Symbolische Interpretation Technischer Zeichnungen

Dissertation

zur Erlangung des akademischen Grades Doktoringenieur (Dr.-Ing.)

vorgelegt an der
Technischen Universität Dresden
Fakultät Informatik

eingereicht von

Dipl.-Inform. Oliver Bringmann
geboren am 24. März 1971 in Berlin

Gutachter: **Prof. Dr.-Ing. habil. Siegfried Fuchs**, Technische Universität Dresden, Fakultät Informatik, Institut für Künstliche Intelligenz,
Prof. Dipl.-Geol. Dr. phil. Manfred Buchroithner, Technische Universität Dresden, Fakultät Forst-, Geo- und Hydrowissenschaften, Institut für Kartographie und
Prof. Dr. rer. nat. habil. Lutz Plümer, Universität Bonn, Landwirtschaftliche Fakultät, Institut für Kartographie und Geoinformation.

Tag der Verteidigung: 8.August 2002

Dresden im Juni 2001

Kurzfassung

Gescannte und vektorisierte technische Zeichnungen werden automatisch unter Nutzung eines Netzes von Modellen in eine hochwertige Datenstruktur migriert.
Die Modelle beschreiben die Inhalte der Zeichnungen hierarchisch und deklarativ. Modelle für einzelne Bestandteile der Zeichnungen können paarweise unabhängig entwickelt werden. Dadurch werden auch sehr komplexe Zeichnungsklassen wie Elektroleitungsnetze oder Gebäudepläne zugänglich.
Die Modelle verwendet der neue, sogenannte Y-Algorithmus: Hypothesen über die Deutung lokaler Zeichnungsinhalte werden hierarchisch generiert. Treten bei der Nutzung konkurrierender Modelle Konflikte auf, werden diese protokolliert. Mittels des Konfliktbegriffes können konsistente *Interpretationen* einer kompletten Zeichnung abstrakt definiert und während der Analyse einer konkreten Zeichnung bestimmt werden.
Ein wahrscheinlichkeitsbasiertes Gütemaß bewertet jede dieser alternativen, globalen Interpretationen. Das Suchen einer bzgl. dieses Maßes optimalen Interpretation ist ein *NP-hartes* Problem. Ein *Branch and Bound*-Algorithmus stellt die adäquate Lösung dar.

Keep it simple!

Folklore

1 Einleitung

Gute Konzepte und Theorien tendieren dazu, einfach zu sein. Das hier beschriebene System ist einfach: Im Kern läßt es sich auf nur drei Relationen, zwei Regeln und zwei Begriffe reduzieren. Umfangreiches Beiwerk wurde zur Verifikation, Optimierung und Motivation hinzugefügt. Hoffentlich überdeckt es nicht die grundlegenden Ideen.

1.1 Motivation – Das große Datenproblem

Die Informationsgesellschaft bringt neben immer neuer Soft- und Hardware enorme Mengen an Daten hervor.
Die Organisation der Daten ist für die Lösung eines bestimmten Problems optimiert und wird nur selten allen heutigen und noch weniger allen morgigen Anforderungen gerecht.
Eine Mehrfachnutzung der – oft teuer erzeugten Daten – ist schwierig, insbesondere wenn die Daten hoch spezialisiert strukturiert sind. Manuelle Arbeit von Spezialisten ist notwendig, um die Daten von einer Repräsentation in eine andere zu überführen.
Die automatische Migration von Daten des speziellen Typs *Technische Zeichnung* ist Gegenstand der Dissertation.

Diese Arbeit entstand während der Realisierung eines Forschungsprojektes zur *modell-gesteuerten, automatisierten Erfassung von Leitungsplänen und Integration in Geo-informationssysteme* an den Instituten für Künstliche Intelligenz und Kartographie der Technischen Universität Dresden. Ziel war die Entwicklung einer Technik, die es ermöglicht, ausgehend von Papier- oder Foliendarstellungen komplexer Leitungspläne, die wesentlichen Zeichnungsinhalte automatisch zu extrahieren. Diese Arbeiten wurden von der Deutschen Forschungsgemeinschaft in zwei Phasen gefördert. Es entstanden in diesem Umfeld Beiträge zur Vektorisierung und zur symbolischen Bildinterpretation. Der vorliegende Text behandelt den zweiten Schwerpunkt. Eine neue, die Modellierung und symbolischen Interpretation **technischer Zeichnungen** umfassende Technologie wird vorgeschlagen. Sie wird mit Y bezeichnet. Durch das Zeichen Y werden zwei Prozesse, die hier eine dominierende Rolle spielen, symbolisiert:

- Von oben nach unten betrachtet, deutet das Y die Aggregation zweier einfacher Objekte zu einem komplexeren Gebilde an.
- Von unten nach oben gelesen, steht das Zeichen für eine mehrdeutige Situation, welche zwei unvereinbare Interpretationen gestattet.

Y führt die symbolische Interpretation technischer Zeichnungen auf eine einzige globale, numerische Optimierung zurück. Das Verfahren ist skalierbar: Verfeinerungen von Modellen und das Hinzufügen von neuen Aspekten ist jederzeit möglich und führt prinzipiell zu einer Verbesserung der Erkennungsleistung. Die Anpassung bereits bestehender Modelle ist nicht notwendig. Neu entstehende Quellen für Widersprüche werden automatisch erkannt. Diese Konflikte werden stets optimal im Sinne einer globalen Interpretation gelöst.
Die verwendeten Modelle sind deklarativ und objektorientiert. Unscharfe
und exakte Modellteile werden explizit unterschieden.

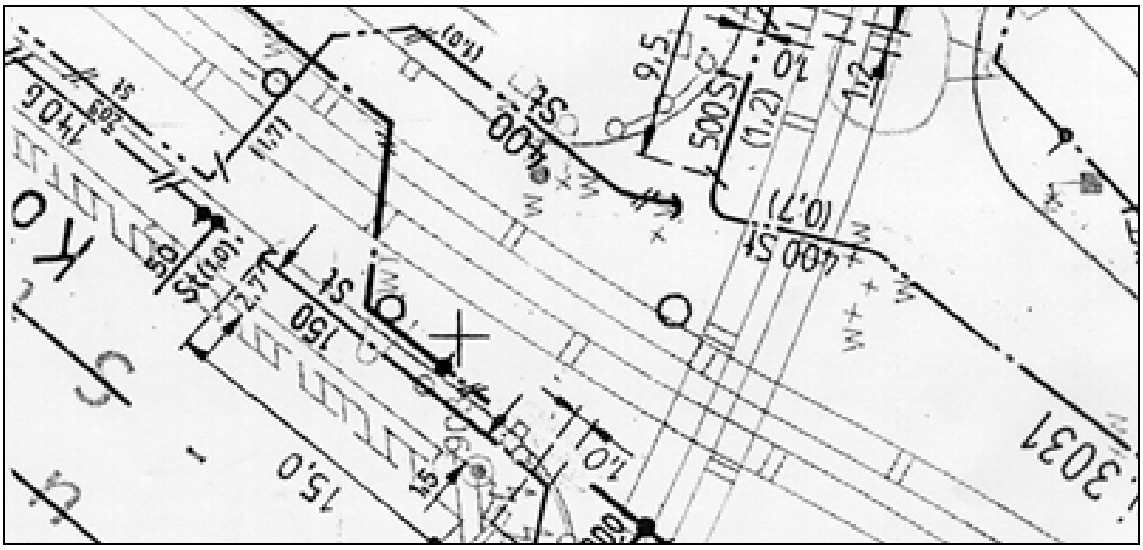


Abb. 1-1 Gasleitungsplan Dresden-GAS; Maßstab 1:500 [Gas 1995]

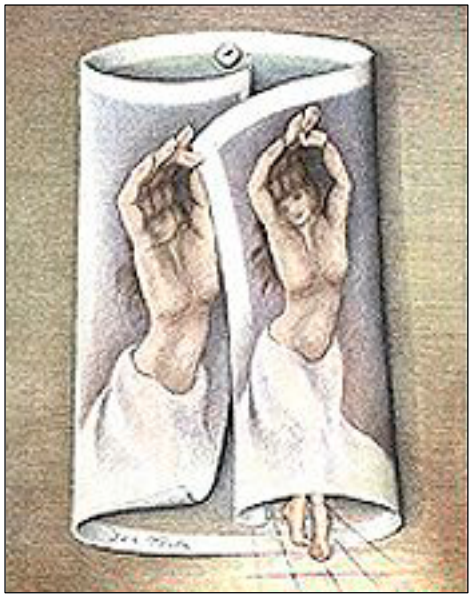
Unter dem Begriff **technische Zeichnung** wird im Kontext dieser Arbeit folgendes verstanden:

Definition 1. *Die graphische Kodierung von Informationen aus einem definierten Fachgebiet unter Nutzung eines festgelegten, oft parametrisierten Zeichenvorrats (Linien, Symbole, Schraffuren) auf einer planaren Fläche zum Zwecke der Interpretierbarkeit durch einen Experten dieses Fachgebiets heißt* ***technische Zeichnung****. Dabei sind die Zeichen stets vollständig dargestellt, d. h. sie überlappen einander nicht.*

Meist gilt die weitere Einschränkung, daß die Zeichnung lokal verständlich sein muß: Einer Fachfrau muß es möglich sein, bei Betrachtung einer räumlich beschränkten Umgebung um ein Objekt (um eine informationstragende Konfiguration von Zeichen), dieses Objekt zu erkennen.

1.2 Menschliche Wahrnehmung

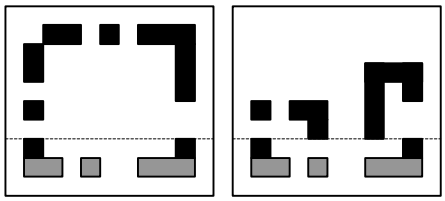
Viele Ideen, die zu dieser Arbeit geführt haben, beruhen auf der subjektiven Beobachtung menschlicher Kognition. Einige etwas spekulative, aber vielleicht auch den Leser inspirierende Bemerkungen zu diesem interessanten Thema seien vorab gestattet:
Die menschliche Wahrnehmung von Bildern ist ein faszinierendes und unerreichtes Vorbild beim Erfinden von Algorithmen für das automatische Bildverstehen. Insbesondere aus Fehlleistungen des Menschen beim Betrachten von Illusionen und sogenannten optischen Täuschungen kann einiges gelernt werden:



Die nebenstehende Grafik des Schweizer Künstlers Sandro Del Prete *Gesture of a Ballerina* zeigt, daß menschliches Sehen kein passiver, sondern im Gegenteil ein aktiver, sogar kreativer Prozeß ist. Beim Einordnen des Wahrgenommenen in unser Weltbild sehen wir oft das, was wir – durch den umgebenen Kontext suggeriert – sehen wollen. So ist dem Autor erst beim Lesen einer Beschreibung dieser Grafik aufgefallen, daß es sich beim linken Bildteil keineswegs um die etwas verwaschene Wiederholung der Ballerina handelt sondern um eine Hand.
Eine weitere Beobachtung ist, daß der Mensch zunächst die leicht und eindeutig erkennbaren Bildteile interpretiert und das dabei gewonnene Wissen für die Analyse der weniger zugänglichen Bildinhalte nutzt.
Es wird offenbar möglichst effizient eine widerspruchsfreie und wahrscheinliche Erklärung für

den erlebten Sinneseindruck gesucht.
Bei der – oft extrem generalisierenden – Gestaltung von Firmenlogos und Piktogrammen werden häufige und wichtige Erfahrungen des Menschen ausgenutzt. Die Designer streben beim Betrachter Assoziationen an, deren Auftreten bei der Zielgruppe hohe a-priori-Wahrscheinlichkeiten besitzen. In unterbestimmten Situation dominieren die a-priori-Wahrscheinlichkeiten bei der unterbewußten Auswahl einer Interpretation. Kontextinformation kann nicht hinzugezogen werden. So wird mit dem nebenstehenden Logo der *Bundesvereinigung Lebenshilfe für Menschen mit geistiger Behinderung* oft ein zu beschützender Mensch in einer bedrängten Situation assoziiert, obwohl objektiv nur ein Kreis und ein Bogen abgebildet sind. Teilweise wird auch ein umarmter Mensch wahrgenommen. Die Interpretation einer Kreisfläche als Kopf ist bei fehlendem Kontext ein naheliegendes *Vorurteil*. Nicht einmal das wichtige technische Konzept *Rad* kann diese sehr zwingende Deutung verdrängen. In der Abbildung gibt es allerdings einen sinnbildenden Indikator, das Wort *Lebenshilfe.*

Die bisher angesprochenen Effekte betreffen Syntax und Semantik von visuellen Wahrnehmungen. Ein weiterer interessanter Aspekt ist die Pragmatik, der Sinn und Zweck der Bildinterpretation durch den Menschen: Gehen furchtsame Menschen nachts durch einen unbekannten Wald, sehen sie eventuell in einer Buschsilhouette ein gefährliches Tier. Hier spielt offenbar die Minimierung eines Risikos (Kosten×Wahrscheinlichkeit) eine gewisse Rolle bei der unterbewußten Entscheidung für eine Bildinterpretation. Die äußerst wahrscheinliche Deutung des Sinneseindruckes als *Vegetation* wird zugunsten der gefährlichen und damit eine schnelle Handlung erfordernden Interpretation *Raubtier* verworfen. Die Optimierung jeglicher Risiken wird in dieser Arbeit nicht betrachtet. Alle übrigen an dieser Stelle skizzierten Phänomene haben dagegen Eingang in das hier vorgeschlagene Konzept gefunden ([Poggio 1986], [Gibson 1950]).
Während der Mensch anscheinend nur selten vor derart verwirrende Situationen gestellt wird, sehen sich die gängigen bildverarbeitenden Algorithmen ständig mit solchen Problemen konfrontiert. Der Grund dafür ist die geometrische und semantische Begrenzung der Wahrnehmung durch die Software: In der Abbildung rechts sieht man, wie ein naiver Linienverfolgungsalgorithmus versagen würde. Bei lokaler Betrachtung des Kontextes unterhalb der dünnen, gerissenen Linie kann nicht entschieden werden, ob ein, zwei oder gar drei gestörte Linien vorliegen. Die inhärent geringe semantische Tiefe der Bildinterpretation ausschließlich durch eine Liniensuche führt auch bei vollständiger Bildinformation zu sehr unsicheren Ergebnissen. Der Algorithmus steht hier vor einem ähnlichen Konflikt wie der Mensch bezüglich der linken Abbildung. Auf dieser nimmt man entweder eine junge oder eine alte Frau wahr (mit rückkoppelnden Konsequenzen für die Interpretation der Pixelgruppe in der Bildmitte als Ohr oder aber als Auge). Wird die oben angesprochene Linienverfolgung um die Kenntnis eines übergeordneten Modells *Rechteck* bereichert, lösen sich die prinzipiellen Probleme auf; zumindest wenn es gelingt, die beiden Konzepte *Linie* und *Rechteck* interagieren zu lassen.



Der Hauptteil dieser Arbeit besteht aus zwei Abschnitten. In Kapitel **3 Modellierung Technischer Zeichnungen** wird eine Technik zur manuellen Modellierung technischer Zeichnungen vorgestellt. Eine solche Modellierungsstrategie ist ihrer Natur nach nicht beweisbar. So ist die oben angegebene Definition einer technischen Zeichnung vielleicht

anschaulich, aber keineswegs formal. Daher kann auch nicht abschließend gezeigt werden, daß sich alle technischen Zeichnungen elegant in ein bestimmtes Modellierungsschema integrieren lassen. Es muß sich vielmehr in der Praxis zeigen, ob eine bestimmte Methode der Abstraktion von der realen Welt sinnvoll ist oder nicht. Allerdings kann ein neuer Ansatz mit bereits bewährten Modellierungskonzepten verglichen werden. Der zweite größere Teil des Textes, das Kapitel **4 Optimale Interpretation Technischer Zeichnungen** beschreibt, wie unter Nutzung der so entwickelten Modelle eine konkrete Vorlage automatisch interpretiert werden kann. Dieser Teil ist einer strengeren, analytischen Betrachtungsweise zugänglich. Er macht von den Definitionen des ersten Teils Gebrauch.
In zwei vorangestellten Abschnitten wird auf die gängigen Ansätze zur symbolischen Bildinterpretation und auf die Vorverarbeitung sowie Vektorisierung von Dokumenten eingegangen.
Den Abschluß dieser Arbeit bildet ein Abschnitt über eine effektive Methode zum automatisierten Lernen der benötigten Wahrscheinlichkeiten. Diesen Ausführungen folgen einige Bemerkungen zur Implementierung und zu bereits erzielten praktischen Ergebnissen. Im letzten Abschnitt werden Anmerkungen über prinzipielle Grenzen des Ansatzes Y sowie ein kurzer Ausblick auf zukünftige Untersuchungen notiert.

Programming is understanding.

Kristen Nygaard

2 Symbolische Bildinterpretation

In der Literatur haben sich viele Begriffe mit ähnlichen Inhalten im Umfeld der symbolischen Interpretation von Bildern etabliert. So wird von symbolischen, strukturellen oder auch syntaktischen Methoden in der Mustererkennung gesprochen. Ziel dieser Verfahren ist immer, einen Weg zu einer symbolischen, formalen Beschreibung des Bildinhaltes zu finden. Die so entstehende Datenstruktur soll alle – entsprechend eines Modells – wesentlichen Bildinformationen enthalten. Diese Informationen sollen explizit gespeichert und daher durch triviale Algorithmen extrahierbar sein. In technischen Zeichnungen sind beispielsweise oft topologische Beziehungen zwischen den Zeichnungsbestandteilen von Belang. Durch eine relationale Datenstruktur können diese repräsentiert und abgefragt werden. Existieren solche Relationen nicht, muß die Topologie – etwa über identische Punkte der Zeichnungsbestandteile – rekonstruiert werden. Ein solches Verfahren ist sicher nicht trivial.
Auf dem Forschungsfeld der symbolischen Bildinterpretation werden, folgt man der Analyse von Karl Tombre [Tombre 1996], verschiedene Formalismen und Methoden untersucht und verwendet:

- **Relationale Strukturen** zwischen Teilinterpretationen repräsentieren die Gesamtinterpretation. Die vorgegebenen relationalen Strukturen auf Modellebene (constraints) werden mit im Bild entdeckten relationalen Strukturen in bestmögliche Übereinstimmung (matching) gebracht. Dabei bedient man sich verschiedener, gut untersuchter Techniken, wie z. B. Verfahren zur Subgraphenisomorphie.
- Die oft adäquat *hierarchisch* darstellbare Struktur von Bildinformationen hat die Nutzung von **formalen Grammatiken** nahegelegt. Die Modellierung der möglichen Bildinhalte erfolgt durch primitive Muster und darüber definierte grammatikalische Regeln. Ob ein Bild zu einer so erzeugten Sprache gehört, kann mittels eines Parsers nachgewiesen werden. Der durch den Parser erzeugte Beweis liefert die konkrete symbolische Bildbeschreibung gleich mit. [Sanifeliu 1993] verweist auf die Begrenzungen von Grammatiken in praktischen Anwendungen: „Die beschreibende Kraft von Zeichenketten- und Baumgrammatiken ist begrenzt. Das impliziert, daß einige komplexe Muster nicht generiert werden können oder in anderen Fällen es nicht einfach ist, die Grammatik zu finden.“
- Konzepte aus der **künstlichen Intelligenz**, wie blackboard- oder wissensbasierte Systeme ([Tanimoto 1990], [Russel Norvig 1995]) können als weniger restriktive Universalverfahren verstanden werden. Beispielsweise kann eine formale Grammatik als Wissensbasis, der dazugehörige Parser als Inferenzmaschine betrachtet werden. Grammatiken können also als Spezialfall wissensbasierter Systeme aufgefaßt werden.
- Die explizite Repräsentation von **Unschärfe** bei der Beschreibung von Modellen wird als wesentliche Ergänzung zu allen genannten Ansätzen angesehen.

