Formale Sprachen: Graphtransformation

Hans-Jörg Kreowski

Universität Bremen
Studiengang Informatik
kreo@informatik.uni-bremen.de

Sommersemester 2004

Inhaltsverzeichnis

Kapitel 1

Graphen in der Informatik

Wo in der Informatik schwierige Zusammenhänge veranschaulicht oder kompliziert strukturierte Datenobjekte bearbeitet und untersucht werden sollen, sind Graphen als Darstellungsmittel beliebt. *Graphen* sind Strukturen, mit denen sich Beziehungen zwischen Einzelheiten ausdrücken lassen. Es gibt Graphen in vielen Variationen; denn ihre drei Grundbestandteile – Knoten, Kanten und Markierungen – können unterschiedlich miteinander verknüpft werden.

In allen Graphenbegriffen kommen Knoten und Kanten vor. *Knoten* sind voneinander unterscheidbare, eigenständige Objekte; Kanten verbinden Knoten. Während Knoten als Elemente von (Knoten-)Mengen in praktisch allen Graphenbegriffen geeignet identifizierbare Objekte sind, können Kanten sehr verschiedenartig gewählt werden, um Beziehungen zwischen Knoten auszudrücken:

(a) Kann eine Kante Beziehungen zwischen beliebig vielen Knoten herstellen, spricht man von einer *Hyperkante*. Dementsprechend werden Graphen mit Hyperkanten *Hypergraphen* genannt. Üblicher jedoch ist, dass eine Kante zwei Knoten verbindet, was im folgenden behandelt wird.

(b) Sind die beiden Knoten einer Kante gleichberechtigt, handelt es sich um eine *ungerichtete Kante*. Andernfalls ist die Kante *gerichtet*. Technisch lässt sich eine Kante dadurch „richten“, dass ihr ein geordnetes Paar von Knoten zugeordnet wird oder die beiden Knoten verschiedene Bezeichnungen erhalten. Üblich ist die Unterscheidung durch Angabe von *Quelle* und *Ziel* einer Kante. Ein Graph, der nur aus ungerichteten Kanten besteht, wird selbst *ungerichtet* genannt. Analog enthält ein *gerichteter Graph* ausschließlich gerichtete Kanten.

(c) Außerdem können Kanten wie Knoten individuelle Objekte sein; die zugehörigen Knoten müssen dann gesondert zugeordnet werden. Oder Kanten sind aus Knoten zusammengesetzte Größen. Im ersten Fall kann es in einem Graphen gleichartige parallele Kanten zwischen zwei Knoten geben, im zweiten Fall nicht.

Markierungen schließlich dienen dazu, Knoten und Kanten mit zusätzlicher Information zu versehen. Es gibt vier Standardmöglichkeiten, mit Markierungen umzugehen. Sie völlig zu verbieten, liefert *unmarkierte Graphen*. Nur die Markierung von Knoten zu erlauben, ergibt

Knoten–markierte Graphen. Analog werden bei *Kanten–markierten Graphen* nur Kanten markiert. Werden Knoten und Kanten markiert, erhält man *markierte Graphen.*

Diese verwirrende Vielfalt hat den Vorteil, dass das Konzept der Graphen flexibel an konkrete Anwendungssituationen angepasst werden kann. Aber die Variabilität hat auch ihren Preis. Da die verschiedenen Graphenbegriffe unterschiedliche mathematische Eigenschaften aufweisen, lassen sich Ergebnisse über die eine Art von Graphen nicht ohne weiteres auf eine andere Art übertragen:

WARNUNG: Graph ist nicht gleich Graph!

Einige Beispiele sollen die Bandbreite von Graphen und ihre Anwendungsmöglichkeiten illustrieren.

1.1 Wohlstrukturierte Flussdiagramme

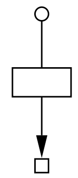
Wohlstrukturierte Flussdiagramme sind graphische Darstellungen der Anweisungen einer einfachen, PASCAL–ähnlichen Programmiersprache. Die Syntax sieht Wertzuweisung, Reihung, Fallunterscheidung (*if–then–else*) und Wiederholung (*while–do*) vor; die zugehörige Backus–Naur–Form lautet:

$$S \;::=\; V := E \mid S; S \mid \textit{if } B \textit{ then } S \textit{ else } S \mid \textit{while } B \textit{ do } S$$

Dabei steht S für „statement“, V für „variable“, E für „expression“ und B für „boolean expression“. Die entsprechenden Flussdiagramme können folgendermaßen als Graphen beschrieben werden:

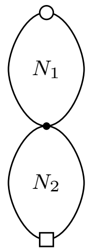
(a) Sie haben immer genau einen Eingang und einen Ausgang, die Knoten sind und mit *begin* bzw. *end* bezeichnet werden.

(b) Eine Wertzuweisung wird als Kante dargestellt:

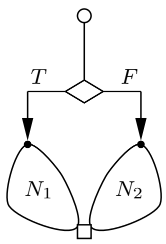


Der Kasten in der Mitte der Kante kann als Markierung aufgefasst werden. Er dient lediglich der Verstärkung des optischen Eindrucks, damit die beschriebenen Graphen auch wie Flussdiagramme aussehen und diese nicht nur formal darstellen. Man beachte, dass nur die Tatsache repräsentiert ist, dass eine Wertzuweisung stattfindet, nicht jedoch, welche Variable welchen Wert erhält. (Das ließe sich durch eine entsprechende Markierung der Kante erreichen.)

