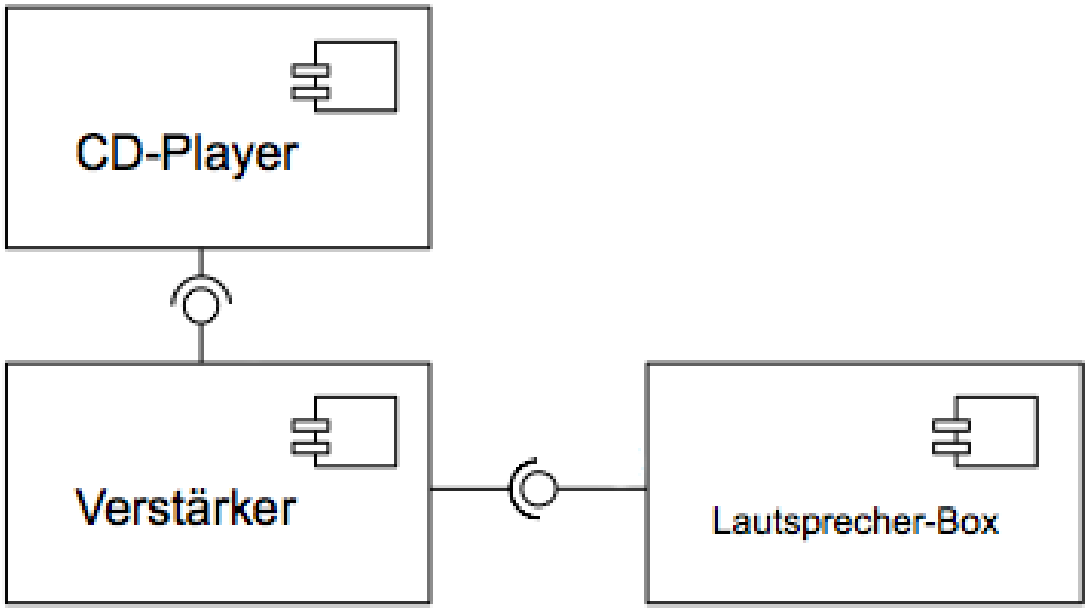
Verteilungsdiagramme (engl. deployment diagrams) beschreiben die Hardware-Komponenten, die in einem System benutzt werden, und wie diese in Beziehung stehen.

Sie können auch konkrete physische Informationseinheiten (Dateien, etc.) enthalten, die als Artefakte bezeichnet werden.



Komponentendiagramme

Ein Komponentendiagramm (engl. component diagram) beschreibt im Gegensatz dazu die Software-Architektur eines Systems. Es beantwortet die Fragen, wie Klassen zur Laufzeit zu größeren Komponenten zusammengefasst werden, und welche Schnittstellen (Services) angeboten und genutzt werden.



Abschließende Bemerkungen

UML als semi-formale Modellierungssprache

UML ist eine semi-formale Modellierungssprache, das heißt, nicht bei jedem Sprachelement gibt es vollständige Einigkeit über die Bedeutung.

Trotz dieser Kritik ist die UML ein großer Fortschritt gegenüber früher genutzten Modellierungssprachen, da sie vereinheitlichte Diagramme bereitstellt, die von jedem Beteiligten am Softwareentwicklungsprozess verstanden werden können.

Abschließende Bemerkungen

Konsistenz von Modellen

Bei der Beschreibung eines großen Systems durch mehrere Modelle ist auch darauf zu achten, dass die Modelle untereinander konsistent sind. (Beispielsweise: Klassennamen, die in einem Sequenzdiagramm verwendet werden, tauchen auch im Klassendiagramm auf.)

Es gibt Ansätze, solche Konsistenz (halb-automatisch) zu erreichen, indem man sogenannte Modelltransformationen durchführt und verschiedene Diagramme ineinander übersetzt.

Abschließende Bemerkungen

Es gibt noch viele Aspekte in der UML, die wir nicht betrachtet haben.

Abschließende Bemerkungen

Es gibt noch viele Aspekte in der UML, die wir nicht betrachtet haben.

Weitergehende Informationen über UML gibt es in zahllosen Büchern (siehe Literaturliste) und auf den Seiten der OMG (Object Management Group):

`http://www.omg.org/technology/documents/formal/uml.htm`

Abschließende Bemerkungen

Es gibt noch viele Aspekte in der UML, die wir nicht betrachtet haben.

Weitergehende Informationen über UML gibt es in zahllosen Büchern (siehe Literaturliste) und auf den Seiten der OMG (Object Management Group):

```
http://www.omg.org/technology/documents/formal
/uml.htm
```

Und natürlich gibt es neben UML und Petrinetzen noch zahlreiche andere Modellierungssprachen!