Die Bewertung verschiedener Ansätze und Methoden muß unter zwei oft ambivalenten Gesichtspunkten erfolgen.

1. Wie komfortabel ist die Modellierung komplexer Zeichnungsdomänen? Es muß untersucht werden, ob die Methode im Allgemeinen zu einfach verständlichen, generischen und wiederverwendbaren Modellen führt.
2. Gibt es einen (möglichst stets effizienten) Algorithmus, um für alle Bilder eine zu dem Bildsignal optimal passende Beschreibung des Bildinhaltes zu erzeugen, welche kompatibel zu dem vorgegeben Modell ist? Zu optimieren ist somit ein Maß für die Bildsignal-Bildbeschreibung-Abweichung.

2.1 Alternative Ansätze

Es existieren zahlreiche, interessante Ansätze zur symbolischen Bildinterpretation. An dieser Stelle wird ein Verfahren diskutiert, welches durch seine positiven und kritikwürdigen Eigenschaften das Anliegen der vorliegenden Arbeit motivieren kann.
Kai Kulschewski hat einen neuen Zugang zur Erkennung von Gebäuden mittels Bayesnetzen vorgestellt [Kulschewski 1997]. Ausgangspunkt ist das Vorliegen einer primitiven Beschreibung des Bildinhaltes durch Polygonflächen sowie derer Nachbarschaftsbeziehungen. Es wird eine Schrägansicht auf die Szene angenommen.

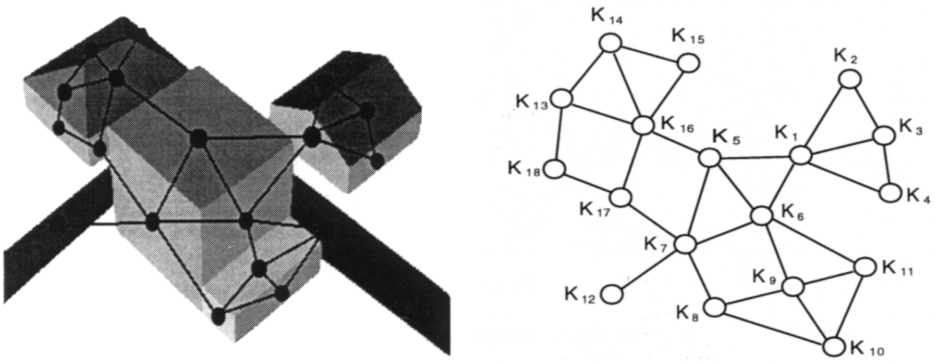
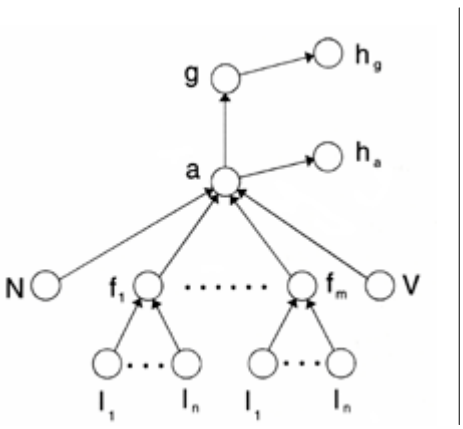


Abb. 2-2 Gebäudeszene und Adjazenzgraph der Polygonflächen

Es existiert ein einheitliches, deklaratives Hausmodell. Ein einzelnes Gebäude egal welchen Typs (Flachdachhaus, Satteldachhaus usw.) und welcher Ansicht ist eine Instanz dieses Modells. Die qualitative Parametrisierung einer solchen Instanz (welcher Haustyp, welche Ansicht) wird durch quantitative Parameter wie den geometrischen Ort des Gebäudes, dessen Ausrichtung, die Hauslänge und vieles mehr ergänzt. Zur Identifikation eines Gebäudes wird das folgende Bayesnetz verwendet:



g	Gebäudetypen (z. B. Satteldachhaus ...)
h_g	Häufigkeit der Gebäudetypen
a	Aspekte (Aquivalenzklasse der Ansichten)
h_a	Häufigkeit der Aspekte
N	Nachbarschft (aus Adjazenzgraph)
V	Verdeckte oder fehlende Flächen
f	Flächenart (z. B. Dreieck, Parallelogramm...)
l	Merkmale
→	Abhängigkeit, Tabelle der bedingten Wahrscheinlichkeiten (z. B. $P(g\|a)$)

Abb. 2-3 Bayesnetz zur Bewertung einer Gebäudehypothese

Die Knoten dieses Graphen sind Zufallsvariablen, die gerichteten Kanten repräsentieren kausale Abhängigkeiten, die durch bedingte Wahrscheinlichkeiten quantifiziert werden (vgl. Abschnitt 4.2.4.1.1 Definition Bayesnetz).
Wird diesem Bayesnetz eine Menge von Merkmalen und der zugehörige Adjazenzgraph der Flächen vorgelegt, kann berechnet werden, wie hoch die Wahrscheinlichkeit dafür ist, daß es sich bei der vorgegeben Konstellation von Flächen um ein Gebäude eines bestimmten Typs handelt. Der entsprechende Eintrag in der Zufallsvariablen g ist dann maximal.
Wie wird nun in der Szene in Abb. 2-3 die richtige Menge an Gebäuden erkannt? Es werden in einem heuristischen Suchverfahren verschiedene Kombinationen von Flächen ausgewählt und als Gebäudehypothesen dem Bayesnetz vorgelegt. Ist die Wahrscheinlichkeit für ein Gebäude in der zugehörigen Zufallsvariable hoch, werden die gewählten Flächen aus dem Adjazenzgraphen

gestrichen und der Prozeß beginnt von neuem. Das Verfahren bricht ab, wenn sich kein weiteres Gebäude aus den verbliebenen Flächen mit hinreichender Wahrscheinlichkeit generieren läßt.
Damit ist das Ergebnis des gesamten Prozesses von der heuristisch gewählten Reihenfolge der Ansteuerung der Flächen abhängig. Ob eine Konstellation von Flächen ein Gebäude bildet, wird ohne die Betrachtung von Kontext und nur durch den Vergleich einer Wahrscheinlichkeit mit einem willkürlichen Schwellwert entschieden. Es wird vor allem nicht festgestellt, ob eine andere Menge von Häusern global gesehen eine bessere Interpretation der Gesamtszene darstellen würde.
Die großen Vorzüge dieses Ansatzes sind:

- Es gibt ein mathematisch fundiertes Unschärfekonzept. Die verschiedenen Vereinfachungen des Modells als Abstraktion der Wirklichkeit werden in klar interpretierbaren bedingten Wahrscheinlichkeiten zusammengefaßt. So werden oft auch Gebäude mit verdeckten Bestandteilen oder leichte Abweichungen vom festgelegten Set der Aspektgraphen erkannt.
- Das Modell ist deklarativ. Die Steuerung der Suche ist klar vom eigentlichen Gebäudemodell getrennt. Sie kann also separat betrachtet und gegebenenfalls verbessert werden.

Wegen der Komplexität der Domänen technischer Zeichnungen kann vermutlich nicht ein einziges komplexes Universalmodell, das den Inhalt einer solchen Zeichnung komplett repräsentiert, entworfen werden. So gibt es in der durch Kulschewski modellierten Domäne kein Modell *Menge von Häusern an Straßen*, sondern es gibt nur einzelne, unabhängig betrachtete Instanzen der Klasse *Haus*. Es ist anzunehmen, daß es auch in realen Anwendungen kein Universalmodell sondern eine Vielzahl von Teilmodellen geben wird. Die Instanzen dieser Teilmodelle werden nicht immer unvereinbar sein. Wäre eine Klasse *Straße* bei Kulschewski modelliert worden, so würde deren simple Struktur (jedes Polygon könnte ein Stück *Straße* sein) dazu führen, daß alle Häuserhypothesen stets mit einer ganzen Menge von Straßenhypothesen konkurrieren. Nun könnte man sich vorstellen, erst alle *Häus*er zu erkennen und anschließend mit den verbleibenden Flächen die Erkennung von *Straße*n zu betreiben. Gäbe es jedoch eine Klasse *leerer_Swimmingpool*, die vermutlich ähnlich komplex wie die Klasse *Haus* wäre, dann ergäbe sich keine vernünftige Reihenfolge für die Analyse der Szene. Die Klasse *leerer_Swimmingpool* müßte also in das Bayesnetz zur Beschreibung von *Häus*ern eingebaut werden. Ein solches Vorgehen führte jedoch zu unübersichtlichen, weil sehr komplexen und unnatürlichen Modellen. Jede Klasse müßte explizit von ähnlichen Klassen abgegrenzt werden. Der Grundsatz „Teile und Herrsche“ der zu der Idee von unabhängigen Teilmodellen führt, würde prinzipiell verletzt werden.
Aus der Analyse dieses Ansatzes können einige Forderungen an ein neues System zur automatischen Interpretation technischer Zeichnungen abgeleitet werden, an denen sich die vorliegende Arbeit messen will:

- Es wird ein Begriff der konsistenten **Interpretation** einer gesamten Zeichnung benötigt, um dem fehlenden Universalmodell und der Mehrdeutigkeit der Analyse Rechnung zu tragen. Bei dem betrachteten Problem wäre eine Menge von *Häus*ern und *Straße*n, ohne daß eine *Straße* identisch mit einer *Haus*fläche wäre und ohne daß zwei *Häus*er identische Flächen beanspruchten, eine solche konsistente Interpretation.
- Diese Interpretationen müssen durch ein integrales Gütemaß bewertet werden und die diesbezüglich optimale Interpretation muß mit vertretbaren Aufwand gefunden werden. Dieses **globale Optimalitätskriterium** sollte sich idealerweise als eine Funktion der lokalen Gütemaße erweisen. Bei Kulschewski ist die Wahrscheinlichkeit eines konkreten *Haus*es ein solches lokales Gütemaß, ein globales Maß fehlt jedoch.

2.2 Prinzipskizze Y

Beim Entwurf des Modellierungs- und Analysesystems Y wird neben den bereits genannten primären Kriterien auf einige weitere Aspekte besonderer Wert gelegt:
Die verschiedenen Zeichnungsinhalte, auf unterschiedlichsten Abstraktionsniveaus sollen einheitlich betrachtet werden. Das heißt beispielsweise, es darf keine endgültige Trennung von

Text und Grafik geben, denn graphische und Textelemente können miteinander verwechselt werden. Ein universelles Unschärfekonzept muß auch schon bei Eingabedaten, den sogenannten **primitiven Objekten**, greifen.
Die Modellierung soll bewährte objektorientierte Konzepte wie strenge Typisierung und Vererbung berücksichtigen. Sie soll deklarativ sein, d. h. *was* zu erkennen ist, soll vom *Wie* des Erkennens strikt getrennt werden. Die verschiedenen Zeichnungsinhalte sollen modular, d. h. weitgehend paarweise unabhängig modelliert werden können. Auch ist eine Trennung von unscharfen und deterministischen Modellteilen anzustreben: Definitive Entscheidungen mittels willkürlicher Schwellwerte sollen entfallen, die Unsicherheit beim Matching explizit gemacht werden.

Als prinzipieller Ablauf der Interpretation einer technischen Zeichnung wird folgendes Schema vorgeschlagen:
Aus der Zeichnung werden primitive Objekte extrahiert. Objekte sind Instanzen einer graphischen Klasse. Eine Definition der Begriffe Objekt und Klasse wird im nächsten Kapitel eingeführt. Primitive Objekte können im Extremfall direkt die Pixel einer gescannten Vorlage sein; im praktisch untersuchten Fall werden jedoch die Ergebnisse einer einfachen Vektorisierung also Linien, Buchstaben usw. verwendet
Unter Benutzung eines umfangreichen Modells, welches die Inhalte dieser Art Zeichnung beschreibt, werden komplexe Objekthypothesen erzeugt: Aus kurzen Linienstücken werden gerissene Linien aggregiert, aus einzelnen Buchstaben Wörter usw.
Viele dieser Hypothesen sind widersprüchlich: Eine Strich-Punkt-Line kann alternativ auch als Kette von Buchstaben ‚i' interpretiert werden.
In einem Optimierungsprozeß wird die global gesehen beste konfliktfreie Menge von Objekthypothesen gebildet und ausgegeben.
In der Abbildung wird der Datenfluß anhand einer Art von Zeichnungen skizziert, die *Quadrate* und rechte *Winkel* enthalten:

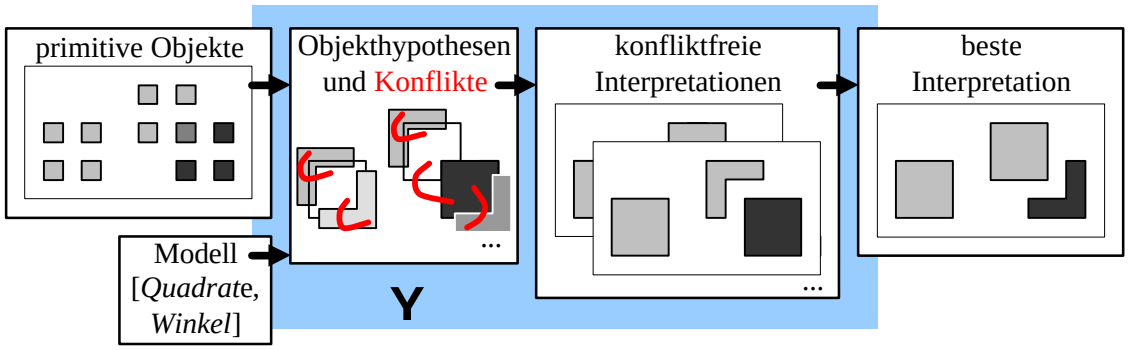


Abb. 2-4 Symbolische Interpretation mit Y, der Datenfluß

2.3 Vorverarbeitung und Vektorisierung

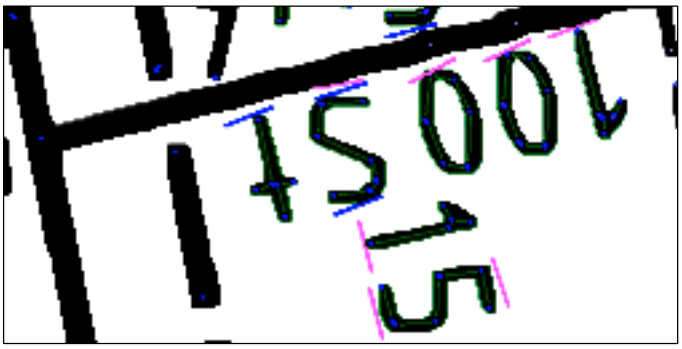
Aufgabe der Vorverarbeitung ist das Bereitstellen von primitiven Bildobjekten. Das gescannte Bild wird umkodiert. Zweck dieser Konvertierung ist eine Vereinfachung des nachfolgenden Analyseprozeßes durch Y. Ziel ist es, die relevante Information aus dem Bild herauszufiltern. Dabei werden bereits einfache Annahmen über die Bildinhalte getroffen. Hier soll unterstellt werden, daß es sich bei den Vorlagen um Liniengrafiken handelt. Es wir davon ausgegangen, daß

- nur eine endliche Menge von gut unterscheidbaren Farben auftritt,
- es eine ausgezeichnete Hintergrundfarbe gibt,
- das morphologische Skelett der Zeichnung in Zusammenhang mit einem Attribut *Strichbreite* den Inhalt der Zeichnung unverfälscht repräsentiert,
- die (alphanumerischen) Zeichen paarweise und von den übrigen graphischen Elementen durch die Hintergrundfarbe isoliert dargestellt sind.

Diese Annahmen sind nicht für die Verwendbarkeit von Y ausschlaggebend. Sie wurden jedoch bei der Implementierung und den Experimenten verwendet. Auch fast alle Beispiele in diesem Text gehen von den genannten Einschränkungen aus.

Eine ganz allgemeine Anforderung an die Vorverarbeitung ist:
Es dürfen keine bedeutungstragenden Informationen verloren gehen. Ist ein bestimmter Bildteil mehrdeutig, kann eine Pixelmenge z. B. als Null oder als Buchstabe ‚O' gedeutet werden, dann müssen alle Varianten erzeugt und – möglichst entsprechend ihrer Wahrscheinlichkeit gewichtet – an Y weitergegeben werden. Um insbesondere dieser letzten Forderung konsequent gerecht zu werden, wurden eigene Verfahren zur Bestimmung der primitiven Objekte eingesetzt und teilweise neu entwickelt. Zunächst wird das Bild binarisiert. Einzelne Pixel werden entfernt und die Konturen geglättet. Anschließend werden Zusammenhangskomponenten von Pixeln gleicher Farbe gebildet. Diese vollständige und disjunkte Zerlegung der Pixelmenge wird zwei unabhängigen Verfahren vorgelegt. Einerseits wird eine Vektorisierung, das heißt eine Beschreibung des Skelettes durch gerade Liniensegmente vorgenommen. Andererseits werden die Konturen der Zusammenhangskomponenten mittels einer homöomorphen Hausdorffmetrik mit einer Datenbank von Symbolprototypen verglichen. Dieser Vergleich ist invariant bezüglich Lage und Rotation der Symbole. Als Ausgabe der Vorverarbeitung werden Y folgende Informationen bereitgestellt:

1. die Menge der durch die Linienstärke und die Koordinaten parametrisierten geraden Linienstücke (Vektoren),
2. die Menge der Symbole mit Einfügepunkt und Richtung,
3. eine zweistellige Relation r_1 zwischen Vektoren und Symbolen, die aussagt, ob eine Zusammenhangskomponente sowohl in einen Vektor als auch in ein Symbol eingegangen ist,
4. eine weitere Relation r_2 über Paare von erkannten Symbolen, welche die Mehrdeutigkeit der Symbolinterpretation reflektiert.



In der Abbildung wird beispielsweise die mittlere Zusammenhangskomponente als Ziffer ‚5', als Buchstabe ‚S' oder als kopfstehender Buchstabe ‚S' erkannt. Diese drei Symbole werden an Y weitergereicht. Jedes dieser Symbole steht weiterhin in Relation r_2 zu den beiden anderen Symbolen. Außerdem werden Y die Vektoren dieser Zusammenhangskomponente angeboten und mittels r_1 zu jedem der drei Symbole in Beziehung gesetzt.
Die Relationen r_1 und r_2 drücken also zusammengenommen die Unvereinbarkeit von Bildprimitiven aus.

Language shapes the way we think,
and determines
what we can think about.

B. L. Worf

3 Modellierung Technischer Zeichnungen

In diesem Kapitel wird beschrieben, wie auf einfache Art und Weise beliebige Klassen von technischen Zeichnungen (also etwa die Menge der topographischen Karten oder die der Gasleitungspläne) modelliert werden können. Der hier vorgeschlagene Zugang basiert auf objektorientierten Entwurfstechniken [Booch 1994] und der in der Dokumentenanalyse weitverbreiteten hierarchischen Modellierung ([Ihle Fuchs 1996], [Albat Boersch Schirmer 1994]). Das Modell einer Klasse von Zeichnungen wird in Teilmodelle zerlegt. Komplexere Zeichnungsbestandteile werden aus primitiveren aggregiert. Die Objektorientierung ist eine gegenwärtig allgemein anerkannte Entwurfstechnik in der Softwaretechnologie und ihre Konzepte können auch die Modellierung technischer Zeichnungen vereinfachen und um Ausdrucksmöglichkeiten bereichern. Ihre Anwendung schlägt sich in einer freien und dennoch strengen Typisierung sowie in der Verwendung von Vererbungsmechanismen nieder: Alle Daten erhalten einen exakten und beliebig strukturierten Definitionsbereich und bekommen direkt Funktionen zugeordnet. Durch Vererbung werden Gemeinsamkeiten und Unterschiede verschiedener Klassen explizit sichtbar.