(c) Ein Graph N ist ein Flussdiagramm, das eine Reihung repräsentiert, wenn er sich in zwei Graphen N_1 und N_2 aufteilen lässt, die beide Flussdiagramme darstellen, derart dass der Eingang von N_1 der Eingang von N, der Ausgang von N_2 der Ausgang von N und der Ausgang von N_1 und der Eingang von N_2 der einzige gemeinsame Knoten von N_1 und N_2 in N ist:

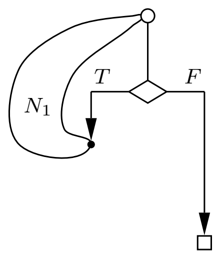


(d) Ein Graph N ist ebenfalls ein Flussdiagramm, das der Fallunterscheidung entspricht, wenn er sich folgendermaßen aufteilen lässt:



Dabei müssen N_1 und N_2 Flussdiagramme sein, deren Ausgang auch Ausgang von N ist und die sonst keine gemeinsamen Knoten haben. Außerdem muss 1 der Eingang von N_1 sein und 2 der von N_2. Neben seinem Eingang muss N noch einen weiteren Knoten besitzen, der mit der Rhombusform markiert ist. Vom Eingang führt eine Kante in diesen Knoten. Von diesem Knoten aus geht je eine Kante nach 1 und 2. Eine davon ist mit T (für „true") markiert, die andere mit F (für „false").

(e) Schließlich ist ein Graph N ein Flussdiagramm, das für die Wiederholung steht, wenn er folgende Form hat:



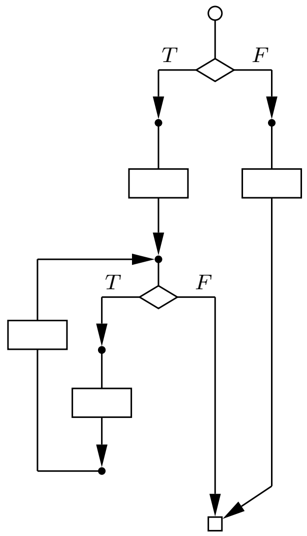
Dabei muss N_1 ein Flussdiagramm sein, dessen Eingang der Knoten 0 und dessen Ausgang der Eingang von N ist. Außerdem besitzt N neben seinem Ausgang noch einen Knoten mit drei anhängenden Kanten wie bei den Fallunterscheidung, wobei hier allerdings die mit F markierte Kante zum Ausgang geht und die mit T markierte zum Knoten 0.

Beachte, dass in (d) und (e) von den konkreten Booleschen Ausdrücken abstrahiert wird.

Ein Programm, das die n-te Potenz von x folgendermaßen bestimmt:

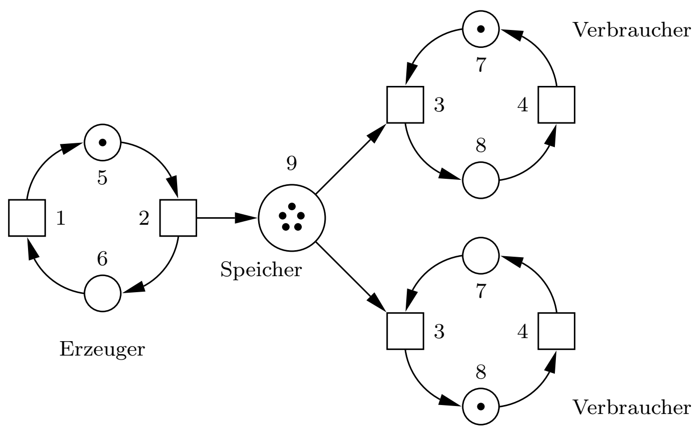
```
if    n > 0
then exp := 1;
     while n > 0 do n := n - 1;
                    exp := exp * x
else exp := 0
```

besitzt als Flussdiagramm den Graphen:



1.2 Erzeuger-Verbraucher-System

Situationen, die in technischen oder informationsverarbeitenden Systemen auftreten, lassen sich sehr häufig angemessen durch Petri-Netze darstellen, die spezielle Graphen sind. Das soll anhand eines vereinfachten Warenlagers demonstriert werden, das von einem Erzeuger bestückt wird und aus dem zwei Verbraucher beliefert werden. Der folgende Graph hält einen „Schnappschuss“ aus diesem System fest:



Dabei gehören die Ziffern nicht zum eigentlichen Graphen, sondern dienen als Referenzen, um über den Sinn der einzelnen Knoten besser sprechen zu können. Alle eckig eingefassten Knoten sind gleich markiert, dazu kann das graphische Objekt einfach als Markierung aufgefasst werden. Diese Knoten repräsentieren jeweils mögliche Aktivitäten des Systems; die folgende Tabelle weist aus, wofür jeder Knoten, abhängig von seiner Ziffer, steht:

1. Vorbereitung (eines Produktionsvorgangs)
2. Herstellung (einer Wareneinheit)
3. Erwerb (einer Wareneinheit)
4. Erwerbsvorbereitung

Die als Kreis gezeichneten Knoten stellen Systemeigenschaften oder -zustände dar. Ihre Markierung ist immer eine natürliche Zahl, die Zahl der im Kreis befindlichen Punkte. Ihr Wert sagt etwas über die konkrete Beschaffenheit der jeweiligen Eigenschaften. Im einzelnen beschreiben die Knoten, die mit 5 bis 9 nummeriert sind, folgende Aspekte des Systems:

5. produktionsbereit
6. nicht produktionsbereit
7. empfangsbereit
8. nicht empfangsbereit
9. Warenlager

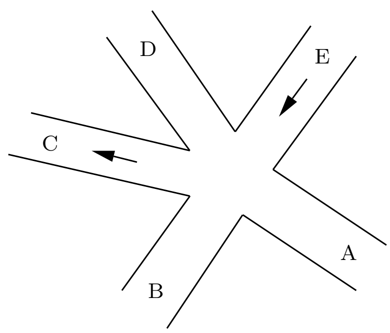
Für die Knoten 5 bis 8 bedeutet die Markierung 0, dass die genannte Eigenschaft gerade nicht vorliegt, während 1 die Erfülltheit beschreibt. Alle anderen Zahlen machen keinen Sinn. Die Markierung des Knotens 9 gibt die Zahl der momentan im Warenlager gespeicherten Wareneinheiten an.

Es ist klar, dass analog zum obigen Graphen durch Veränderung der natürlichzahligen Markierungen andere Systemsituationen repräsentiert werden.

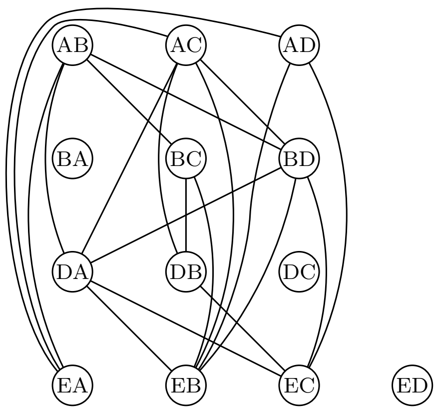
1.3 Verkehrsfluss auf Kreuzungen

Graphen lassen sich auch vorteilhaft einsetzen, wenn bestimmt werden soll, wie viele Ampelphasen erforderlich sind, um komplizierte Kreuzungen verkehrsgerecht zu steuern.

Betrachte etwa folgende Kreuzungssituation:



Die Zufahrtsstraßen sind mit A bis E bezeichnet, und die Fahrtrichtung von Einbahnstraßen ist durch einen zusätzlichen Pfeil angegeben. Paare von Buchstaben beschreiben dann Verkehrsströme, die die Kreuzung passieren können. Sie werden als Knoten eines Graphen genommen. Einige Verkehrsströme können nicht gleichzeitig über die Kreuzung fahren, weil sie sich kreuzen. Wenn das für zwei Verkehrsströme gilt, wird eine Kante (beliebiger Richtung) zwischen ihren Knoten gezogen. Für die obige Kreuzung entsteht so der Graph:



Solche Graphen enthalten wichtige Informationen für die Planung der Ampelphasen an den zugrundeliegenden Kreuzungen, weil grünes Licht für einen Verkehrsstrom als Konsequenz hat, dass alle Verkehrsströme, die mit dem einen durch eine Kante verbunden sind, vor roten Ampeln halten müssen.

Kapitel 2

Ableitung von Graphen durch Anwendung von Regeln

In diesem Kapitel werden die von Chomsky-Grammatiken her bekannten Konzepte der Regel, der direkten Ableitung und der Ableitung, die für Wörter definiert sind, auf Graphen übertragen. So wird es möglich, Graphen aus Graphen durch Regelanwendung abzuleiten.

2.1 Graphmodell

Die Möglichkeiten, Graphen zu definieren, sind vielfältig. Eine häufig verwendete Variante bilden gerichtete, kantenmarkierte Graphen (mit Schleifen und Mehrfachkanten).

2.1.1 Markierungsalphabet

Um nicht bei jedem einzelnen Graphen die Markierungen vereinbaren zu müssen, wird ein globales *Markierungsalphabet* C vorausgesetzt, das allerdings je nach Anwendung und Geschmack frei gewählt werden kann. Beispiele sind die Menge $\mathbb{A}^*$ aller Zeichenketten aus einem Alphabet $\mathbb{A}$, die Mengen $\mathbb{N}$ und $\mathbb{Z}$ der natürlichen und ganzen Zahlen oder die Menge der Wahrheitswerte $BOOL = \{true, false\}$, Aufzählungen wie $COLOR = \{red, green, blue, yellow\}$ sowie Vereinigungen und kartesische Produkte dieser Typen von Markierungen.