Erfahrungsgemäß unterscheiden sich Datenmodelle je nach Verwendungszweck. Beispielsweise ist ein Modell, das für die dauerhafte, speicherplatzsparende Datenhaltung geeignet ist, völlig anders geartet als ein Modell, welches die effektive Verarbeitung der Daten durch Algorithmen gestattet. Ebenso werden an Modelle, die kompatibel zu einem Bildanalysealgorithmus sein sollen, besondere Anforderungen gestellt. Die Kunst besteht darin, ein Modellierungskonzept zu entwickeln, welches zwei Anforderungen gerecht wird: Einerseits soll die Abbildung der Wirklichkeit möglichst natürlich und intuitiv vonstatten gehen, andererseits müssen die Modelle für die automatische, symbolische Bildinterpretation nutzbar sein.
Es wird zwischen statischen und dynamischen Modellen unterschieden. Folgendes Beispiel illustriert das Anliegen dieser Differenzierung.

> **Beispiel Dynamisches und statisches Modell.** In einem Satz Zeichnungen sollen stilisierte Häuser gefunden werden. Das statische Modell beschreibt, **was** ein Haus auszeichnet: Es besteht aus einem Dreieck und einem Rechteck, die sich in einer bestimmten Konstellation zueinander befinden. Dreiecke und Rechtecke bestehen wiederum aus Linien. **Wie** die Häuser in einem Wirrwarr aus Strichen und Objekten anderer Art gefunden werden, beschreibt dagegen das dynamische Modell.

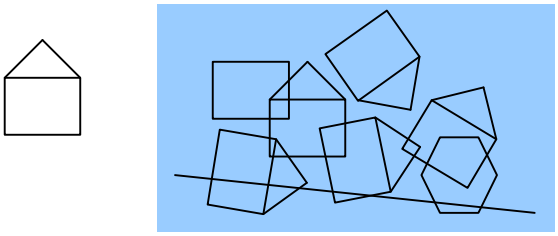


Abb. 3-5 Statisches und dynamisches Modell

Während das statische Modell direkt eine abstrakte Beschreibung der Wirklichkeit darstellt, muß die Aufnahme der Dynamik in das Modellierungskonzept motiviert werden: Die Methode der Zuordnung von Bildteilen zu den statischen Modellteilen ist nicht unabhängig von der betrachteten Domäne. So wird man das Konzept *Gerade* in einem stark verrauschten Bild (Pfeffer und Salz) sicher mittels Houghtransformation suchen. Ist die Domäne dagegen stets durch unverrauschte Vorlagen charakterisiert, kann eine simple Linienverfolgung zielführend sein. Zur Modellierung einer Domäne gehört also neben dem statischen Modell auch die Auswahl geeigneter kombinatorischer Verfahren zur Generierung von Instanzen der statischen Teilmodelle.
Den Aufbau von statischen Modellen für eine bestimmte Sorte Zeichnungen kann man prinzipiell dem Fachmann für diese Zeichnungen zugestehen. Die Entwicklung der dynamischen Modelle erfordert tiefere Kenntnisse auf dem Gebiet der Mustererkennung und der Optimierung.
Deshalb ist eine strikte Trennung der beiden Modellanteile in Theorie und Implementierung sehr bedeutsam. Der hier vorgeschlagene Ansatz unterstützt dieses Konzept vollständig. Aus Gründen einer plausiblen Darstellung wird jedoch während der Beschreibung der statischen Modellierung gelegentlich auf dynamische Aspekte vorgegriffen.

3.1 Statisches Modellierungskonzept

In diesem Abschnitt wird der statische, deklarative Teil der Modelle erläutert. Die Strategie der Objektgenerierung und auch die effizienzsteigernden Einschränkungen des Suchraumes gehören zum dynamischen Modellteil und werden in Abschnitt 3.2 beschrieben.
Für die formale Notation wurde eine klassische funktionale Schreibweise gewählt. Die Implementierung des dargestellten Ansatzes erfolgte objektorientiert, was bei der Wahl einiger Begriffe offensichtlich wird. Es empfiehlt sich vor der weiteren Lektüre dieses Textes, den Abschnitt Druckkonventionen auf Seite 88 zu lesen.

3.1.1 Allgemeine Anforderungen an statische Modelle

In dieser Arbeit wird stets davon ausgegangen, daß genau eine Art von Zeichnungen erkannt und modelliert werden soll. Alle Modelle seien konsistent zu dieser Zeichnungsart. Die gewählte Zeichnungsart (oder synonym die betrachtete Domäne) kann sehr allgemein sein und z. B. alle technischen Zeichnungen umfassen oder aber sehr speziell und nur die Pläne einer einzigen Firma beinhalten. Ein weiterer dazu orthogonaler Aspekt ist die zu modellierende semantische Tiefe. Ist diese gering, wird also bei weitem nicht alles, was ein Experte aus der Zeichnung herauslesen kann, modelliert, dann kann auch die Zuverlässigkeit einer automatischen Interpretation nur begrenzt sein. Ein Beispiel dafür stellt einfache Vektorisierungssoftware dar. Nur die graphischen und aphanumerischen Elemente einer Zeichnung werden durch solche Systeme erkannt. Die tieferliegende Bedeutung des Dargestellten bleibt unverstanden. Durch die fehlende Modellierung komplexerer Zeichnungsinhalte werden oft auch die tatsächlich betrachteten einfachen Elemente falsch erkannt. Ziel dieser Arbeit ist es, ein Werkzeug vorzustellen, welches eine semantisch tiefschürfende Modellierung gestattet und eine solche Modellierung für eine qualitativ hochwertige Interpretation verwendet.
Man wird in diesem Kapitel sehen, daß die entstehenden Modelle anschaulich, deklarativ, unscharf und semantisch lokal definiert sind. Solche Eigenschaften sind wichtig, weil in komplexen Domänen einige hundert Teilmodelle identifiziert werden müssen. Sind die Modelle sehr speziell, gibt es vermutlich nur relativ wenige Vorlagen, die mittels dieser Modelle interpretiert werden können. Wird beispielsweise die exotische Beschriftungsvorschrift einer speziellen Firma X modelliert, können mit diesem Modell vermutlich nur Zeichnungen dieser Firma erkannt werden. Das heißt, man muß bei sehr detaillierter Modellierung die hoch spezialisierten Teilmodelle häufig anpassen bzw. teilweise austauschen. Ein modularer und intuitiver Aufbau der Modelle ist unbedingt erforderlich.

3.1.2 Klassen und Objekte

Entsprechend des allgemein üblichen Zugangs bei einer objektorientierten Modellierung wird zunächst erläutert, wie die zentralen Begriffe **Klasse** und **Objekt** im Kontext dieser Arbeit verstanden werden.
Die Definition der beiden eng zusammenhängenden Begriffe **Klasse** und **Parametertyp** erfordert eine ausführliche Diskussion.

Definition 2. *Eine **Klasse** c ist die Menge aller semantisch gleichwertiger Entitäten einer einzelnen Zeichnung. Die Menge aller Klassen, die – entsprechend der betrachteten Zeichnungsdomäne – auf der Zeichnung vorkommen können, sei C.*

Definition 3. *Einer Klasse c wird ein kartesisches Produkt von n_c Mengen zugeordnet:*

$$Para_c := Para_c^1 \times Para_c^2 \times ... \times Para_c^{n_c} . \quad \textbf{(3-1)}$$

*Die Mengen $Para_c^1 ... Para_c^{n_c}$ heißen **Parametertypen** der Klasse c. Elemente dieser Mengen heißen **Parameter**.*

Beispielsweise ist eine Klasse *Kreis* durch zwei Parametertypen beschreibbar: N×N (Mittelpunktskoordinate; ein Paar natürlicher Zahlen) und N (Radius). Die runden Gebilde auf einer konkreten Zeichnung sind die Elemente der Menge *Kreis*. Enthält die Zeichnungsdomäne prinzipiell *Quadrat*e, auf der konkret betrachteten Zeichnung sind jedoch keine *Quadrat*e vorhanden, dann enthält *C* die leere Menge *Quadrat*.
Eine Domäne ist der eigentliche Gegenstand der Modellierung. Sie ist die Menge aller Zeichnungen, die mittels des an Klassen geknüpften Wissens, beschrieben und später automatisch erkannt werden sollen. Hier wird von einer frei wählbaren, der Definition 1 genügenden, aber dann festgehaltenen Domäne ausgegangen.

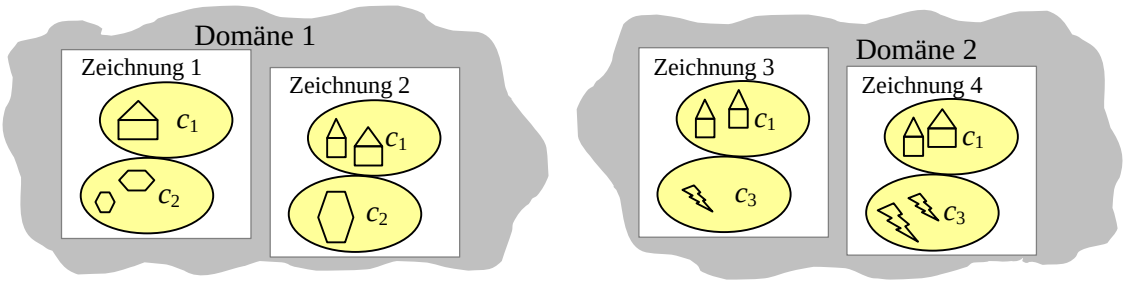


Abb. 3-6 Klassen (Ellipsen) bündeln alle semantisch gleichwertigen Teile einer Zeichnung

Die Formulierung „gleichwertigen Teile“ in der Definition von Klasse meint nur die Gleichwertigkeit bezüglich einer bestimmten Abstraktionsstufe. So werden etwa bestimmte *Linie*n auf einer rein graphischen Betrachtungsebene als gleichwertig betrachtet, weil sie gleichermaßen als *gerissene Linie* auftreten. Auf einem höheren semantischen Niveau werden sie jedoch unterschieden. So können sowohl bestimmte *Leitung*en als auch *Maßhilfslinie*n eine gerissene graphische Repräsentation besitzen.
Die konkreten Ausprägungen der Zeichnungselemente einer Klasse sind also bis auf eine Menge von **Parametern** und den sie umgebenden Kontext völlig gleich. Über die Struktur der Parametertypen wird keine Aussage getroffen. Parametertypen können beispielsweise die geometrische Lage oder bei einer Klasse *Bemaßung* der Wert der Maßzahl sein. Die Parametertypen sind also reelle Zahlen, Punkte in einem euklidischen Raum, Zeichenketten oder auch beliebig komplexe Strukturen wie Histogramme oder Graphen.

Die Zerlegung der Inhalte technischer Zeichnungen in Klassen kann mit folgender sehr allgemeinen Gestaltungsrichtlinie motiviert werden: Die Informationsträger mit ähnlicher Bedeutung haben an allen Stellen der Zeichnungen eine ähnliche Gestalt. Außerdem wird die Technik „Teile und Herrsche“ durch das Klassenkonzept kultiviert. Die Beschreibung komplizierter Zeichnungsbestandteile wird auf die Modellierung einfacherer und oft universellerer Komponenten sowie deren Zusammenwirken zurückgeführt.

In allen Modellen $c \in C$ muß neben den Typen der Parameter auch ein Kontext, in dem Instanzen dieser Modelle auftreten, definiert werden. Der Kontext sind zunächst alle diejenigen Klassen, welche die betrachtete Klasse *c* direkt beeinflussen. Neben der Aufzählung der beeinflussenden Klassen muß für die vollständige Beschreibung des Kontextes angegeben werden, welche Bedingungen diese Klassen bezüglich *c* erfüllen müssen. Beispielsweise müssen *Maßhilfslinien* und *Maßpfeile* rechtwinklig zueinander stehen. Bevor der Kontext formalisiert werden kann, muß geklärt werden, was im weiteren unter dem Begriff **Objekt** verstanden wird.

Definition 4. *Eine Instanz einer Klasse wird **Objekt** o genannt. Die Menge aller auf einer Zeichnung vorhandenen Objekte wird mit O bezeichnet.*

In einem Objekt nehmen die Parameter der zum Objekt *o* gehörigen Klasse c_o konkrete Werte an:

$$p_o := (p_1, p_2 \ldots p_{n_{c_o}}) \in Para_{c_o}. \qquad \textbf{(3-2)}$$

Ein Objekt ist eine physisch auf einer Zeichnung vorhandene Ausprägung einer Klasse. Seine individuellen Parameter, deren Typen für die zugehörige Klasse c_o spezifiziert wurden, sind im n-Tupel p_o gespeichert.
Um den Zusammenhang zwischen Objekten und Klassen formal zu beschreiben und um später zu einer bequemen Notation zu gelangen, werden die Objekte *O* und die Klassen *C* als disjunktes Mengensystem aufgefaßt:

$$\bigcup_{c \in C} c = O$$
$$\forall c, c' \in C : c \neq c' \Rightarrow c \cap c' = \emptyset. \qquad \textbf{(3-3)}$$

Eine Klasse ist daher eine Menge von Objekten und jedem Objekt *o* kann eindeutig eine Klasse c_o zugeordnet werden.
Die Objekte einer Klasse sind die Zeichnungsteile, die semantisch gleichwertig, d. h. mit einem Modell kompatibel sind. Klassen selbst sind Mengen von Objekten. Das Wissen über die Domäne wird den Klassen durch die Definition der Parametertypen und später durch die Definition von Funktionen und Relationen über den Klassen zugeordnet.
Die Definition geeigneter Funktionen über den Klassen ist die Aufgabe der Modellierung. In der Analysephase einer konkreten Zeichnung werden die Instanzen der Klassen, d. h. die Objekte, erzeugt und den entsprechenden Klassen zugeordnet.

3.1.3 Kontext und Objektgenerierung

Der bereits erwähnte Kontext einer Klasse wird nun etwas genauer beschrieben. Es wird zunächst eine zweistellige Relation über der Menge der Klassen eingeführt. Diese Relation definiert den unmittelbaren Kontext einer Klasse *c´*. Alle Klassen *c*, die *c´* direkt beeinflussen, stehen zu *c´* in der Relation *causes*. Man kann die Klassen als Knoten, die Relation *causes* als Kanten eines gerichteten Graphen interpretieren. Von diesem Graphen werde gefordert, daß er frei von gerichteten Zyklen sei.

Definition 5. *Direkte Abhängigkeiten zwischen den Klassen werden durch eine binäre Relation* ***causes*** $\subseteq C \times C$ *auf paarweise Abhängigkeiten beschränkt. Die strukturellen Eigenschaften von causes lauten:*

$$\begin{aligned} &\forall c,c' \in C : c = c' \Rightarrow (c,c') \notin causes && (Irreflexivität) \\ &\forall c,c' \in C : (c,c') \in \overline{causes} \Rightarrow (c',c) \notin \overline{causes} && (Asymmetrie, keine\ gerichteten\ Zyklen) \end{aligned} \quad \textbf{(3-4)}$$

In der Definition von *causes* werde mit $\bar{r}$ die transitive Hülle einer Relation *r* bezeichnet. $\bar{r}$ kann folgendermaßen rekursiv definiert werden:

$$\bar{r} := \left\{ (c,c') \;\middle|\; \begin{array}{ll} (\exists c'' : (c,c'') \in r \wedge (c'',c') \in r) & \vee \\ (\exists c'' : (c,c'') \in \bar{r} \wedge (c'',c') \in \bar{r}) & \end{array} \right\}. \quad \textbf{(3-5)}$$

Die *causes* Relation repräsentiert in erster Näherung die Aggregationsstruktur der Modelle. Die Details und die in der Definition genannten Abhängigkeiten werden später in diesem Kapitel spezifiziert.
Die Angabe der *causes* Relation erfolgt während der Modellierung einer Domäne durch einfache Aufzählung ihrer Elemente. Dadurch wird ein Teil des domänenspezifischen Wissens modelliert.

An dieser Stelle sei ein Vorgriff auf den Abschnitt 3.2.3 Algorithmus Objektgenerierung gestattet, um die in der Definition genannte Forderung nach Zyklenfreiheit zu motivieren. Für die Dynamik der Objekterzeugung ist diese Eigenschaft unbedingt erforderlich. Weil der entstehende Klassengraph frei von Zyklen ist, kann stets eine Ordnung über den Klassen festgelegt werden. Während der Analyse einer konkreten Zeichnung werden die Objekte entsprechend dieser Ordnung generiert. Analog zu *causes* über Klassen wird man eine Relation ***causes*** **über Objekten** definieren. Eine Eigenschaft dieser Relation lautet:

$$\forall (c,c') \in causes : \forall o' \in c' : \exists o \in c : (o,o') \in causes \quad \textbf{(3-6)}$$

Die Objekte einer Klasse werden dann erzeugt, wenn alle Objekte der Vorgängerklassen generiert wurden. In Abb. 3-7 sind (c_1, c_2, c_3, c_4, c_5) und (c_4, c_1, c_3, c_2, c_5) mögliche Reihenfolgen für die Ansteuerung der Klassen.

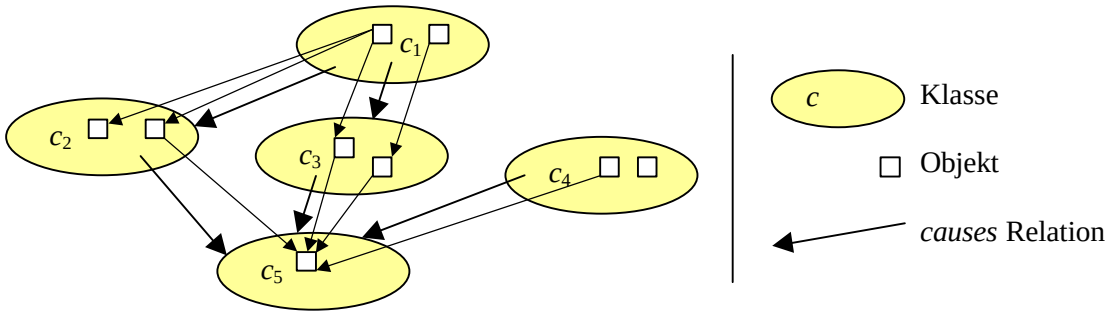


Abb. 3-7 zulässiger Klassen- und Objektgraph

Die Klassen, die in der *causes*-Relation zu einer gegebenen Klasse *c* stehen, werden synonym auch als **Vorgängerklassen** oder **Komponentenklassen** von *c* bezeichnet.

Die Parameter der Objekte einer Klasse *c* werden als Funktion der Parameter der Objekte der Vorgängerklassen von *c* berechnet. Solche Funktionen stellen domänenspezifisches Wissen dar und werden ebenfalls während der Modellierung der Domäne entwickelt.

Definition 6. *Jeder Klasse c wird eine Funktion*

$$g_c : Para_{c_1} \times Para_{c_2} \times ... \times Para_{c_{m_c}} \rightarrow Para_c$$
$$mit\ c_1\ causes\ c \wedge c_2\ causes\ c \wedge ... \wedge c_{m_c}\ causes\ c \qquad \textbf{(3-7)}$$

*zugeordnet. Diese Funktion wird als **Objektgenerator** bezeichnet und besitzt klassenspezifisch m_c Argumente.*
Es gibt neben $c_1 ... c_{m_c}$ keine weitere Klasse c‘ mit c‘ causes c.

Eine solche Funktion beschreibt, wie die Objekte einer Klasse *c* aus den Objekten der Komponentenklassen von *c* hervorgehen. Im Definitionsbereich des Objektgenerators können einige Klassen mehrfach auftreten. In der Abb. 3-7 wurde offenbar eine Funktion

$$g_{c_5} : Para_{c_2} \times Para_{c_3} \times Para_{c_3} \times Para_{c_4} \rightarrow Para_{c_5} \qquad \textbf{(3-8)}$$

modelliert. Zwei Objekte der Klasse c_3 begründen neben jeweils einem Objekt der Klassen c_1 und c_4 die Existenz eines Objektes der Klasse c_5. Die Position einer Klasse in dem n-Tupel der Funktionsargumente kodiert die Rolle der jeweiligen Klasse bezüglich derjenigen Klasse, die den Funktionswert bestimmt. Werden beispielsweise zwei *Ziffer*n – also zwei Objekte einer Klasse – zu einer *Zahl* aggregiert, dann wird die linke *Ziffer* die Rolle des Zehners, die rechte *Ziffer* die des Einers in einem Objekt der Klasse *Zahl* einnehmen.
Daß formal gesehen ein Objekt zweimal in ein anderes Objekt einfließen kann, soll an dieser Stelle nicht stören. Derartige Situationen werden im Abschnitt 3.2.4 zur automatischen Konfliktgenerierung ausgenutzt.

Definition 7. *Klassen, die keine Vorgängerklassen besitzen, werden als **primitive Klassen** $C_{primitive}$ bezeichnet. Sie benötigen keine Objektgeneratoren:*

$$C_{primitive} := \{c \in C \mid \nexists c' \in C : c' causes\ c\}. \qquad \textbf{(3-9)}$$

Definition 8. *Die Menge der Objekte, die Elemente primitiver Klassen sind, heißt $O_{primitive}$:*

$$O_{primitive} := \bigcup_{c \in C_{primitive}} c\ . \qquad \textbf{(3-10)}$$

Die Objekte der primitiven Klassen werden direkt durch Aufzählung angegeben. Ebenso werden die Parameter dieser Objekte als bekannt vorausgesetzt.
Viele Klassen dienen nur der Speicherung und Modellierung von Zwischenergebnissen bei der Analyse einer technischen Zeichnung. Beispielsweise wird eine Klasse *Bemaßungspfeilspitze* nur ein Zwischenprodukt bei der Konstruktion einer Klasse *Bemaßung* sein. Hat letztere die Semantik „Projizierter Abstand zweier Punkte“, dann enthält sie alle Informationen, die von einem entsprechenden GIS-Datensatz erwartet werden. Die Klassen, die in ihrer Gesamtheit die volle Semantik einer Vorlage beschreiben, müssen während der Modellierung identifiziert werden.

Definition 9. *Die Menge der Klassen, welche die Ausgabeschnittstelle einer Domäne repräsentieren, sei C_{out}. Es gilt $C_{out} \subseteq C$.*

Definition 10. *Die Menge der Objekte der Ausgabeschnittstelle heißt O_{out}:*

$$O_{out} := \bigcup_{c \in C_{out}} c \qquad \textbf{(3-11)}$$

Um eine Vorlage automatisch analysieren zu können, müssen die primitiven Objekte und alle Modelle – also die Klassen – existieren.

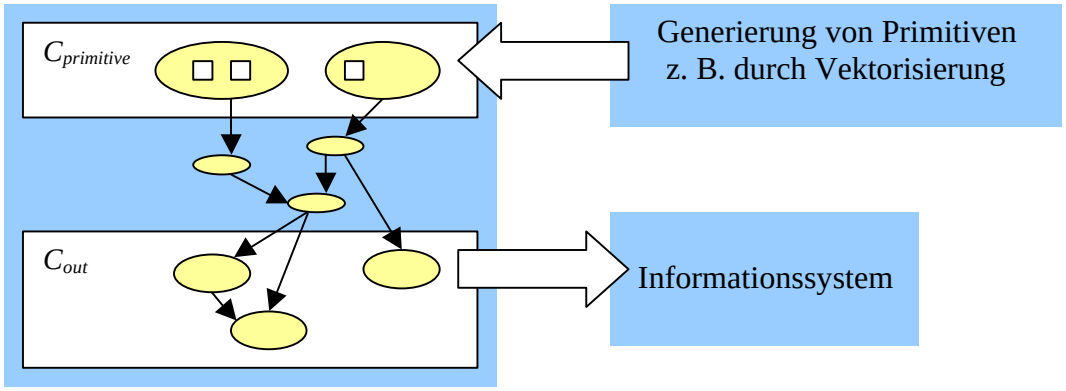


Abb. 3-8 Situation vor der Analyse einer Vorlage

3.1.4 Entwurfsparadigma

Die Beschreibung des Kontextes einer Klasse erfolgt bisher recht unspezifisch mittels der *causes* Relation. Die meisten (auch alle bisher in Beispielen betrachteten) Elemente der *causes* Relation können einen wesentlich stärkeren Zusammenhang ausdrücken, als es diese Relation vermag. So ist eine *Ziffer* nicht nur durch eine stochastische Abhängigkeit mit einer *Zahl* verbunden, sondern eine *Ziffer* ist ein exklusiver Teil einer *Zahl*. Eine *Ziffer* kann nur Teil genau einer *Zahl* sein. Bei der Modellierung eines Kreuzworträtsels kann dagegen ein *Buchstabe* Teil mehrerer *Wörter* sein. In diesem Fall gilt nur die weniger starke Aussage: *Buchstabe causes Wort.*

In vielen Arten von technischen Zeichnungen kann eine Teilmenge der *causes* Relation identifiziert werden, die eine solche Teil-Ganzes Beziehung ausdrückt.

In der Objektorientierung wird diese Relation meist mit „part of“ bezeichnet. Eine solche Relation ist gekennzeichnet durch das exklusive Enthaltensein eines Objektes in einem anderen.

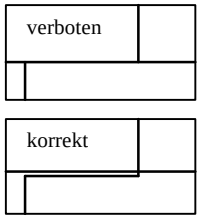
Ein wichtiges Gestaltungsmuster, das bei der Modellierung der Klassen beachtet werden muß, ist das folgende **Entwurfsparadigma**. Es charakterisiert den Unterschied zwischen der Teil-Ganzes-Relation und *causes*.

Um ein automatisches Erkennen von Zeichnungen mit dem hier dargestellten Verfahren vornehmen zu können, muß die Auswahl und der Entwurf der Klassen einer grundlegenden Forderung genügen:

Entwurfsparadigma. *Ein Zeichnungsbestandteil darf unmittelbarer und exklusiver Teil von höchstens einem anderen Zeichnungsbestandteil sein. Alle Zeichnungsbestandteile sind vollständig dargestellt.*

Die Motivation für das Paradigma ist aus folgender Beobachtung abzuleiten: In technischen Zeichnungen jeglicher Art wird auf eine gute Lesbarkeit durch den menschlichen Betrachter Wert gelegt. Auch kleine Ausschnitte von Zeichnungen sollen stets eindeutig interpretierbar sein. Beschriftungen werden so angelegt, daß sie einander und andere Zeichnungsinhalte nicht verdecken. Überlappungen von wichtigen Informationsträgern sind verboten. In der Kartographie wird deshalb beispielsweise das Konzept der Verdrängung verwendet. Mindestabstände zwischen Objekten müssen in Karten eingehalten werden. Um eine gute Lesbarkeit zu gewährleisten, wird vermieden, daß ein und dasselbe graphische Primitiv in verschiedenen Kontexten verwendet wird. So darf eine Straßenbegrenzungslinie niemals gleichzeitig als Symbolik für eine Leitung eingesetzt werden. Die Leitung muß – auch wenn

dies geometrisch nicht ganz korrekt ist – *neben* der Straßenbegrenzungslinie eingetragen werden, auch wenn beide Klassen die selbe graphische Signatur aufweisen. Eine in der Zeichnung detektierte Linie ist entweder eine Leitung oder eine Straßenbegrenzungslinie. Damit ist gewährleistet, daß die Einhaltung des Entwurfsparadigmas tatsächlich durchführbar und natürlich ist.



Hier ein Beispiel aus einer Aggregationsstruktur, bei deren Definition der Teilmodelle das Entwufsparadigma zu beachten ist:

> **Beispiel Entwurfsparadigma.** Eine *Bemaßung* existiert genau dann, wenn ein *Maßpfeil* und eine *Maßzahl* existieren. Außerdem müssen sie Teil der *Bemaßung* sein (durchgezogene Pfeile). Die zu bemaßenden Klassen *Leitung* und *GK-Element* (Grundkartenelement) müssen auch existieren. Sie sind aber nicht exklusiver Teil der *Bemaßung* (gerissene Pfeile).
> Wäre die *Leitung* „Gas“ ein Teil der *Bemaßung*, so wären nach dem Entwurfsparadigma die beiden *Bemaßung*en in dem Leitungsplan unvereinbar. Die *Leitung* im Leitungsplanausschnitt würde gleichzeitig Teil zweier *Bemaßung*seinheiten sein. Da der später dargestellte Algorithmus zur Erkennung des Planes auf die Einhaltung des Paradigmas vertraut, würde er die beiden *Bemaßung*en als unvereinbar markieren. Dies wäre jedoch offensichtlich ein Fehler.
> Da ähnliche Situationen bei Grundkartenelementen möglich sind, dürfen auch diese nicht als Teil einer Bemaßung modelliert werden.

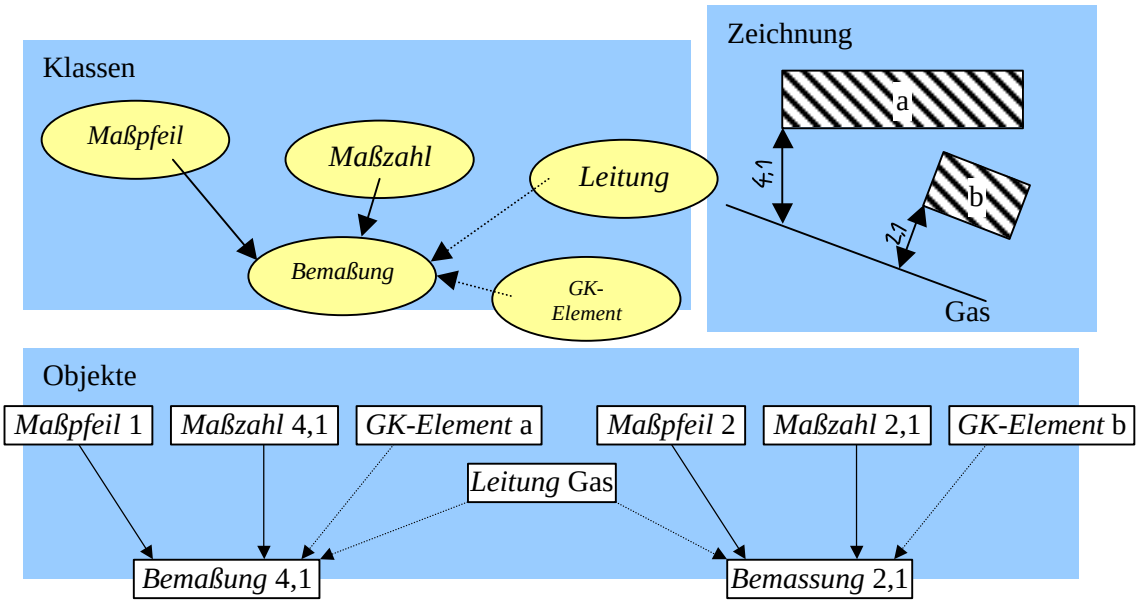


Abb. 3-9 Links eine Aggregationsstruktur, rechts ein Leitungsplan mit zwei Bemaßungen, einer Leitung und zwei rechteckigen Grundkartenelementen, unten die Objektstruktur

Werden die Modelle entsprechend der hier vorgestellten Technik entwickelt, so eignen sie sich für die automatische Analyse mittels des im Kapitel 4 vorgeschlagenen Verfahrens. Die dort spezifizierte Methode wird dabei intensiv vom Entwurfsparadigma Gebrauch machen: Eine **Interpretation** einer Zeichnung ist nur dann gültig, wenn das Paradigma für alle Objekte der Interpretation eingehalten wurde.

Eine Interpretation, die gleichzeitig die Objekte „1005“ und „Stahl“ enthält, ist ungültig, weil die Kontur, die zur Ziffer 5 bzw. zum Buchstaben S aggregiert wurden, identisch ist. Dagegen ist die Interpretation {„100“, „Stahl“} ebenso zulässig wie die Objektmenge {„1005“, „tahl“}, wenn auch letztere weniger wahrscheinlich ist.

Abb. 3-10 Ausschnitt aus einem Leitungsplan mit alternativen Interpretationen

Um das Entwurfparadigma einhalten zu können, muß ein formaler Mechanismus vorgesehen werden, der dem Modellierenden gestattet, im Definitionsbereich eines Objektgenerators g_c diejenigen Klassen zu markieren, die als echter Teil in die Klasse *c* eingehen. Im Beispiel Entwurfsparadigma müssen im Generator

$$g_{Bemaßung} : Para_{Maßpfeil} \times Para_{Maßzahl} \times Para_{Leitung} \times Para_{GK-Element} \rightarrow Para_{Bemaßung} \qquad \textbf{(3-12)}$$

die Argumente, die zu den Klassen *Maßpfeil* und *-zahl* gehören, markiert werden.

Definition 11. *Jeder Klasse c wird ein boolescher Vektor*

$$parts_c := (b_1, b_2 .. b_{m_c}) \ \ mit$$

$$b_i = \begin{cases} 1\,, \ wenn\ c_i\ in\ (\ g_c : ... \times Para_{c_i} \times ... \rightarrow Para_c\ )\ ein\ Teil\ von\ c\ ist, \\ 0\,, \ wenn\ c_i\ in\ (\ g_c : ... \times Para_{c_i} \times ... \rightarrow Para_c\ )\ kein\ Teil\ von\ c\ ist. \end{cases} \qquad \textbf{(3-13)}$$

zugeordnet.

Eine zur Klasse *Bemaßung* gehörige Markierung $parts_{Bemaßung}$ = (1,1,0,0) gewährleistet die Einhaltung des Entwurfsparadigmas im oben genannten Beispiel.
Der Vektor *parts* besagt also, welche Objekte exklusiver Teil eines anderen Objektes sind. Durch *parts* wird eine Teilmenge von *causes* definiert.

Zur Illustration aller bisher dargestellten Modellierungswerkzeuge wird ein einfaches, ausbaufähiges Beispiel vorgestellt:

Beispiel *Leitung.* Eine *Zahl* wird mit einer *Linie* zu einer *Leitung* zusammengefaßt. Die *Zahl* beschriftet die *Linie* und bekommt innerhalb eines Objektes der Klasse *Leitung* eine spezielle Semantik, die des Leitungsdurchmessers. Umgekehrt wird die *Linie* durch die Beschriftung Teil einer *Leitung*. Im Standardfall verläuft die Grundlinie der Zahl parallel zur *Linie* in einem in der Zeichenvorschrift niedergelegten Abstand. In der Abb. 3-11 wurde die mögliche Abweichung von dieser optimalen Konstellation zur Veranschaulichung übertrieben.

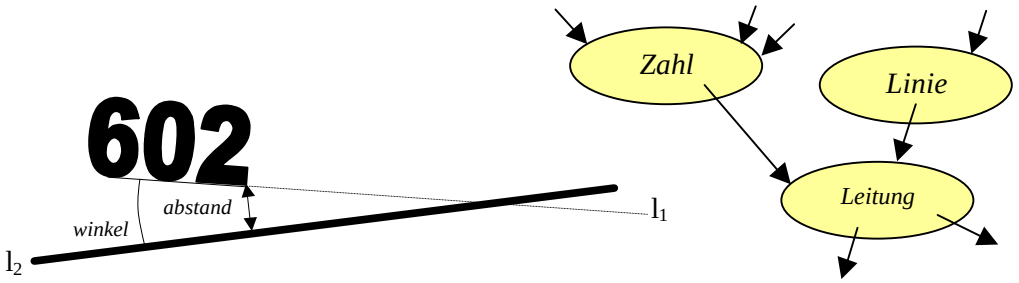


Abb. 3-11 Leitungsplanfragment und die entsprechenden Klassen mit *causes* Relation

Die bisher angesprochenen Teile des Modellierungskonzeptes werden nun auf dieses Beispiel angewandt. Dabei wird vereinfachend angenommen, daß nur die bereits erwähnten Klassen zum Gesamtmodell gehören. Als Parameter der Klassen seien natürliche Zahlen N sowie eine Struktur *line* verfügbar. Diese Struktur enthalte zwei Koordinatenpaare. Ferner seien zwei Funktionen *abstand* und *winkel* über der Struktur *line* definiert. Diese Funktionen liefern ebenfalls eine natürliche Zahl als Rückgabewert.
Alle bisher eingeführten Komponenten des statischen Modells des Beispiels *Leitung* lauten:

$C = \{Zahl, Linie, Leitung ...\} \qquad C_{out} = \{...\}$

$causes = \{(Zahl, Leitung), (Linie, Leitung)\}$

$Para_{Zahl} = ? \times line$

(Wert der Zahl und Koordinatenpaar der Grundlinie)

$Para_{Linie} = line$

(Koordinaten der Linie)

$Para_{Leitung} = ? \times line \times ? \times ?$

(Durchmesser, Koordinaten der Leitung, Abstand und Winkel)

$g_{Leitung} : Para_{Zahl} \times Para_{Linie} \rightarrow Para_{Leitung}$

$$g_{Leitung}\left(\begin{pmatrix} wert \\ l_1 \end{pmatrix}, (l_2)\right) = \begin{pmatrix} wert \\ l_2 \\ abstand(l_1, l_2) \\ winkel(l_1, l_2) \end{pmatrix} \quad mit\ wert \in ?\ \ und\ l_1, l_2 \in line \qquad \textbf{(3-14)}$$

$parts_{Leitung} = (1,1)$ *(alle Komponenten sind exklusiver Teil der Leitung)*

3.1.5 Unschärfekonzept

Bisher ließ die Modellierungsmethode zu, immer dann ein Objekt zu generieren, wenn die nötigen Komponenten vorhanden waren. Auch wenn diese Komponentobjekte offensichtlich und lokal beschreibbar nichts miteinander zu tun hatten, konnte der Objektgenerator ein Objekt erzeugen. Die Generierbarkeit eines Objektes hängt jedoch neben der Existenz seiner Komponenten vor allem von deren Konstellation zueinander ab. Nun wird in den Parametern $Para_c$ einer Klasse c bzw. in deren Objektgenerator g_c Wissen über die Konstellation der Teilmodelle von c gespeichert bzw. erzeugt. Jedoch nicht alle Parameter enthalten signifikante Informationen für die Existenz eines Objektes. Daher sollte aus dem Vektor der Parameter genau der Teil selektiert werden, welcher die Konstellation der beeinflussenden Objekte beschreibt. Dies geschieht mittels einer Selektionsfunktion.

Definition 12. *Jeder Klasse c wird eine Funktion zugeordnet, die aus dem Parametervektor von c die signifikanten Parameter auswählt:*

$$const_c : \bigtimes_{i=1}^{n_c} Para_c^i \rightarrow \bigtimes_{i \in a} Para_c^i$$

$$mit\ \ a \subseteq \{1 \ldots n_c\}.\quad a\ enthält\ die\ Indizes\ signifikanter\ Parameter, \qquad \textbf{(3-15)}$$

$$const_{c_o}(p_o) = const_{c_o}(p_1 \ldots p_{n_{c_o}}) := (p_{\min(a)} \ldots p_{\max(a)}).$$

*Diese Funktion wird als **Konstellationsfunktion** bezeichnet.*

Im Beispiel *Leitung* wird die Konstellation eines Objektes der Klasse *Leitung* allein durch den Abstand und den Winkel zwischen der Grundlinie einer *Zahl* und einer *Linie* bestimmt. Die konkrete geometrische Lage des *Leitung*sobjektes und der Wert der *Zahl* sind dagegen nicht für die Existenz eines *Leitung*sobjektes von Belang. Der Definitionsbereich der Klasse *Leitung* wird deshalb mittels

$$const_{Leitung} : \mathrm{N} \times line \times \mathrm{N} \times \mathrm{N} \rightarrow \mathrm{N} \times \mathrm{N} \qquad \textbf{(3-16)}$$

auf die letzten beiden Parameter eingeschränkt. Das heißt in diesem Fall gilt: $a = \{3,4\}$.