2.1.2 Gerichtete, kantenmarkierte Graphen

Ein *gerichteter, kantenmarkierter Graph* ist ein System $N = (V, E, s, t, m)$, wobei

- V eine endliche Menge von *Knoten* ist,
- E eine endliche Menge von *Kanten* ist,
- $s : E \to V$ und $t : E \to V$ zwei Abbildungen sind, die jeder Kante $e \in E$ ihre Quelle $s(e)$ und ihr Ziel $t(e)$ zuordnen,
- $m : E \to C$ eine Abbildung ist, die jeder Kante $e \in E$ ihre *(Kanten–)Markierung* $m(e)$ zuordnet.

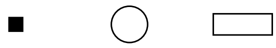


Abbildung 2.1: Verschiedene Knotenformen

In Graphvisualisierungen werden Knoten als geometrische Figuren (Kreis, Rechteck, Ellipse) dargestellt — entweder ausgefüllt oder nur mit ihrer Umrandung (Abb. 2.1).

Eine gerichtete Kante ist an ihrem Quell- und Zielknoten aufgehängt bzw. mit diesem inzident. Grafisch wird dies durch Pfeile dargestellt. In Abbildung 2.2 ist eine gerichtete Kante $e \in E$ zu sehen, die als Quelle $s(e) = v_1 \in V$ und als Ziel $t(e) = v_2 \in V$ hat.

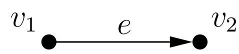


Abbildung 2.2: Grafische Darstellung einer gerichteten Kante

Eine Kante $e \in E$ mit $s(e) = t(e)$, die also an einem einzigen Knoten aufgehängt ist, wird als *Schleife* bezeichnet. Sie kann als gewöhnliche gerichtete Kante mit Pfeilspitze dargestellt oder alternativ als Fähnchen gezeichnet werden (siehe Abb. 2.3).

Abbildung 2.3: Grafische Darstellung von Schleifen

In der grafischen Darstellung einer Kante, bei der Quelle und Ziel verschieden sind, wird die Markierung in der Regel neben dem Pfeil platziert. Bei einer Schleife gibt es weitere Möglichkeiten. Bei der Darstellung als Fähnchen wird die Markierung in die Fahnenfläche geschrieben. Außerdem können Schleifenmarkierungen auch wie Knotenmarkierungen dargestellt und verwendet werden. In diesen Fällen werden die Schleifen nicht gezeichnet, sondern nur ihre Markierungen neben den zugehörigen Knoten oder in diesen geschrieben. Schließlich darf in der grafischen Darstellung einer Kante, die mit dem leeren Wort (λ) markiert ist, die Markierung weggelassen werden, was einer unmarkierten Kante entspricht. Falls es sich um eine Schleife handelt, darf sie allerdings nicht auch noch weggelassen werden, da dieser Fall sonst so aussähe, als sei keine Schleife vorhanden. Die möglichen Darstellungen für Schleifenmarkierungen sind in Abbildung 2.4 zu sehen.

Abbildung 2.5 zeigt einen einfachen gerichteten, kantenmarkierten Graphen, mit vier Knoten und vier Kanten, welche alle mit „a“ markiert sind. In den folgenden Abschnitten wird ein fortlaufendes Beispiel in einer Umgebung mit gerichteten Graphen zur Illustration verwendet. Der abgebildete Graph taucht dort immer wieder als Arbeitsgraph M_0 auf.

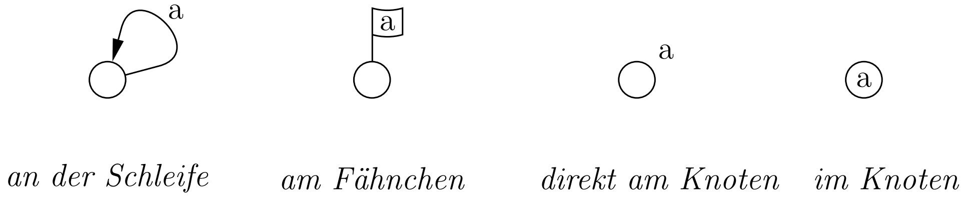


Abbildung 2.4: Schleifenmarkierungen

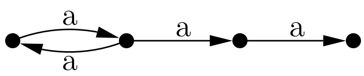


Abbildung 2.5: Gerichteter, kantenmarkierter Graph

2.1.3 Bemerkungen

Die Bezeichnungen für die Graphkomponenten haben mnemonische Zuordnungen aus dem Englischen: V von „**v**ertex", E von „**e**dge", s von „**s**ource", t von „**t**arget", m von „**m**arking" und C von „**c**olor", denn Färben wird synonym für Markieren gebraucht.

Die Endlichkeit der Mengen V und E müsste nicht unbedingt gefordert werden, in der Implementierung der Graphen sind allerdings nur endliche Mengen zu verarbeiten. Das Markierungsalphabet kann unendlich sein.

In einer Umgebung mit mehreren Graphen werden die Komponenten des jeweiligen Graphen zusätzlich mit dessen Namen als Index beschriftet. Ein gerichteter, markierter Graph $N = (V, E, s, t, m)$ ist dann durch $G = (V_N, E_N, s_N, t_N, m_N)$ bestimmt. Dies geschieht, damit die Komponenten der Graphen unterscheidbar sind, ohne dass sie einzeln eingeführt werden müssen.

Im folgenden werden die Attribute „kantenmarkiert" und „gerichtet" oft weggelassen.