Die Entscheidung, ob ein Objekt existiert, wird im Allgemeinen nicht lokal zu treffen sein. Auch die Konstellation der Bestandteile eines Objektes ist nur ein Anhaltspunkt für diese Entscheidung. Deshalb wird die bedingte **Wahrscheinlichkeit** der Existenz eines Objektes einer Klasse c unter der Bedingung der durch $const_c$ selektierten Merkmale dem statischen Modell hinzugefügt.

Definition 13. *Die bedingte **Wahrscheinlichkeit** $P_c : WerteBereich(const_c) \rightarrow [0,1]$ der Existenz eines Objekts o der Klasse c unter der Bedingung daß die signifikanten Parameter $const_c(p)$ gemessen wurden, sei durch*

$$P_c(const_c(p)) := P(\exists o \in c \mid const_c(p)) \qquad \textbf{(3-17)}$$

gegeben.
Für die bedingte Wahrscheinlichkeit eines konkreten Objektes o mit den Parametern p_o wird die Bezeichnung P_o verwendet:

$$P_o := P(o \in c \mid const_c(p_o)). \qquad \textbf{(3-18)}$$

Die Wahrscheinlichkeitsfunktion P_c gehört zum Modell der jeweiligen Klasse und wird diskretisiert und für einen relevanten Definitionsbereich in einer Tabelle gehalten. In Kapitel 0 wird gezeigt, wie man diese Wahrscheinlichkeiten näherungsweise durch einen überwachten

Lernprozeß erhält. Die fast sicheren und fast unmöglichen Ereignisse mit den Wahrscheinlichkeiten 0 bzw. 1 können in der Praxis nicht auftreten, weil der Lernprozeß die bedingten Häufigkeiten statt der Wahrscheinlichkeiten ermittelt.
Im Gegensatz zu einer bedingten Wahrscheinlichkeit P_o existiert für die Objekte meist keine a-priori-Wahrscheinlichkeit $P(o)=P(o|\text{Domäne})$, weil kein entsprechendes Ereignis sinnvoll definiert werden kann (Die Wahrscheinlichkeit der Existenz einer Zahl „42.73“ auf einem Leitungsplan der Firma X).
Die bedingte Wahrscheinlichkeit P_c subsumiert folgende Einflüsse, die zu Unsicherheit bei Entscheidungen führen:

- Unvollständigkeit des Kontextes (Zur sicheren Erkennung der Ziffer ‚0‘ in Abgrenzung zu Buchstaben und anderen Zeichen ist mindestens die Analyse der benachbarten Zeichen notwendig),
- Unterschiedliche Häufigkeit des Auftretens der Klassen (Die Ziffer ‚0‘ tritt vermutlich häufiger als der Buchstabe ‚O‘ auf. Unter dieser Bedingung gilt für zwei Objekte *Ziffer0* und *BuchstabeO* meist: $P_{Ziffer0} > P_{BuchstabeO}$.),
- prinzipielle Schwächen bei der Gewinnung und Auswahl von Merkmalen (Die unterschiedlichen Krümmungen an der Ziffer ‚5‘ wurde nicht als wesentliches Merkmal zur Unterscheidung vom Buchstaben ‚S‘ extrahiert und ausgewertet.),
- signifikante Abweichungen vom Modell (Der Zeichner hat sich nicht exakt an die Zeichenvorschrift gehalten. Die Objektsemantik kann deshalb nicht zweifelsfrei bestimmt werden.),
- Störungen, Rauschen die aus der Vorverarbeitung und Vorlagenqualität resultieren.

Die $const_c$ Funktion kann nun unter einem stochastischen Blickwinkel betrachtet werden. Sie muß so gewählt werden, daß folgende bedingte Unabhängigkeit für alle Klassen $c \in C$ gilt:

$$\textbf{(I)} \qquad \forall p \in Para_c : P(\exists o \in c \mid const_c(p)) = P(\exists o \in c \mid p)\,. \qquad \textbf{(3-19)}$$

Damit ist auch gesagt, was mit der Formulierung „Auswahl signifikanter Parameter“ in der Definition 11 der Funktion *const*, gemeint ist. Die Einschränkung auf einige Parameter begrenzt den Aufwand, der bei der Bestimmung der Wahrscheinlichkeitsfunktion P_c erforderlich ist. P_c ist nur Funktion einiger weniger Parameter. Für das Beispiel „Leitung“ ergibt sich die in Abbildung Abb. 3-12 dargestellte Wahrscheinlichkeit als Funktion von Winkel und Abstand zwischen *Zahl* und *Linie*. Für die optimale Konstellation der Komponenten – Parallelität der Linien und Abstand nach Zeichenvorschrift, ergeben sich maximale Wahrscheinlichkeiten.

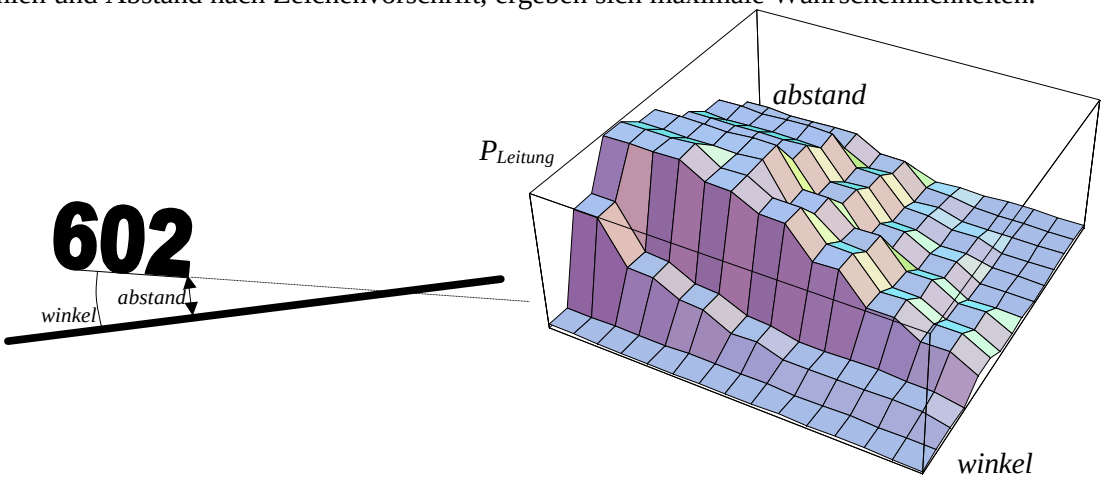


Abb. 3-12 Bedingte Wahrscheinlichkeit der Klasse *Leitung*

Die Parameter, die mittels *const* selektiert werden, müssen so gewählt und durch den Objektgenerator zur Verfügung gestellt werden, daß nur Information, die in der Konstellation

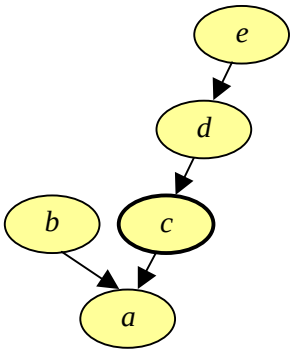
der Teilobjekte, nicht aber in den Teilobjekten selbst, d. h. also gewissermaßen nur neues Wissen durch *P* bewertet wird. Bereits für die Erzeugung der Komponenten benutzte Information darf nicht in noch einmal in die Bewertung einfließen. Die Aussage

$$\textbf{(II)} \qquad \begin{aligned} &\forall p \in Para_c : \forall p' \in \bigtimes_{\forall c' \in C:\, c' \text{ causes } c} Para_{c'} : \\ &P(\exists o \in c \mid const_c(p)) = P(\exists o \in c \mid p, p') \end{aligned} \qquad \textbf{(3-20)}$$

beschreibt eine bedingte Unabhängigkeit und ist bei der Modellierung einer Klasse *c* zu gewährleisten. Sie beinhaltet die Forderung (**I**).
Für das Beispiel *Leitung* hat das die folgende Konsequenz: Die *Zahl*, die in das Modell *Leitung* einfließt, könnte ja auch irrtümlich als *Zahl* klassifiziert worden sein. Beispielsweise könnte ein Wert „90“ bei undeutlicher Vorlage leicht als das englische Wort „go“ interpretiert werden. Bei der Bewertung einer Leitung könnte also diesem Umstand Rechnung getragen werden, indem das Verhältnis der Hauptachsen der Ellipse der Ziffer ‚0‘ in die Parameterliste von $P_{Leitung}$ aufgenommen würde. Dies widerspräche jedoch der Forderung (**II**), denn bei der Klassifikation der Ziffer ‚0‘ ist naheliegender Weise die Verwechslungsmöglichkeit mit dem Buchstaben ‚o‘ bereits in einen Definitieonsbereich eingegangen.
Idealer Weise wird eine über (**II**) hinausgehende Forderung erhoben:

$$\textbf{(III)} \qquad \begin{aligned} &\forall p \in Para_c : \forall p' \in \bigtimes_{c' \in C:\, \neg(c \text{ causes } c')} Para_c : \\ &P(\exists o \in c \mid const_c(p)) = P(\exists o \in c \mid p, p') \end{aligned} \qquad \textbf{(3-21)}$$

Diese Forderung hat rein theoretischen Wert. Sie wird später bei der stochastischen Analyse des hier vorgestellten Modellierungskonzeptes (vgl. 4.2.4.1.2 Abbildung der *causes*-Struktur auf ein Bayesnetz) eine gewisse Rolle spielen. Diese Forderung kann jedoch in der Praxis meist nicht eingehalten werden, weil sie die Lokalität des Klassifizierens untergräbt. So ist das Auftreten einer Ziffer nicht wirklich unabhängig vom Auftreten eines Buchstaben oder einer anderen Ziffer in der unmittelbaren Nachbarschaft.



In der nebenstehende Abbildung werden die Forderungen (**II**) und (**III**) bezüglich der Klasse *c* illustriert: Die Objekte der Klasse *c* sind nach (**II**) bei vorliegenden Objekten der Klasse *d* von den Objekten der Klasse *e* stochastisch unabhängig. Nach (**III**) gilt darüber hinaus eine Unabhängigkeit von *b*.

Nun kann das statische Modell einer Sorte technischer Zeichnungen zusammenfassend definiert werden. Alle klassen-spezifischen Begriffe $x \in \{Para, const, g, parts, P\}$, die man sonst mit einem Klassennamen indiziert, werden dabei gebündelt:

$$x_C := \{x_{c_1} \ldots x_{c_{|C|}}\}. \qquad \textbf{(3-22)}$$

Definition 14. *Das **statische Modell** M_{stat} einer Domäne wird durch eine Menge von Klassen, die Ausgabeschnittstelle, den klassenspezifischen Parametern, Konstellationsfunktionen, Objektgeneratoren, parts-Vektoren und bedingten Wahrscheinlichkeiten beschrieben:*

$$M_{stat} := (C, C_{out}, Para_C, const_C, g_C, parts_C, P_C). \qquad \textbf{(3-23)}$$

Ferner gilt die Forderung (**II**).

Die Relation *causes* gehört nicht explizit zum statischen Modell, weil sie durch die Objektgeneratoren *g* bereits in dieses eingegangen ist.

In diesem Abschnitt wurde der statische Teil des in Y verwendeten Modellierungskonzeptes vorgestellt. Im Weiteren wird gezeigt, wie das Interpretationssystem Y von den Teilmodellen Gebrauch macht, wie es Objekte erzeugt bzw. löscht. Um den Suchraum einzuschränken, kann der Modellierende, den Klassen noch weitere Informationen zuordnen, die die Erzeugungsdynamik betreffen.

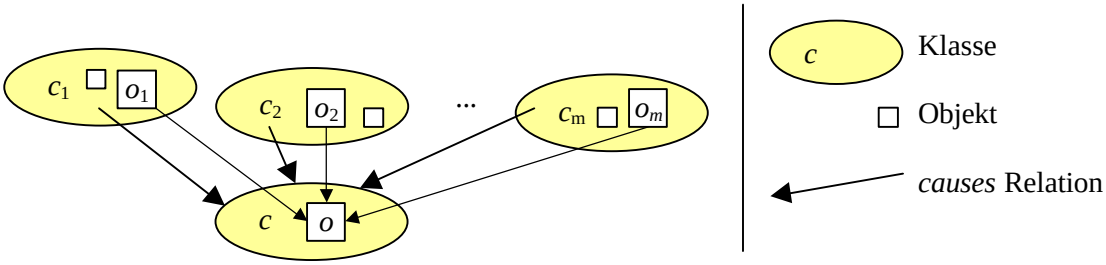
3.2 Dynamisches Modellierungskonzept

In diesem Abschnitt wird ein Algorithmus angegeben, der die Generierung von Objekten während der automatischen Analyse einer konkreten technischen Zeichnung zum Inhalt hat. Zunächst wird beschrieben, wann die Objektgeneratoren der einzelnen Klassen aufgerufen werden. Die dabei erzeugten Objekte werden anschließend bewertet. Gibt es nur eine sehr geringe Wahrscheinlichkeit für die Existenz eines Objektes, so wird dieses sofort wieder gelöscht. Diese Schwellwertentscheidung dient der Effizienzsteigerung. Man wird sehen, daß die Einstellung des Schwellwertes wesentlich unkritischer ist als bei einem Verfahren wie in Abschnitt 2.1 Alternative Ansätze beschrieben wurde. Alle nicht gelöschten Objekte dienen als Grundlage für eine optimale Interpretation, wie sie in Kapitel 4 beschrieben wird.
Das Verständnis des Objektbegriffes muß an dieser Stelle etwas verfeinert werden. Ein Objekt sollte keinesfalls als endgültige Interpretation eines kleinen Zeichnungsbestandteils verstanden werden. Vielmehr handelt es sich bei einem Objekt um eine Hypothese, also eine mögliche Auslegung einer bestimmten Situation in der Zeichnung. Weiterführendes Wissen, d. h. Funktionen über Klassen eines höheren semantischen Niveaus, werden diese Hypothese untermauern oder aber falsifizieren.

3.2.1 Strategien für das Zusammenfassen von Objekten

Das Hauptattribut des in dieser Arbeit diskutierten Verfahrens lautet: Bottom-Up. Die Definitionen für dieses Schlagwort gehen in der Literatur weit auseinander [Russel Norvig 1995]. Hier soll damit folgende Einschränkung gemeint sein:
Die Inhalte einer Zeichnung werden sukzessive entsprechend der impliziten Ordnung der *causes* Relation aus einfacheren Bausteinen zusammengesetzt. Alle Objekte einer Klasse c werden quasi gleichzeitig generiert. Es ist nicht möglich, später, nach der Generierung von Objekten einer Klasse c' mit $c \overline{causes}\ c'$, nach weiteren Objekten der Klasse c zu suchen, obwohl es nun vielleicht Anhaltspunkte gibt, daß solche Objekte existieren müßten. Das bedeutet, daß bereits bei der Anwendung des Modells c alle Objekte, deren Existenz auch nur wenig wahrscheinlich ist, erzeugt werden müssen.
Die Beschränkung auf eine reine Bottom-Up-Strategie wurde eingegangen, weil man so wesentlich freier bei der Gestaltung der Teilmodelle ist. Die Annahmen, die das Gerüst über die Klassen treffen muß, sind relativ schwach. Bottom-Up-Modelle sind tendenziell wesentlich einfacher als Top-Down-Modelle.

Beispiel Top-Down. Eine Klasse c werde durch m Klassen direkt beeinflußt.



Das Bottom-Up-Modell besteht prinzipiell nur aus einer nutzerdefinierten Funktion g_c, die beschreibt, wie ein Objekt der Klasse c aus den Objekten der Klassen c_1 ... c_m

generiert oder durch sie beeinflußt wird und aus einem Prädikat, das über die Existenz, d. h. die Plausibilität des Objektes entscheidet. Ein Gerüst ruft nun im einfachsten Fall für das Kartesische Produkt $c_1 \times c_2 \times \ldots \times c_m$ die Funktion g_c und das Prädikat auf.
Ein allgemeines Top-Down-Modell müßte diesen Vorgang invertieren. Es gibt also die Hypothese, daß ein Objekt o der Klasse c existiert ohne daß alle Komponenten von o bereits identifiziert wurden. Angenommen die Komponenten o_1 und o_m existieren, dann muß eine Suchfunktion $s(o,o_1,o_m)$ gerufen werden, die die restlichen Komponenten $o_2 \ldots o_{m-1}$ finden kann. Es müssen 2^{m-1} Suchfunktionen für das Modell der Klasse c beschrieben werden.
Diese unangenehme Komplexität ließe sich nur reduzieren, wenn strenge Anforderungen an die Funktion g_c (z. B. Invertierbarkeit) und die Parameter der Klassen gestellt würden. Das würde jedoch die Mächtigkeit des Modellierungskonzeptes erheblich einschränken [Pasternak 1996].

Im Gegensatz zur Objektgenerierung kann das Löschen von Objekten auch zu einem späteren Zeitpunkt, wenn mehr Wissen über den Inhalt der Zeichnung extrahiert wurde, vorgenommen werden. Das Erzeugen von zu vielen Objekten ist also unproblematisch. Diese Aussage gilt allerdings nur für die Qualität der Interpretation einer Zeichnung. Die benötigte Zeit zur Analyse einer Zeichnung, kann sich mit der Anzahl der im System befindlichen Objekte rasch erhöhen. Über den Objekten der Ausgabeschnittstelle C_{out} wird später eine NP-harte Optimierung durchgeführt. Insbesondere diese Menge muß also klein gehalten werden.

In diesem Abschnitt werden die Verfahren vorgestellt, welche die Objektgenerierung klassenspezifisch steuern. Sie reduzieren einerseits die Anzahl der erzeugten Objekte stark gegenüber einem trivialen Generieren aller syntaktisch definierten Objekte. Andererseits wird verhindert, daß Objekte, die eine gewisse Wahrscheinlichkeit besitzen, unberücksichtigt bleiben.

Während der Modellierung einer Klasse muß entschieden werden, welcher der im weiteren beschriebenen Algorithmen dieser Klasse zugeordnet wird, das heißt welche Maßnahme zur Reduktion des Suchraumes geeignet erscheint.

3.2.1.1 Klassen mit konstanter Komponentenanzahl

Der Objektgenerator g_c einer Klasse c ist für den gesamten Defintionsbereich $Para_c$ definiert. Damit kann prinzipiell aus allen Kombinationen von Objekten der Klassen c', die in der *causes* Relation zu c stehen, ein neues Objekt der Klasse c generiert werden. Da es aber oft sehr unwahrscheinlich ist, daß geometrisch weit voneinander entfernte Objekte sinnvoll miteinander aggregiert werden können, sollte nicht für alle Kombinationen von Objekten der Komponentenklassen c' der Objektgenerator aufgerufen werden.

Beispiel Umgebung. Es werden wiederum *Leitung*en aus *Strich*en und *Zahl*en aggregiert. Nur *Strich*e, die in der geometrisch-lokalen Umgebung einer *Zahl* existieren, kommen als Teile einer *Leitung* in Betracht.

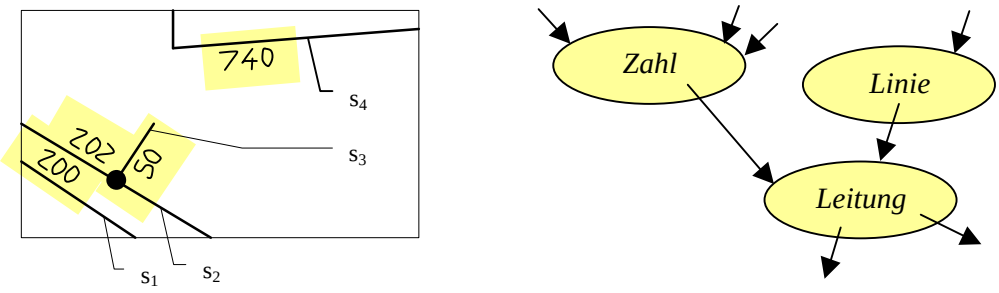


Abb. 3-13 Umgebungen von *Zahl*en, in denen entsprechend des Modells *Leitung* eine *Linie* erwartet wird

Folgende Mengen von *Strich*en werden den *Zahl*en in der Abb. 3-13 zugeordnet:
200..... s_1 und s_2,
202..... s_2 und s_3,
50..... s_2 und s_3,
740..... s_4.
Damit werden 2+2+2+1 *Leitung*sobjekte als Hypothesen erzeugt. Würde auf die beschriebene Suchraumeinschränkung verzichtet werden, müßten 4×4 *Leitung*sobjekte erzeugt werden.

Einer Klasse c kann eine Menge von Funktionen ***environ*** zugeordnet werden, die ausgehend von einem Objekt o von c_1 alle Objekte der Klassen $c_2 ... c_{m_c}$ liefert, die sich in einer geometrischen Umgebung des Objektes o befinden.
Später in diesem Abschnitt werden dann weitere Funktionen definiert, die einer Klasse anstatt der ***environ*** Funktionen zugeordnet werden können, um den Suchraum noch adäquater einzuschränken.

Definition 15. *Einer Klasse c kann während der Modellierung eine Menge von m_c-1 Funktionen **environ**$_c$ zugeordnet werden.*

$$\forall i \in \{2...m_c\}:$$

$$environ_{c,c_i} : c_1 \rightarrow 2^{c_i} \text{ mit } c_i \text{ causes } c, \qquad \textbf{(3-24)}$$

$$\text{d. h. } environ_{c,c_i}(o_1) = \{o_2...o_k\} \text{ mit } o_1 \in c_1 \text{ und } o_2...o_k \in c_i.$$

$environ_{c,c_i}$ ist so zu gestalten, daß alle Objekte der Klasse c_i geliefert werden, die sich in einer sinnvollen geometrischen Umgebung zum entsprechenden Objekt der Klasse c_1 befinden.

Mittels dieser Funktionen kann nun ein effektiver Algorithmus angegeben werden, der die Generierung der Objekte einer Klasse c vornimmt. Diesem Algorithmus soll vorher zum Vergleich das triviale Verfahren der Kartesisches Produktbestimmung vorangestellt werden. Dieses Verfahren kommt ohne die Definition der *environ* Funktionen aus.

Durch den Objektgenerator einer Klasse c

$$g_c : Para_{c_1} \times Para_{c_2} \times ... \times Para_{c_{m_c}} \rightarrow Para_c \qquad \textbf{(3-25)}$$

ist festgelegt, welche Objekte der Klasse c in der Vorlage prinzipiell erzeugbar sind, nämlich die Menge aller Kombinationen der Objekte der Klassen $c_1 ... c_{m_c}$:

$$c_1 \times c_2 \times ... \times c_{m_c} . \quad \textbf{(3-26)}$$

Der zugehörige Algorithmus, der die Objekte einer Klasse c aus bereits existenten Objekten ableitet, kann sofort angegeben werden:

Algorithmus 1 Kartesisches Produkt$_c$.

In: Klasse $c = \emptyset$, Klassen $c_i \neq \emptyset$ und c_i *causes* c	**Out:** Klasse $c \neq \emptyset$

$$\forall o_1 \in c_1 : \forall o_2 \in c_2 : ... \forall o_{m_c} \in c_{m_c} :$$

Generiere ein Objekt o der Klasse c mit den Parametern

$$p_o := g_c(p_{o_1}, p_{o_2} ... p_{o_{m_c}}). \quad \textbf{(3-27)}$$

if (*unwahrscheinlich*(*o*))

Lösche das Objekt o.

Die im Algorithmus verwendete globale Funktion *unwahrscheinlich*(.) wird im nächsten Abschnitt genau definiert. Sie liefert im Fall einer geringen Wahrscheinlichkeit den Wert *true* (Schwellwertentscheidung). Für viele der erzeugten Objekte, existiert nämlich bereits genügend lokal in den Klassen formuliertes Wissen, welches deren Existenz extrem unwahrscheinlich macht. Diese Objekte können sofort wieder gelöscht werden. Alle anderen Objekte bleiben als Hypothese im System existent. Im Beispiel „Umgebung“ kann so bereits lokal entschieden werden, daß die *Zahl* „50“ nicht zum *Strich* s_4 passen kann.

Entsprechend der im Beispiel „Umgebung“ skizzierten Idee und unter Benutzung der *environ* Funktionen, wird nur eine wesentlich kleinere Menge von Objekten als im Algorithmus „Kartesisches Produkt“ generiert. Es ergibt sich das folgende Verfahren, das sich in der ersten Zeile vom Verfahren „Kartesisches Produkt“ unterscheidet.

Algorithmus 2 Umgebung$_c$.

In: Klasse $c = \emptyset$ Klassen $c_i \neq \emptyset$ und c_i *causes* c	**Out:** Klasse $c \neq \emptyset$

$$\forall o_1 \in c_1 : \forall o_2 \in environ_{c,c_2}(o_1) : ... \forall o_{m_c} \in environ_{c,c_{m_c}}(o_1) :$$

Generiere ein Objekt o der Klasse c mit den Parametern :

$$p_o := g_c(p_{o_1}, p_{o_2} ... p_{o_{m_c}}). \quad \textbf{(3-28)}$$

if (*unwahrscheinlich*(*o*))

Lösche das Objekt o.

Welche stochastischen Forderungen an die klassenspezifische Definition der *environ* Funktionen zu stellen sind, wird in Abschnitt 3.2.2 Lokale Klassifikation beschrieben.
Die Klasse c_1 nimmt eine steuernde Rolle bei der Auswahl der Objekte der anderen Klassen c_2 bis c_m ein. Dadurch läßt sich nicht nur der Zeit- und Speicherbedarf der Objekterzeugung reduzieren, sondern auch eine Strategie des Menschen bei der Bildinterpretation nachempfinden: Sind die an einem Objekt der Klasse c beteiligten Objekte unterschiedlich zuverlässig erkennbar, so wählt man eine besonders gut erkennbare Klasse aus und verwendet diese als c_1 in obigem Algorithmus. Diese *Aufmerksamkeitssteuerung* kann beispielsweise dazu

dienen, in der folgenden Abbildung das vierblättrige Kleeblatt sehr schnell zu finden (ohne etwa alle vierelementigen Teilmengen von *Niere*n zu generieren). Die geometrische Klasse *Kreis* wird als c_1 eingesetzt. Nur Blätter in der Umgebung eines *Kreis*es werden als Teile eines *KleeVier* in Betracht gezogen.

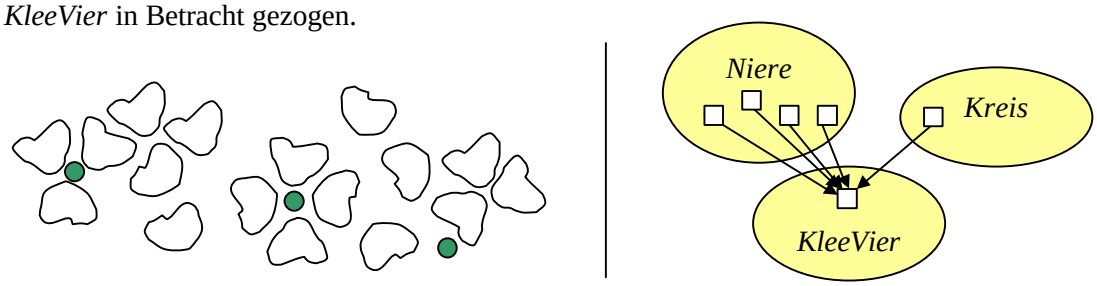


Abb. 3-14 Das Finden von Klee wird durch die leicht erkennbaren Kreise gesteuert

Durch die verringerte Anzahl von Aufrufen des Objektgenerators gegenüber dem Algorithmus „Kartesisches Produkt“ kann auch ein Gedanke implementiert werden, der sonst zur besonderen Charakteristik von Top-Down-Ansätzen gehört: Das Vorliegen eines Objekts der Klasse c_1 läßt die Existenz der übrigen für c notwendigen Objekte als wahrscheinlich erscheinen. Es kann ein besonders zuverlässiger aber langsamer Algorithmus zum Finden der Objekte der anderen Klassen c_2 bis c_m verwendet werden. Im Beispiel von Abb. 3-14 wirkt sich die geringe Häufigkeit der *Kreis*e und der Umstand, daß nur ein Kreis zu einem *Kleevier* gehört, positiv auf das Zeitverhalten der Suche aus.

Ein recht universelles bisher nicht betrachtetes Modellierungskonzept stellt die Generalisierung dar. Beispielsweise können die verschiedensten Bemaßungsarten von einem abstrakten Standpunkt aus als gleich angesehen werden: Eine Bemaßung beschreibt den Abstand zwischen zwei Punkten in einer bestimmten Projektion, egal ob ein Maßpfeil oder eine Maßkette die graphische Repräsentation bilden. Ein Zusammenfassungsalgorithmus, der einer Klasse c zugeordnet werden kann, wenn diese eine Generalisierung der Klassen $c_1 \ldots c_{m_c}$ bedeutet, lautet:

Algorithmus 3 Generalisierung$_c$.

In: Klasse $c = \varnothing$, Klassen $c_i \neq \varnothing$ und c_i *causes* c	**Out:** Klasse $c \neq \varnothing$

$\forall o_1 \in c_1$:
  Generiere ein Objekt o der Klasse c mit den Parametern :
  $p_o := g_{c,c_1}(p_{o_1})$.
  if (*unwahrscheinlich*(*o*))
    Lösche das Objekt o.
    ⋮ **(3-29)**
$\forall o_{m_c} \in c_{m_c}$:
  Generiere ein Objekt o der Klasse c mit den Parametern :
  $p_o := g_{c,c_{m_c}}(p_{o_{m_c}})$.
  if (*unwahrscheinlich*(*o*))
    Lösche das Objekt o.

Der Objektgenerator g_c entartet hier zu einer Menge von Funktionen $g_{c,c_1}...g_{c,c_{m_c}}$. Für jede Komponentenklasse $c_1...c_{m_c}$ wird ein eigener Generator benötigt, der die Parameter dieser Klasse auf die Parameter der allgemeineren Klasse *c* abbildet. Durch die Indizierung der Algorithmenbezeichnung „Generalisierung$_c$“ mit einer Klasse *c* wird auch hier symbolisiert, daß eine klassenspezifische Definition von Funktionen vorgenommen werden muß.
Neben den drei bisher betrachteten Zusammenfassungsalgorithmen sind weitere denkbar. Die Beschaffenheit solcher Verfahren ist anwendungsabhängig und damit Teil des Modellierungsprozeßes. Man wird jedoch weitaus häufiger neue Klassen entwickeln, als die Menge der Zusammenfassungsalgorithmen zu erweitern. Im weiteren sollen diese Verfahren als ***Metaklassen*** bezeichnet werden, weil jeder einzelne soeben beschriebene Algorithmus die gemeinsame Dynamik einer Vielfalt von Klassen beschreibt. Um bestimmte ***Metaklassen*** nutzen zu können, müssen die Klassen oft einige Funktionen deren Definitions- und Wertebereich der ***Metaklasse*** bekannt ist, konkret definieren.

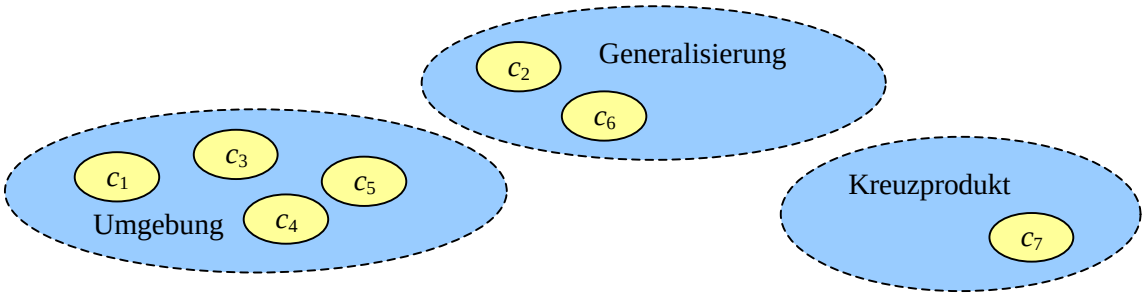


Abb. 3-15 Zuordnung einer Dynamik zu Klassen mittels *Metaklassen*

Dies ist ein objektorientiertes Muster: Eine Klasse erbt die Funktionalität einer (abstrakten) ***Metaklasse*** und implementiert lediglich einige virtuelle Funktionen.

Eine wichtige Menge von ***Metaklassen*** wurde noch nicht angesprochen. Diese ***Metaklassen*** dienen der Behandlung von Klassen mit einer variablen Anzahl von Komponenten.

3.2.1.2 Klassen mit variabler Komponentenanzahl

Viele Elemente von technischen Zeichnung bestehen aus einer unbestimmten Anzahl von Komponenten. Gerissene Linien, Zahlen oder Zeichenketten müssen so modelliert werden, daß während der Analyse die Anzahl der Teilobjekte, also der Linienabschnitte, der Ziffern sowie der Buchstaben ermittelt wird. Als Beschreibungsmittel solcher Situationen bieten sich Rekursion und Iteration an. Da der Klassengraph bestehend aus Klassen und der *causes* Relation zyklenfrei sein muß, können keine Rekursionen modelliert werden. Ein entsprechender Objektgenerator für eine Klasse *Zahl* hätte eine der Definition 6 widersprechende Struktur:

$$g_{Zahl}^{start} : Para_{Ziffer} \rightarrow Para_{Zahl}$$
$$g_{Zahl}^{recursion} : Para_{Zahl} \times Para_{Ziffer} \rightarrow Para_{Zahl}. \qquad \textbf{(3-30)}$$

Der rekursive Teil von g_{Zahl} impliziert, daß die Klasse *Zahl* mit sich selbst in der *causes* Relation steht. Damit wird die Irreflexivität der *causes* Relation verletzt.
Dagegen kann eine unbestimmte Komponentenanzahl m_{Zahl} unproblematisch in das bereits beschriebene statische Modellierungskonzept integriert werden. Der Objektgenerator entspricht genau der Definition:

$$g_{Zahl} : \bigtimes_{i=1}^{m_{Zahl}} Para_{Ziffer} \rightarrow Para_{Zahl}. \qquad \textbf{(3-31)}$$

Neben dem wichtigen Konzept einer variablen Komponentenzahl, ist folgende Beobachtung bei Objekten mit einer flexiblen Größe verallgemeinerungswürdig: Versucht man die *Ziffer*n, die potentiell zu einer zehnstelligen *Zahl* gehören, mittels des Algorithmus „Umgebung“ zu finden, ergibt sich zumindest dann ein Problem, wenn man annimmt, daß die Zahlen nicht auf einer exakten Linie stehen müssen. Sie können auch an eine gekrümmte Kurve geschrieben worden sein. Die geometrische Position der *n*-ten *Ziffer* ist dann sicher nur wenig von der ersten *Ziffer* abhängig. Jedoch die Ausrichtung und Lage der (*n*-1)-ten *Ziffer* ist ein starkes Indiz für die Lageparameter der *n*-ten *Ziffer*. Für iterative Modelle sind also korrespondierende Algorithmen zum Verfahren „Umgebung“ zu entwickeln. Diese Algorithmen verlangen auch einen etwas modifizierten Umgebungsbegriff:

Definition 16. *Einer Klasse c kann während der Modellierung eine Funktion* ***environNext***$_c$ *sowie eine Funktion* ***environPrev***$_c$ *zugeordnet werden:*

$$environNext_c : c_1 \rightarrow 2^{c_1} \text{ mit } c_1 \text{ causes } c,$$
$$environPrev_c : c_1 \rightarrow 2^{c_1} \text{ mit } c_1 \text{ causes } c. \qquad \textbf{(3-32)}$$

Damit wird c zu einer iterativen Klasse. Nur genau eine Klasse c_1 darf in der causes Relation zu c stehen. $EnvironNext_c(o)$ liefert dabei potentielle Nachfolger von o. Mittels $EnvironPrev_c(o)$ wird die Menge der Vorgänger erzeugt.
Um alternative Fortsetzungen von o miteinander vergleichen zu können, ist eine weitere Funktion ***better***$_c$ *zu implementieren, welche die beste Fortsetzung der Kette liefert:*

$$better_c : c_1 \times c_1 \times c_1 \rightarrow c_1 \cup \{stop\} \quad mit$$
$$better_c(o, o_1, o_2) = \begin{cases} o_1 & o_1 \text{ ist eine bessere Fortsetzung von } o \text{ als } o_2 \text{ oder aber } o_1 \text{ und } o_2 \text{ sind identisch.} \\ o_2 & o_2 \text{ ist eine bessere Fortsetzung von } o \text{ als } o_1. \\ stop & \text{Lokal kann keine sinnvolle Entscheidung zwischen } o_1 \text{ und } o_2 \text{ getroffen werden, oder beide Alternativen können nicht akzeptiert werden.} \end{cases} \qquad \textbf{(3-33)}$$

Ein zugehöriger Zusammenfassungsalgorithmus, der sich dieser Funktionen bedient, kann leicht angegeben werden. Die Menge *c* soll dabei eine disjunkte Zerlegung der Menge c_1 bilden. Am Anfang wird eine Kopie der Klasse c_1 in eine Menge *A* geschrieben. Alle Objekte von c_1, die im Laufe des Verfahrens in ein Objekt der Klasse *c* eingehen, werden aus *A* entfernt. So wird erreicht, daß einerseits alle Objekte von c_1 die Chance bekommen, ein Teil eines Objektes von *c* zu werden und andererseits keines dieser Objekte Teil zweier Objekte vom Typ *c* werden kann. So werden auch Zyklen verhindert und ein Terminieren des Algorithmus erzwungen. In der Menge *B* werden stets die Objektteile gesammelt, die zu einem Objekt der Klasse *c* zusammengefaßt werden sollen.
Ausgehend von einem beliebigen Objekt der Komponentenklasse c_1 wird zunächst vorwärts nach weiteren Komponenten gesucht bis entweder kein weiteres Objekt mehr gefunden wird oder bis die Fortsetzung nicht mehr (eindeutig) möglich ist. Anschließend wird ab dem Label *prev* ein rückwärtige Suche vorgenommen.

Beispiel Iteration. Es sollen Punkte zu Ketten zusammengefaßt werden. Existiert eine starke Annahme, daß Punktketten nur geringe Krümmungen aufweisen, dann kann das Ergebnis der disjunkten Zerlegung der Punktwolke der Abb. 3-16 b) entsprechen. Existieren keine solche Annahmen, ist also ein Ergebnis c) erwünscht, implementiert man *better*(.) so, daß die Funktion stets *stop* zurückgibt.

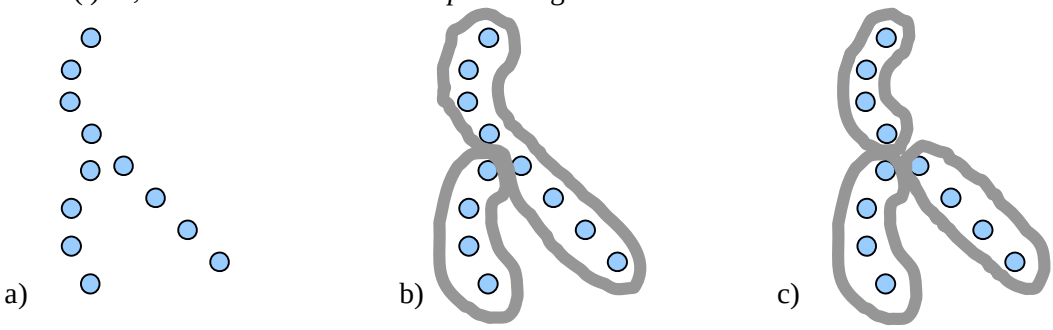


Abb. 3-16 Alternative disjunkte Zerlegungen einer Punktwolke

Algorithmus 4 Iteration1$_c$.