2.2 Graphmorphismen und Untergraphen

Es werden Graphmorphismen und Untergraphen eingeführt, um Graphen in anderen Graphen zu lokalisieren.

2.2.1 Morphismus für gerichtete Graphen

Seien L und M gerichtete Graphen. Dann besteht ein *Graphmorphismus* $g = (g_V, g_E) : L \to M$ von L nach M aus zwei Abbildungen $g_V : V_L \to V_M$ und $g_E : E_L \to E_M$, derart dass für alle $e \in E_L$ und $v \in V_L$ gilt:

① $g_V(s_L(e)) = s_M(g_E(e))$,

② $g_V(t_L(e)) = t_M(g_E(e))$,

③ $m_L(e) = m_M(g_E(e))$.

Ein Graphmorphismus g bildet Knoten und Kanten von L auf korrespondierende Objekte aus M ab. Dabei darf eine Kante von L nur dann auf eine Kante von M abgebildet werden, wenn auch Quellen und Ziele entsprechend aufeinander abgebildet werden (nach ① und ②). Außerdem dürfen Kanten nur einander zugeordnet werden, wenn sie gleich markiert sind (③). Die Struktur von L findet sich also in M wieder, denn alle Knoten, Kanten, Aufhängungen und Markierungen werden durch den Morphismus erhalten. Insbesondere der Zusammenhang von Objekten in L überträgt sich vollständig auf korrespondierende Objekte in M.

Ein Beispiel für einen Graphmorphismus g_0 von L_0 nach M_0 ist in Abbildung 2.6 zu sehen. Dort ist mit gestrichelten Linien die Abbildung g_{V0} der Knoten von L_0 nach M_0 eingezeichnet. Die Abbildung der Kanten g_{E0} und deren Markierungen ist eindeutig spezifiert, da an jedem Knoten maximal eine ausgehende und maximal eine eingehende Kante aufgehängt ist. Aus diesem Grund muss die Abbildung g_{E0} nicht dargestellt werden, was zur besseren Übersicht beiträgt.

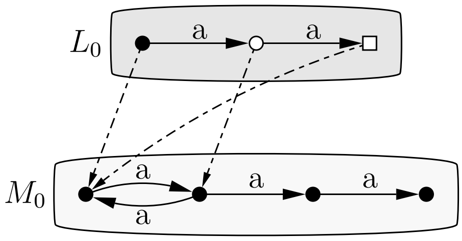


Abbildung 2.6: Morphismus g_0 von L_0 nach M_0

2.2.2 Untergraph

Seien U und M gerichtete Graphen mit $V_U \subseteq V_M$ und $E_U \subseteq E_M$. Dann ist U ein *Untergraph* von M, wenn für alle $e \in E_U$ gilt:

④ $s_U(e) = s_M(e)$,

⑤ $t_U(e) = t_M(e)$,

⑥ $m_U(e) = m_M(e)$.

Abbildung 2.7 zeigt einen Untergraphen U_0 von M_0. Alle Objekte, die vollständig von der Umrandung, die U_0 zugeordnet ist, umschlossen sind, gehören zu U_0.

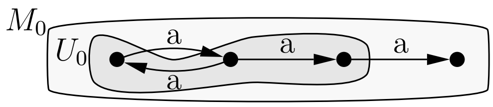


Abbildung 2.7: Untergraph U_0 von M_0

Die Definition von Untergraphen sorgt dafür, dass die Aufhängung von Kanten aus U in M erhalten bleibt (④, ⑤).

Gleiches gilt für Kantenmarkierungen, die für Kanten in U dieselbe sein muss wie in M (⑥).

Ist U ein Untergraph von M, kann dies als $U \subseteq M$ geschrieben werden.

U ist genau dann ein Untergraph von M, wenn die Inklusionen der Knoten- und Kantenmengen einen Morphismus bilden.

2.3 Regelanwendung

Mit Graphen und Morphismen sind die Konzepte verfügbar, die für den grundlegenden Begriff der regelbasierten Graphtransformation gebraucht werden: die Regelanwendung, mit der ein gegebener Graph lokal verändert werden kann. Eine Regel besteht aus einer linken und rechten Seite, die beide Graphen sind und einem Klebegraphen, der gemeinsamer Untergraph von linker und rechter Seite ist (2.3.1). Die linke Seite ohne Klebegraph beschreibt, was bei einer Regelanwendung aus einem Graphen gelöscht werden soll, die rechte Seite ohne Klebegraph, was zugefügt werden soll. Um eine Regel auf einen Graphen anwenden zu können, muss zuerst ein Ansatz (Matching) der linken Seite gefunden werden, was als Morphismus der linken Seite in den zu ändernden Graphen realisiert wird (2.3.2). Die Regel darf allerdings nur angewendet werden, wenn beim Löschen ein Graph entsteht, der sogenannte Zwischengraph (2.3.3). Das ist genau dann der Fall, wenn mit einem Knoten auch alle inzidenten Kanten gelöscht werden. Dafür sorgt die Kontaktbedingung (2.3.2). Das Ergebnis der Regelanwendung, der direkt abgeleitete Graph, entsteht dann aus dem Zwischengraphen durch Zufügen der rechten Seite ohne Klebegraph, indem man quasi Zwischengraph und rechte Seite im Klebegraphen miteinander verklebt (2.3.4). Zusammengenommen ergibt das eine Regelanwendung, die auch direkte Ableitung genannt wird (2.3.5). Wenn direkte Ableitungen nacheinander ausgeführt werden, bezeichnet man dies als Ableitung (2.3.6).

2.3.1 Regel

Eine *Regel* $r = (L \supseteq K \subseteq R)$ besteht aus den drei Graphen L, K und R, wobei der *Klebegraph* K Untergraph von L und R ist. L wird *linke Seite* von r genannt, R *rechte Seite.*

Statt $L \supseteq K \subseteq R$ kann man auch $L \Rightarrow R$ schreiben, wenn sich K durch geeignete Konventionen aus dem Kontext ergibt.

In Abbildung 2.8 ist eine Regel r_0 mit ihren drei Graphkomponenten — linke Seite L_0, Klebegraph K_0 und rechte Seite R_0 — dargestellt.

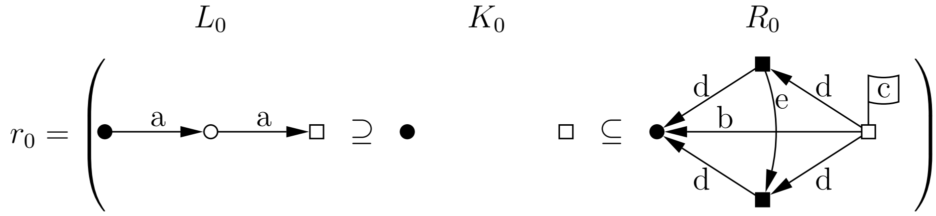


Abbildung 2.8: Regel mit linker Seite, Klebegraph und rechter Seite

2.3.2 Ansatz und Kontaktbedingung

Sei $r = (L \supseteq K \subseteq R)$ eine Regel und M ein Graph.
Dann wird ein Morphismus $g : L \to M$ *Ansatz* von r in M (oder *Matching* von r in M) genannt.

Eine Regelanwendung beginnt mit der Suche nach einem geeigneten Ansatz. Wird kein Ansatz für r in M gefunden, ist eine Graphtransformation mit der Regel r nicht durchführbar.

Ein Ansatz erfüllt die *Kontaktbedingung*, wenn eine Kante e aus $M - g(L)$, d.h. $e \in E_M - g_E(E_L)$, weder ihre Quelle noch ihr Ziel in $g(L) - g(K)$ hat, d.h. $s_M(e), t_M(e) \notin g_V(V_L) - g_V(V_K)$.

Im Beispiel-Arbeitsgraphen M_0 gibt es insgesamt vier verschiedene Ansätze für die linke Seite der Regel r_0, wobei zwei davon die Kontaktbedingung verletzen.
Ein Ansatz, der die Kontaktbedingung nicht einhält, war bereits in Abbildung 2.6 zu sehen. Die Verletzung der Kontaktbedinung besteht darin, dass die mittlere Kante aus M_0 in $E_{M_0} - g_{E0}(E_{L_0})$ liegt, aber in $g_{V0}(V_{L_0}) - g_{V0}(V_{K_0})$ ihre Quelle hat, nämlich den zweiten Knoten von links in M_0.
Abbildung 2.9 (i) zeigt einen weiteren Ansatz g_1 von r_0 in M_0, wobei die Kontaktbedingung verletzt ist durch die Kante, die von rechts nach links verläuft.
Abbildung 2.9 (ii) und (iii) stellen weitere Ansätze g_2 und g_3 dar, die allerdings die Kontaktbedingung erfüllen. Die Kanten, die in $E_{M_0} - g_{E2/3}(E_{L_0})$ liegen und Quelle bzw. Ziel in $g_{V2/3}(V_{L_0})$ haben, berühren dabei nur $g_{V2/3}(V_{K_0})$, was erlaubt ist. Dieser dritte und vierte Ansatz g_2 aus (ii) und g_3 aus (iii) werden in den Beispielen weiterverfolgt.

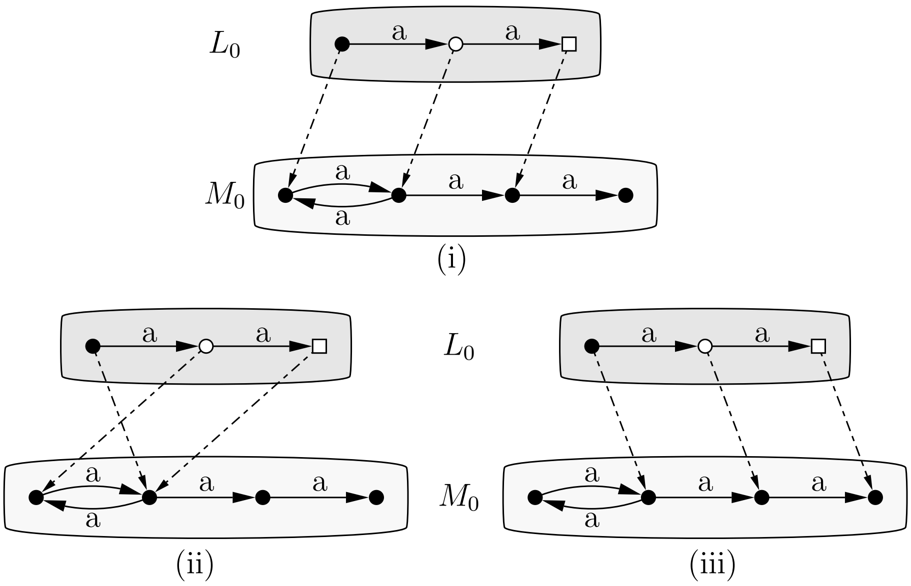


Abbildung 2.9: Ansätze mit verletzter (i) und erfüllter Kontaktbedingung (ii, iii)