In: Klasse $c = \emptyset$, Klasse $c_1 \neq \emptyset$ und c_1 *causes* c **Out:** Klasse $c \neq \emptyset$

$A := c_1$

$\forall o \in A$:

$B := \{o_{cur} := o\}$

$\forall (o_1, o_2) \in envionNext_c(o_{cur}) \times envionNext_c(o_{cur})$:

$if\ (better_c(o_{cur}, o_1, o_2) = stop)\ goto\ prev$

$o_{cur} := better_c(o_{cur}, o_1, o_2)$

$B := B \cup \{o_{cur}\},\ A := A - \{o_{cur}\}$

prev:

$B := \{o_{cur} := o\}$

$\forall (o_1, o_2) \in envionPrev_c(o_{cur}) \times envionPrev_c(o_{cur})$:

$if\ (better_c(o_{cur}, o_1, o_2) = stop)\ goto\ end$

$o_{cur} := better_c(o_{cur}, o_1, o_2)$

$B := B \cup \{o_{cur}\},\ A := A - \{o_{cur}\}$

end:

Generiere ein Objekt o der Klasse c mit den Parametern :

$p_o := g_c(B).$

$if\ (unwahrscheinlich(o))$ **(3-34)**

Lösche das Objekt o.

$A := A \cup B$

In diesem Algorithmus ist die folgende abkürzende Schreibweise für die Parameter eines Objektgenerators g_c verwendet worden:

$$g_c(B) = g_c(p_{o_1} ... p_{o_{|B|}}) \text{ mit } B = \{o_1 ... o_{|B|}\}. \quad \textbf{(3-35)}$$

Die Wirkungsweise dieses Verfahrens „Iteration1“ soll nun an einem Beispiel demonstriert werden. Hierbei werden lange, geradlinige, gerissene Linien aus kurzen Linien generiert.

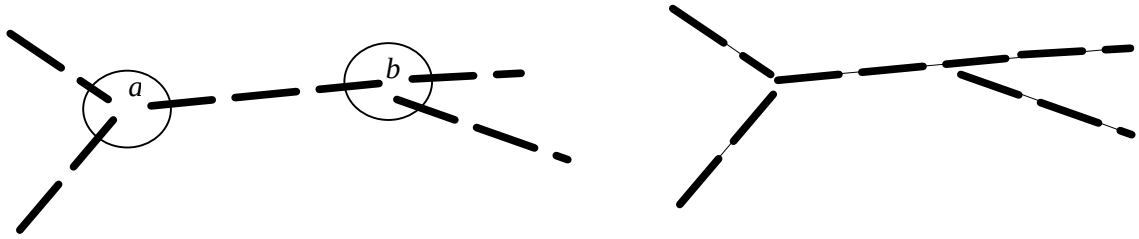


Abb. 3-17 Links Vorlage; rechts Ergebnis der „Iteration1“, mittels durchgezogener Linien werden erkannte gerissene Linien symbolisiert

Die Funktion $better_c$ definiert also Verzweigungspunkte bzw. allgemein unsichere Situationen für eine Klasse *c*. Im Namen des Algorithmus „Iteration1“ bedeutet die Eins, daß zwischen zwei solchen unsicheren Situationen genau ein Objekt erzeugt wird. In der Abbildung sind diese Situationen mit *a* und *b* bezeichnet. Die Menge *B* der Komponenten wird als Ganzes an den Objektgenerator weitergereicht.
Mit diesem Ansatz kann ein Objektverkettung entlang einer Dimension modelliert werden. Höherdimensionale Verkettungen können durch Nacheinanderausführung dieses Verfahrens modelliert werden.

Nicht immer kann eine Klasse *c* als sinnvolle disjunkte Zerlegung der Menge c_1 lokal definiert werden. Das heißt, es kann passieren, daß nicht alle Objekt-Hypothesen erzeugt werden, die später – in anderen Klassen – benötigt werden. Eine höhere Flexibilität wird erreicht, wenn man folgende Erweiterung des Algorithmus „Iteration1“ betrachtet:
Man erzeuge zu jedem Objekt *o*, das bereits durch „Iteration“ erzeugt worden wäre, die Menge aller Objekte, die aus einer zusammenhängenden Teilmenge der Komponenten von *o* bestehen. Im rechten Teil der Abb. 3-18 sind alle Objekte einer Klasse *gerissene Linie* dargestellt, die aus einer Vorlage wie sie links dargestellt ist, erzeugt werden. Alle geometrisch übereinander liegende Linien sind in einem gemeinsamen Rechteck aber nebeneinander visualisiert worden. Die erzeugten *gerissenen Linien* im rechten Teil der Abbildung sind als durchgezogenen Linien dargestellt!

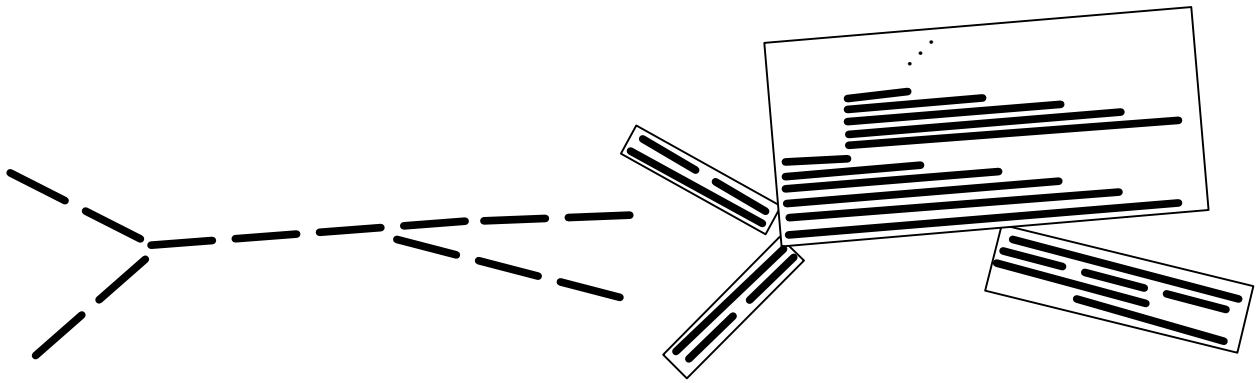


Abb. 3-18 Links Vorlage, bestehend aus kurzen Strichen; rechts gerissene Linien, mittels durchgezogener Linien symbolisiert

Die Bedeutung dieser Variante des Algorithmus „Iteration“ kann gut an einem bereits zitierten Beispiel zur Erkennung von *Zahl*en bzw. *Wort*en gezeigt werden.

Beispiel 100Stahl. Viele Buchstaben ähneln bei lokaler Betrachtung anderen Buchstaben oder gar Ziffern. Deshalb ist bei Betrachtung der Zeichenkette in der Abbildung nicht zu erwarten, daß die Frage, ob das Symbol S eine ‚5‘ oder den Buchstaben ‚S‘ darstellt,

zuverlässig auf dem Niveau der *Ziffer*n und *Buchstabe*n zu beantworten ist. Erst wenn die Objekte der Klasse *Material* erzeugt worden sind, kann eine sinnvolle Entscheidung getroffen werden. Dann werden die zuvor entdeckten *Wort*-Hypothesen (Ketten von *Buchstabe*n) mit einem Wörterbuch verglichen. Der *Material*bezeichner „Stahl“ sei in diesem Wörterbuch enthalten.

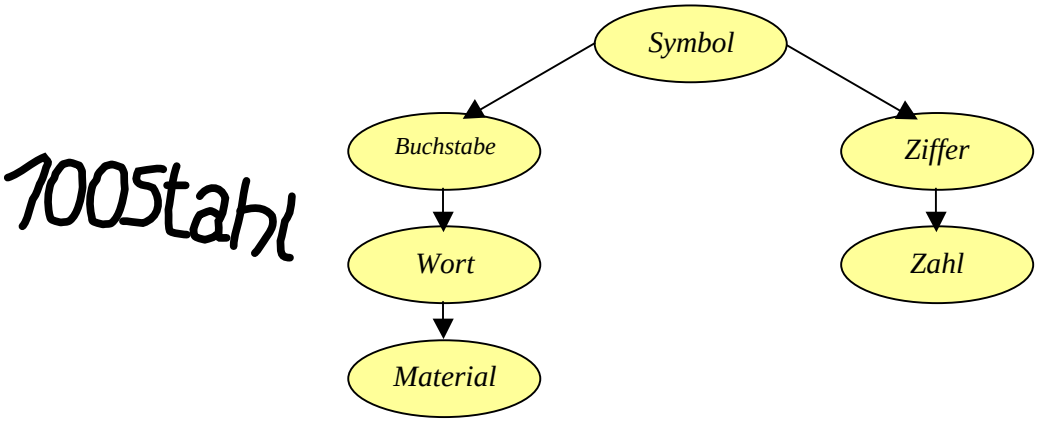


Abb. 3-19 Links: Teil einer Leitungsbeschriftung, rechts relevante Klassen und *causes* Relation

Da das hier entwickelte Analysesystem eine reine Bottom-Up-Strategie verfolgt, müssen auf der Ebene der *Worte* und *Zahl*en alle möglichen Varianten generiert werden:

Von unten lesbare *Worte*	O, OO, OOS, OOSt ... OOStahl, O, OS, OOSt ...OStahl ... l
Von oben lesbare *Worte*	S,SO,SOO O,OO, O,
Von unten lesbare *Zahl*en	1,10,100,1005, 0,00,005, 0,05, 5
Von oben lesbare *Zahl*en	5,50,500, 0,00, 0

Damit wird sichergestellt, daß tatsächlich alle Objekte generiert werden, die in komplexeren Klassen interpretiert werden könnten. Später wird man sehen, wie die optimale Interpretation des in Abb. 3-19 dargestellten Planausschnittes als „100 Stahl“ gefunden wird.

Nun kann das allgemeine Verfahren „IterationN^2“ angegeben werden, das im Unterschied zu „Iteration 1“ für jede Menge *B* eine Vielfalt an Objekten erzeugt. Es werden mit

$$\sum_{i=1}^{|B|} i = \frac{|B|(|B|+1)}{2} \qquad \textbf{(3-36)}$$

vertretbar viele Objekte generiert. Die quadratische Größenordnung dieser für die Laufzeit wichtigen Zahl führt zu der Bezeichnung „Iteration$\mathbf{N^2}$“. Der Algorithmus unterscheidet sich nach dem Label *end* von dem „Iteration1“ Verfahren.

Algorithmus 5 Iteration$\mathbf{N^2_c}$.

In: Klasse $c = \emptyset$, Klasse $c_1 \neq \emptyset$ und c_1 *causes* c **Out:** Klasse $c \neq \emptyset$

$A := c_1$
$\forall o \in A$:
  $B := \{o_{cur} := o\}$
  $\forall (o_1, o_2) \in envionNext_c(o_{cur}) \times envionNext_c(o_{cur})$:
    if $(better_c(o_{cur}, o_1, o_2) = stop)$ *goto prev*
    $o_{cur} := better_c(o_{cur}, o_1, o_2)$
    $B := B \cup \{o_{cur}\}$, $A := A - \{o_{cur}\}$
prev:
  $B := \{o_{cur} := o\}$
  $\forall (o_1, o_2) \in envionPrev_c(o_{cur}) \times envionPrev_c(o_{cur})$:
    if $(better_c(o_{cur}, o_1, o_2) = stop)$ *goto end*
    $o_{cur} := better_c(o_{cur}, o_1, o_2)$
    $B := B \cup \{o_{cur}\}$, $A := A - \{o_{cur}\}$
end:
  $\forall B' \in$ *Menge aller zusammenhängenden Teilmengen von* B:
    Generiere ein Objekt o der Klasse c mit den Parametern :
    $p_o := g_c(B')$.
    if (*unwahrscheinlich*(*o*)) **(3-37)**
      Lösche das Objekt o.
      $A := A \cup B'$

Der Vollständigkeit halber muß auch ein Algorithmus zur Generierung der Objekte der primitiven Klassen, also der Klassen ohne Vorgänger, angeben werden. Diese Objekte und ihre Parameter müssen extern festgestellt werden. Sie können z. B. das Ergebnis einer einfachen Vektorisierung sein.

Algorithmus 6 Primitive$\mathbf{_c}$.

In: Klasse $c = \emptyset$, $c \hat{I} C_{primitive}$ **Out:** Klasse $c \neq \emptyset$

Generiere alle Objekte o der Klasse c mit den Parametern p_o. **(3-38)**

Die in diesem Abschnitt beschriebenen Gruppierungsalgorithmen können nun formal zusammengefaßt werden.

Definition 17. *Alle Zusammenfassungsalgorithmen bilden die*

$$Metaklassen := \left\{ \begin{array}{c} Kartesisches\ Produkt, \\ Umgebung, Generalisierung \\ Iteration1, IterationN^2, Primitive \end{array} \right\}. \quad \textbf{(3-39)}$$

Jeder Klasse $c \in C$ wird genau eine $metaklasse \in Metaklassen$ zugeordnet. Die Klassenspezifische Implementierung einer metaklasse wird mit dem Namen der Klasse indiziert. Die Menge aller Implementierungen einer Domäne wird mit der Menge C der die Domäne beschreibenden Klassen indiziert:

$$Metaklassen_C = \{metaklasse_c \mid c \in C \wedge metaklasse \in Metaklassen\}. \quad \textbf{(3-40)}$$

3.2.2 Lokale Klassifikation

Ziel dieses Abschnittes ist zu zeigen, wie ein durch einen Objektgenerator erzeugtes Objekt bewertet wird und unter welchen Umständen es sofort wieder gelöscht werden kann. Die in den *Metaklassen* verwendete Funktion *unwahrscheinlich*(.) wird definiert. Es wird ferner hier und im folgenden Abschnitt gezeigt, wie Klassen, die unabhängig voneinander modelliert wurden, miteinander konkurrieren können. Trotz einer sehr lokalen Beschreibung der einzelnen Klassen können auch sehr weitreichende Konflikte in einer Zeichnung aufgedeckt und global gelöst werden.
Zunächst wird untersucht, wie die Frage, ob ein Objekt existiert, auf eine Bayes-optimale Entscheidung abgebildet werden kann. Das dabei entstehende Mehrklassenproblem der Klassifikation wird anschließend auf eine einfache Schwellwertentscheidung reduziert.

Eine Verwechslungssituation ist dadurch charakterisiert, daß eine Menge von Objekten *A* unterschiedlich interpretiert werden kann. Mit *B* sei die Menge aller möglichen alternativen Objekte bezeichnet, deren Komponentenobjekte genau die *A* sind.

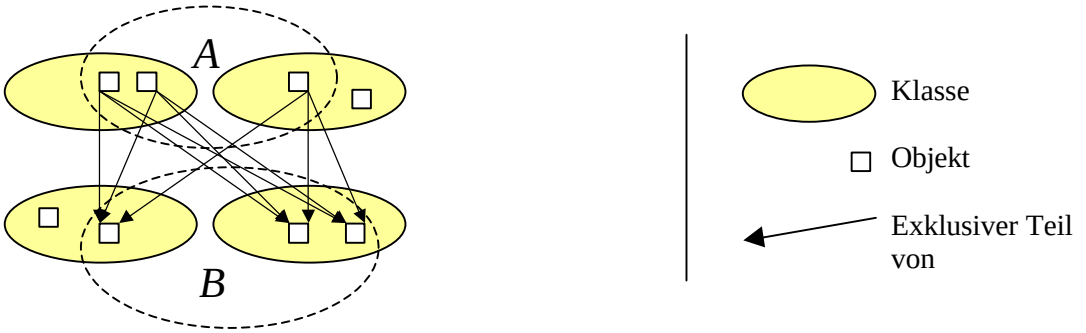


Abb. 3-20 Jedes Objekt der Menge *A* ist Teil von jedem Objekt aus *B*.

Solche Mengen *A* und *B* werden bei der eigentlichen Interpretation einer Zeichnung nicht explizit erzeugt. Sie sind hier ein Hilfsmittel, um einige Eigenschaften von Y zu beschreiben.

Satz 1. *Bei der Klassifikation einer Objektmenge A ohne Rückweisung und mit symmetrischen Kosten ergibt sich die optimale Entscheidung für ein Objekt o_{opt} der Menge B={... o ...} aus den bedingten Wahrscheinlichkeiten P_o der Objekte (vgl. Definition 13):*

$$o_{opt} = \arg\max_{\forall o \in B}(P_o). \tag{3-41}$$

Beweis. Es ist zu zeigen, daß eine Entscheidung, die direkt auf einer maximalen Objektwahrscheinlichkeit $P(o)$ basiert, zum selben Ergebnis führt wie eine Entscheidung, die die bedingte Wahrscheinlichkeit P_o verwendet.
Ein a-priori-Wahrscheinlichkeit für Objekte $P(o)$ existiert nicht. Ein Objekt o (mit fixierten Parametern p_o) existiert nur, wenn auch seine Komponenten, also die Objekte der Menge A existieren. Statt der a-priori-Wahrscheinlichkeit wird die Verbundwahrscheinlichkeit $P(o, A)$ als Kriterium für eine optimale Entscheidung genommen:

$$\begin{aligned} o_{opt} &= \arg\max_{\forall o \in B}(P(o, A)) \\ &= \arg\max_{\forall o \in B}(P(o \mid A)\ P(A)) \\ &= \arg\max_{\forall o \in B}(P(o \mid A)\ P(o' \mid A - o')P(o'' \mid A - o' - o'')\ldots) \text{ mit } o', o'' \ldots \in A. \end{aligned} \tag{3-42}$$

$P(A)$ ist also unabhängig von o, verhält sich also neutral in der argmax Bildung über alle o.

$$o_{opt} = \arg\max_{\forall o \in B}(P(o \mid A)\ ) \tag{3-43}$$

Nun ist ein Objekt o durch seine Merkmale p_o eindeutig bestimmt. Diese sind wiederum eine Funktion der Komponentenobjekte A, weil die Generatoren aller Objekte aus B denselben Definitionsbereich haben. Dieser gemeinsame Definitionsbereich wird mit X seine Elemente mit x bezeichnet.

$$\begin{aligned} o_{opt} &= \arg\max_{\forall o \in B}(P(o \mid A)) \\ &= \arg\max_{\forall o \in B}(P(o \mid x \in X)). \end{aligned} \tag{3-44}$$

Die Objekte und ihre bedingte Wahrscheinlichkeiten sind nur mittelbar eine Funktion der Merkmale x. Der Generator g bildet das Bindeglied zwischen den Merkmalen der Komponenten x und den Parametern p_o des Objektes o.
Die relevanten Parameter werden durch die Funktion *const* selektiert.

$$\begin{aligned} o_{opt} &= \arg\max_{\forall o \in B}(P(o \mid p_o = g_{c_o}(x \in X))) \\ &= \arg\max_{\forall o \in B}(P(o \mid const_{c_o}(p_o)). \end{aligned} \tag{3-45}$$

Nach anschließender Anwendung des zweiten Teils der Definition 13 erhält man:

$$o_{opt} = \arg\max_{\forall o \in B}(P_o). \tag{3-46}$$

■

Das Objekt o_{opt} sollte erzeugt werden, wenn eine endgültige und eindeutige Entscheidung gefällt werden muß. Die Fehlerwahrscheinlichkeit einer solchen Bayes-optimalen Entscheidung erhält man aus:

$$P_{error}^{bayes} = \sum_{x \in X} \sum_{\forall o \in B} p(x)\ p(o \mid x)\ error(o)$$
$$= \sum_{x \in X} p(x) \sum_{\forall o \in B} P_o\ error(o) \quad \text{mit} \qquad \textbf{(3-47)}$$
$$error(o) = \begin{cases} 0 & \text{wenn } o = \arg\max_{\forall o \in B}(P_o) \\ 1 & \text{sonst.} \end{cases}$$

Die Abb. 3-21 illustriert dies am Beispiel der Objekte, die entsprechend einer Bayes-optimalen Entscheidung den Klassen c_1, c_2 bzw. c_3 angehören. Für jeden gemessenen Merkmalsvektor x wird genau ein Objekt erzeugt. Die beiden anderen syntaktisch möglichen Objekte werden nicht generiert, obwohl sie oft eine relativ hohe Wahrscheinlichkeit aufweisen.