2.3.3 Zwischengraph

Sei $g : L \to M$ ein Ansatz von $r = (L \supseteq K \subseteq R)$ in M, der die Kontaktbedingung erfüllt.
Dann ist der *Zwischengraph* Z derjenige Untergraph von M, der entsteht, wenn man alle Knoten

und Kanten aus $g(L) - g(K)$ löscht, d.h. $g_V(V_L) - g_V(V_K)$ aus V_M und $g_E(E_L) - g_E(E_K)$ aus E_M. Dafür wird kurz $Z = M - (g(L) - g(K))$ geschrieben.

Mit dem Zwischengraph Z ergibt sich auch ein Morphismus $d : K \to Z$ mit $d(x) = g(x)$ für alle $x \in K$, der also genau wie g abbildet, allerdings nur für Elemente in K. Wegen der Kontaktbedingung ist die Bildmenge von d auf Z beschränkt.

Der Schnitt des Zwischengraphen Z und $g(L)$ ist gerade $g(K)$. Anschaulich ist also Z mit $g(L)$ über $g(K)$ zu M verklebt. Daher kommt die Bezeichnung *Klebegraph* für K.

Unter Verwendung der Ansätze g_2 aus Abbildung 2.9 (ii) und g_3 aus Abbildung 2.9 (iii) entstehen die Zwischengraphen Z_2 und Z_3, die in Abbildung 2.10 zu sehen sind.

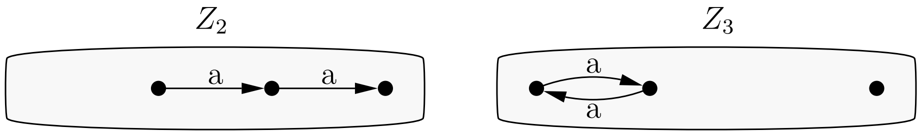


Abbildung 2.10: Zwischengraphen Z_2 und Z_3

2.3.4 Ergebnisgraph

Zum Schluss kann der *Ergebnisgraph* N konstruiert werden als $N = Z + (R - K)$, wobei im einzelnen gilt:

- $V_N = V_Z + (V_R - V_K)$,
- $E_N = E_Z + (E_R - E_K)$,
- $m_N : E_N \to C$ mit
  $m_N(e) = \begin{cases} m_Z(e) & \text{für } e \in E_Z, \\ m_R(e) & \text{für } e \in E_R - E_K \end{cases},$
- $s_N : E_N \to V_N$ mit
  $s_N(e) = \begin{cases} s_Z(e) & \text{für } e \in E_Z, \\ h_V(s_R(e)) & \text{für } e \in E_R - E_K, \end{cases}$
- $t_N : E_N \to V_N$ analog,

mit $h_V : V_R \to V_N$ definiert durch $h_V(v) = \begin{cases} d_V(v) & \text{für } v \in V_K \\ v & \text{sonst.} \end{cases}$

$h : R \to N$ ist schließlich ein weiterer Morphismus, der für alle Elemente $x \in R$ definiert ist durch:

$$h(x) = \begin{cases} x & \text{für } x \in R - K, \\ d(x) & \text{für } x \in K. \end{cases}$$

Praktische Auswirkung dieses Transformationsschrittes ist ein Verkleben von R mit Z ohne Teile von K erneut einzufügen, da diese bereits in Z vorhanden sind. Die Kantenaufhängung einer Kante, die Quelle oder Ziel in Z hat, bleibt in h_V bewahrt. Sonst gilt die Aufhängung in R.

Die Ergebnisgraphen N_2 und N_3 der Beispiel-Regelanwendungen sind in Abbildung 2.11 illustriert.

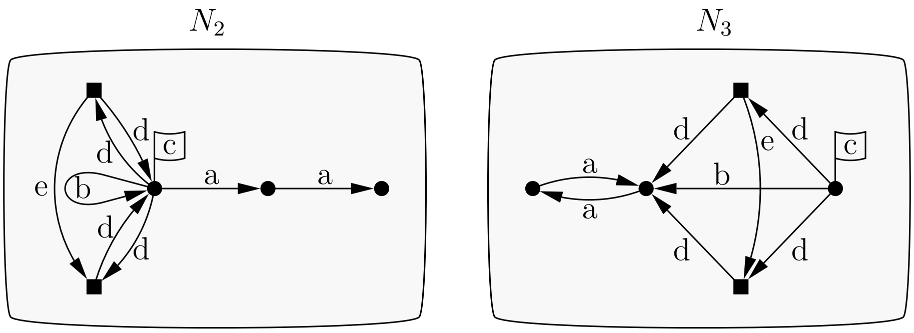


Abbildung 2.11: Ergebnisgraphen N_2 und N_3

2.3.5 Direkte Ableitung

Man sagt, dass der Ergebnisgraph N *direkt* aus dem Graphen M durch Anwendung der Regel $r = (L \supseteq K \subseteq R)$ mit dem Ansatz $g : L \to M$ *abgeleitet wird*, falls die Kontaktbedingung erfüllt ist. Dafür schreibt man kurz $M \underset{r/g}{\Rightarrow} N$ bzw. $M \underset{r}{\Rightarrow} N$, wenn der Ansatz nicht explizit von Interesse ist.

Jede Regel r definiert also durch alle direkten Ableitungen $M \underset{r}{\Rightarrow} N$ eine binäre Relation auf Graphen $\underset{r}{\Rightarrow} \subseteq \mathcal{G} \times \mathcal{G}$, wenn $\mathcal{G}$ die Klasse aller Graphen bezeichnet. Statt $\underset{r}{\Rightarrow}$ kann auch $SEM(r)$ geschrieben werden, wenn betont werden soll, dass die Regelanwendungsrelation als operationelle Semantik einer Regel angesehen werden kann.

Die direkte Ableitung des Beispiel-Graphen M_0 nach N_3 mit der Regel r_0 (Abb. 2.8) und dem Ansatz g_3 aus Abbildung 2.9 (iii) ist in Abbildung 2.12 dargestellt.

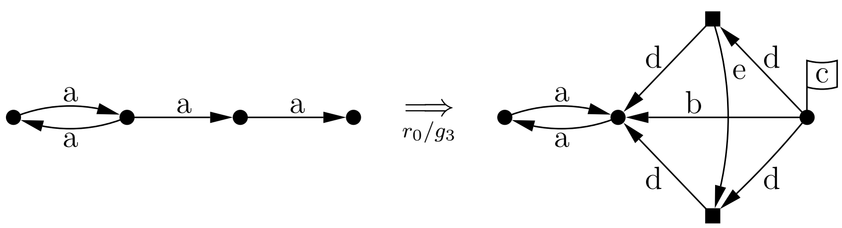


Abbildung 2.12: Direkte Ableitung von N_3 aus M_0

2.3.6 Ableitung

Eine Sequenz direkter Ableitungen wird *Ableitung* genannt.

Seien $M_0 \ldots M_k$ Graphen, $r_1 \ldots r_k \in R$ Regeln aus einer Regelmenge R für $i = 1, \ldots, k$ und g_i ein Ansatz von r_i in M_{i-1}.
Dann sind für eine Ableitung von M_0 nach M_k der Länge k folgende Schreibweisen möglich:

- $M_0 \underset{r_1/g_1}{\Longrightarrow} M_1 \underset{r_2/g_2}{\Longrightarrow} \ldots \underset{r_k/g_k}{\Longrightarrow} M_k$,
- $M_0 \underset{r_1}{\Longrightarrow} M_1 \underset{r_2}{\Longrightarrow} \ldots \underset{r_k}{\Longrightarrow} M_k$,
- $M_0 \underset{R}{\Longrightarrow} M_1 \underset{R}{\Longrightarrow} \ldots \underset{R}{\Longrightarrow} M_k$,
- $M_0 \Longrightarrow M_1 \Longrightarrow \ldots \Longrightarrow M_k$,
- $M_0 \overset{(k)}{\underset{R}{\Longrightarrow}} M_k$,
- $M_0 \overset{(k)}{\Longrightarrow} M_k$,
- $M_0 \overset{*}{\underset{R}{\Longrightarrow}} M_k$,
- $M_0 \overset{*}{\Longrightarrow} M_k$.

Dabei ist auch der Fall $k = 0$ zugelassen. Mit der Ableitung der Länge 0, bei der keine Regel angewendet wird, leitet jeder Graph sich selbst ab.

Literaturverzeichnis

[AH77] K. Appel and W. Haken. Every planer map is 4-colorable. *Illinois J. Math.*, 21:429–567, 1977.

[Aig84] M. Aigner. *Graphentheorie – Eine Entwicklung aus dem 4-Farben-Problem.* Teubner, Stuttgart, 1984.

[Coo71] S.A. Cook. The complexity of theorem-proving procedures. In *Proc. STOC '71*, pages 151–158. ACM, New York, 1971.

[Eve79] S. Even. *Graph Algorithms.* Computer Science Press, Rockville, 1979.

[Gib85] A. Gibbons. *Algorithmic Graph Theory.* Cambridge University Press, Cambridge, New York, Port Chester, Melbourne, Sydney, 1985.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and intractability – A guide to the theory of NP-completeness.* W.A. Freeman and Company, New York, 1979.

[Jun90] D. Jungnickel. *Graphen, Netzwerke und Algorithmen.* BI Wissenschaftsverlag, Mannheim, Wien, Zürich, 1990.

[Meh84] K. Mehlhorn. *Graph Algorithms and NP-Completeness.* Springer, Berlin, Heidelberg, New York, Tokyo, 1984.