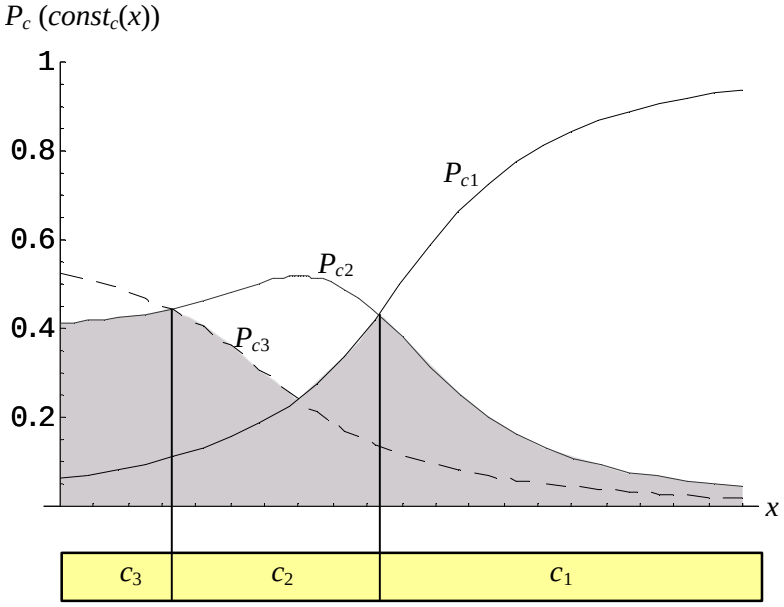


Abb. 3-21 Bayesoptimale Entscheidungen und Menge der Fehlentscheidungen

Die Objektaggregation ist ein mehrstufiger Prozeß. Auch Objekte, die im lokalen Kontext nur eine geringe Wahrscheinlichkeit besitzen, können – unter der Betrachtung von mehr Kontext – verifiziert werden. Denn sie können als Bestandteil eines komplexeren und sehr wahrscheinlichen Objektes Verwendung finden.

Beispiel B-8. In der nebenstehenden Abbildung ist das zweite Symbol einem Buchstaben 'B' sehr ähnlich. Im Kontext der Zeichenkette wird sich jedoch die Interpretation '8' als wahrscheinlich erweisen. Wenn eventuell noch ein Maßpfeil in einer geeigneten Konstellation gefunden werden kann, gibt es keinen Zweifel mehr.
Die Idee und der Name B-8 für dieses Beispiel ist dem Roman „Congo“ von Michael Crichton [Crichton 1993] entnommen.

Eine Alternative zum Bayes-optimalen Entscheiden stellt der **Schwellwertklassifikator** dar. Dabei wird die Entscheidung bei großer Unsicherheit solange vertagt, bis ein größerer Kontext zur Verfügung steht. Dies ähnelt der bekannten Strategie der Rückweisung an einen übergeordneten Klassifikator [Steinhagen Fuchs 1976]. Diese lautet:

$$e_j = \arg\max_{j=0}^{|B|} \begin{cases} P_{o_j} & j = 1 \ldots |B| \\ b & j = 0. \end{cases} \quad \textbf{(3-48)}$$

Eine Entscheidung e_0 bedeutet dabei eine Rückweisung, $e_1 ... e_{|B|}$ zieht eine Generierung des korrespondierenden Objektes $o_1 ... o_{|B|}$ nach sich. Doch auch bei diesem Klassifikator wird maximal ein Objekt erzeugt. Im Fall großer Unsicherheit (Entscheidung e_0) bleibt die volle Mehrdeutigkeit bestehen. Um diesem Dilemma zu entgehen, wird hier ein neuer Zugang vorgeschlagen.

Definition 18. *Ein klassenspezifischer **Schwellwertklassifikator** gibt an, wann ein Objekt als Element einer Klasse generiert wird:*

$$o \in c_o \Leftrightarrow P_o > b_{c_o}. \quad \textbf{(3-49)}$$

Die Menge aller Schwellwerte sei $b_C = \{b_c : c \in C\}$

Das bedeutet, daß bei geringen Wahrscheinlichkeiten durchaus mehrere Objekte der oben genannten Menge B aus einer Menge von Komponentenobjekten A erzeugt werden. Die volle Mehrdeutigkeit des Klassifikators mit Rückweisung im unsicheren Fall, wird durch eine eingeschränkte Mehrdeutigkeit ersetzt. Genauer gesagt werden alle Objekte generiert, die mindestens eine Wahrscheinlichkeit aufweisen, die größer als b ist. Die Abb. 3-22 verdeutlicht diese Überlegung und die verbleibende Fehlerwahrscheinlichkeit:

$$P_{error}^{threshold} = \sum_{x \in X} P(x) \sum_{\forall o \in B} P_o \; error(o) \text{ mit} \quad \textbf{(3-50)}$$

$$error(o) = \begin{cases} 0 & \text{wenn } P_o > b_{c_o} \\ 1 & \text{sonst.} \end{cases}$$

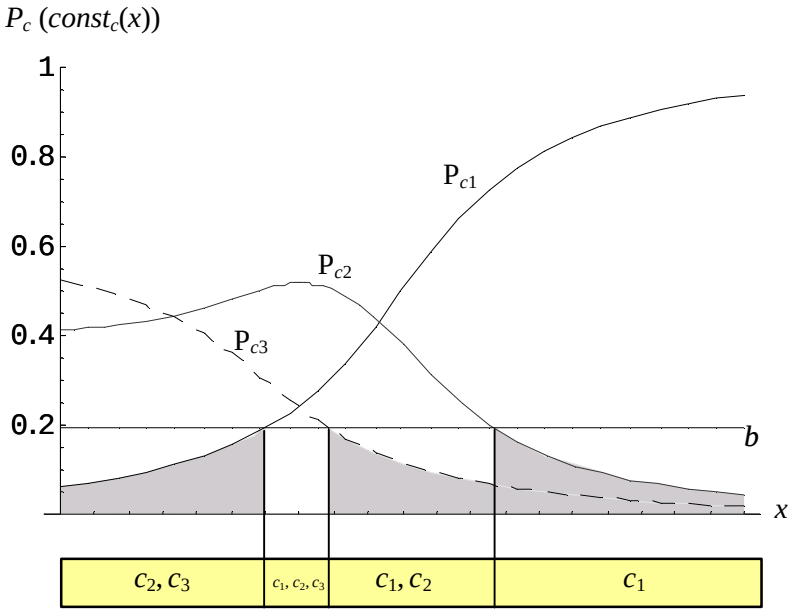


Abb. 3-22 Schwellwertklassifikator und die Fehlerbereiche

Damit der Schwellwertklassifikator nicht „schlechter“ als ein Bayes-optimaler Klassifikator ist, muß für jede Merkmalsausprägung $x \in X$ wenigstens das Objekt erzeugt werden, das auch durch den Bayes-optimalen Klassifikator erzeugt worden wäre. Damit dies erfüllt ist, muß b folgendermaßen gewählt werden:

$$b < \min_{\forall x \in X} \max_{\forall c \in B} P_c(const_c(x)) . \quad \textbf{(3-51)}$$

In Abb. 3-22 würde ein Schwellwert kleiner als etwa 0.4 das Gewünschte leisten.
Um die Fehlerkosten absolut zu minimieren, müßte b = 0 gewählt werden. Das würde jedoch dazu führen, daß stets alle theoretisch möglichen Objekte erzeugt würden. Der Zeitaufwand zum Generieren von Objekten die Anzahl von Objekten und für die abschließende globale Optimierung wäre nur durch die Struktur der *metaklassen* begrenzt und daher unvernünftig hoch.
Die Wahrscheinlichkeit, daß mehr als ein Objekt erzeugt wird, beträgt:

$$P_{twoObjects}^{threshold} = \sum_{x \in X} P(x)\, twoObjects(x) \;\; mit \quad \textbf{(3-52)}$$

$$twoObjects(x) = \begin{cases} 1 & wenn \; \exists o, o' \in B : P_o, P_{o'} \geq b \wedge o \neq o' \\ 0 & sonst. \end{cases}$$

Die Wahrscheinlichkeiten für Fehlentscheidungen und das Generieren von mehr als einem Objekt sind implizit Funktionen von b. Um einen insgesamt optimalen Schwellwert b_{opt} zu bestimmen, muß ein Optimierungsproblem gelöst werden:

$$b_{opt} = \arg\min_b \left(cost_{twoObjects} \; P_{twoObjects}^{threshold}(b) + cost_{error} \; P_{error}^{threshold}(b) \right) \quad \textbf{(3-53)}$$

Die Kosten für Fehlentscheidungen sind dabei $cost_{error}$, die für das Generieren von mehreren Objekten betragen $cost_{twoObjects}$. Die Lösung dieses Problems hängt von der Verteilung der Wahrscheinlichkeiten P_c ab und kann daher nicht allgemein angegeben werden.
Die in den *metaklassen* verwendete Funktion *unwahrscheinlich*(.) wird entsprechend des Schwellwertklassifikators definiert und entscheidet über die dauerhafte Existenz eines Objektes in den klassenspezifischen Algorithmen.

Definition 19. *Die Funktion unwahrscheinlich*: $O \rightarrow \{True, False\}$ *entscheidet unter Nutzung des Schwellwertklassifkators und benutzt dabei einen klassenspezifischen Schwellwert* b_c*:*

$$unwahrscheinlich(o) := \begin{cases} True & P_o \leq b_{c_o} \\ False & sonst \end{cases} . \quad \textbf{(3-54)}$$

Damit kann eine stochastisch motivierte Forderung an die Entwicklung und den Einsatz der Metaklassen gestellt werden. Den Output einer der Klasse *c* zugeordneten Metaklasse, müssen die selben Objekte bilden, die das triviale Verfahren „Kartesisches Produkt“ generieren würde:

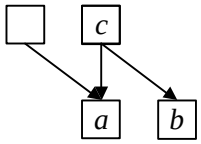
(IV) $Kartesisches\ Produkt_c = metaklasse_c \quad \textbf{(3-55)}$

Dies bedeutet, daß wirklich alle entsprechend des Schwellwertklassifkators möglichen Objekte, also die mit einer Wahrscheinlichkeit größer b_c erzeugt werden. Die gewählte Metaklasse soll die Effizienz steigern, nicht aber die Qualität beeinträchtigen.

3.2.2.1 Vorteile des Schwellwertklassifikators

Für die Bayes-optimale Entscheidung ist es notwendig, alle alternativen Objekte bzw. deren Wahrscheinlichkeit direkt miteinander zu vergleichen. Ein großer Vorteil des Schwellwert-klassifikators ist, daß er für jede Klasse separat definiert ist. Die Parameterräume der

alternativen Klassen müssen nicht aufeinander abgestimmt sein. So kann der Objektgenerator der Klasse Buchstabe 'B' im Beispiel B-8 völlig unabhängig vom Generator der Klasse '8' entwickelt werden. Es könnten sogar unterschiedliche Techniken wie Template-Matching oder Konturvergleich mittels Hausdorffmetrik [Fuchs Bringmann Oganowski 1999] zum Einsatz kommen.
Bisher wurde davon ausgegangen, daß alle alternativen Objekte aus genau den selben Komponentenobjekten, nämlich der Menge *A*, aggregiert werden. In der Praxis ist dies jedoch nicht oft der Fall. Wenn beispielsweise in einer gerissenen Linie ein Buchstabe 'i' erkannt wird, dann benutzt sein Generator nur eine Teilmenge der Objekte, die zur Erzeugung der Linie genutzt werden. Der Schwellwertklassifikator ist im Gegensatz zum Bayes-optimalen Klassifikator auch in diesem Fall definiert. Die weitaus meisten realen Situation bei der Erkennung von technischen Zeichnungen sind so geartet, daß ein Bayes-optimaler Klassifikator nicht definiert ist.



Die Objektgeneratoren der Objekte *a* und *b* haben unterschiedliche Definitionsbereiche. *a* und *b* sind dennoch (wegen der gemeinsamen Verwendung des Objektes *c*) unvereinbar. Das heißt, es darf letztendlich nur entweder *a* oder *b* existieren. Eine Entscheidung und damit ein Klassifikator ist erforderlich.

Abb. 3-23 Minimale Konfiguration von Objekten in der zwar der Schwellwert- nicht aber der Bayes-optimale Klassifikator definiert ist.

3.2.2.2 Definition Modell einer Domäne

Zusammenfassend wird das Modell einer Domäne, einer Sorte technischer Zeichnungen definiert, indem das statische Modell um die domänenspezifische Dynamik und die Schwellwerte erweitert wird.

Definition 20. *Das Modell **M** einer Domäne besteht aus dem statischen Modell M_{stat} und den klassenspezifischen Varianten der Metaklassen sowie den Schwellwerten für die Schwellwertklassifikator:*

$$M := (M_{stat}, Metaklassen_C, b_C). \quad \textbf{(3-56)}$$

Die Forderungen (**II**) und (**IV**) *müssen für alle Klassen erfüllt sein.*

3.2.3 Algorithmus Objektgenerierung

Nach der Modelldefinition sind alle Voraussetzungen für die Angabe des Algorithmus zur Generierung aller Objekte einer Vorlage erfüllt. Dieser ruft seinerseits den Algorithmus *metaklasse* der jeweiligen Klasse auf.

Algorithmus 7 Objektgenerierung.

In: Objektmenge $O = \emptyset$	**Out:** Objektmenge $O \neq \emptyset$
Bestimme eine beliebige zu causes kompatible Reihenfolge der Klassen $(c_1, c_2 \ldots c_{\lvert C\rvert})$, *d.h. es gilt* $\forall i \in \{2 \ldots \lvert C \rvert\} : (c_i, c_{i-1}) \notin \overline{causes}$ *For* $i = 0 \ldots \lvert C \rvert$ $\quad metaklasse_{c_i}$	**(3-57)**

Das Erzeugen aller prinzipiell möglichen Objekte scheint auf den ersten Blick ineffizient, weil Objekte mit einer geringen Wahrscheinlichkeit genauso entwickelt werden, wie die mit einer

hohen Bewertung. Allerdings geht in die Bewertung durch den Schwellwertklassifikator stets nur ein sehr lokaler Kontext ein, so daß diese Wertung nur als erster Anhaltspunkt für eine endgültige Entscheidung verstanden werden kann. Auch ein lokal betrachtet unwahrscheinliches Objekt kann sich letztendlich als korrekt erweisen (vgl. Beispiel „B-8“).
Man könnte sich eine temporäre Ausweitung des Kontextes vorstellen, um die Sicherheit der Bewertung zu erhöhen. Da ein solcher Kontext jedoch nur durch Klassen einer höheren Komplexität beschrieben werden kann, bedeutet eine solche Kontexterweiterung die temporäre Generierung von Objekten der komplexeren Klassen. Wenn ein Objekt temporär generiert wird, kann es auch gleich dauerhaft im Sinne einer Objekthypothese erzeugt werden. Genau dies leistet aber der Algorithmus „Objektgenerierung“ bereits ohnehin. Einen prinzipiellen Ausweg stellen Top-Down-Systeme dar, die aus in 3.2.1 genannten Gründen hier nicht betrachtet werden.
Da durch den Algorithmus „Objektgenerierung“ alle Objekte erzeugt werden, die entsprechend des Schwellwertklassifikators und der jeweils ausgewählten Metaklasse möglich sind, werden auch Objekte angelegt, die paarweise unvereinbar sind. Diese verletzen das Entwurfparadigma. Das Aufdecken dieser Widersprüche ist Gegenstand des nun folgenden Abschnittes.

3.2.4 Konfliktgenerierung

Ein Grundanliegen des Modellierungskonzeptes von Y ist die nur lokal abhängige Modellierbarkeit der Klassen. Deshalb muß aus zwei Modellen nicht unmittelbar hervorgehen, ob Instanzen dieser Modelle in Konkurrenz zueinander stehen können oder nicht. Ein gemeinsamer Definitionsbereich *X* konkurrierender Klassen wie in Abb. 3-22 kann im Allgemeinen nicht vorausgesetzt werden. Ein Beispiel kann dieses Problem verdeutlichen.

> **Beispiel Gerissene Linie.** In einer Domäne werden *Pixel* als einzige primitive Klasse angenommen. Aus diesen Bildelementen werden mittels Templatematchings *Schriftzeichen* gebildet. *Striche* werden durch Skelettierung [Kovalevski 1990] und anschließende Vektorisierung aus den *Pixel*n generiert. Die genannten Bildverarbeitungprozesse sind in den Objektgeneratoren der Klassen zu implementieren. Aus langen und kurzen *Linie*n werden *gerisseneLinie*n aggregiert.

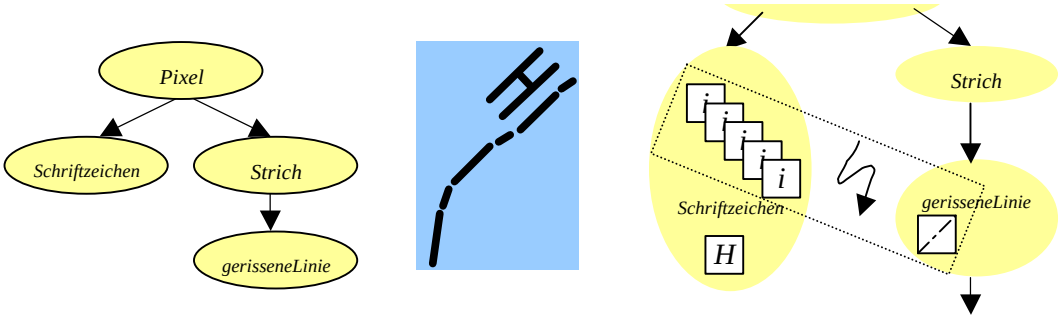


Abb. 3-24 Klassen, eine mögliche Vorlage und einige unvereinbare Objekte

> Nun muß ein erkennendes System die Unvereinbarkeit zwischen den Objekten ‚*i*‘ und der *gerissenen Linie* feststellen. Dies muß auch dann geschehen, wenn ein menschlicher Modellierer dieses Problem nicht vorhergesehen hat.

Das Entwurfsparadigma eröffnet eine Möglichkeit, trotz verschiedener Definitionsbereiche von Objekten Unvereinbarkeiten zwischen Objekten zu erkennen.
Für den Mechanismus der Konfliktgenerierung wird eine ***causes*** Relation über Objekten benötigt. Diese gestaltet sich so, daß alle Objekte *o*, die ein anderes Objekt *o*' entsprechend seines Objektgenerators bedingen, in der Relation *o* ***causes*** *o*' stehen.

Definition 21. *Die Relation* ***causes*** *ist eine Teilmenge von* $O \times O$.

$$causes := \{(o,o') \mid o' = g_{c_{o'}}(...o...)\} \quad \textbf{(3-58)}$$

Da ein Objekt nur existieren kann, wenn auch alle seine Komponenten vorliegen, folgt aus der *causes*-Relation folgende stochastische Aussage für alle Objekte *o* und *o'*:

$$o \; causes \; o' \Rightarrow P(o \mid o') = 1 \quad \textbf{(3-59)}$$

Wenn ein Objekt *o* ein echter Teil eines anderen Objektes *o'* ist, so steht *o* nicht nur in der Relation *causes* zu *o'* sondern darüber hinaus in einer Relation ***part_of***. Die Definition dieser Relation beruht auf dem Vektor $parts_{c'}$ der Klasse *c'* von *o'*.

Definition 22. *Die Relation **part_of** ist eine Teilmenge von O×O.*

$$part\_of := \{(o,o') \mid o' = g_{c_{o'}}(...o_i...) \wedge (parts_{c_{o'}})_i = 1\} \quad \textbf{(3-60)}$$

part_of ist also eine Teilmenge von *causes* über Objekten. Diese beiden Relationen sind durch den Prozeß der Objektgenerierung direkt gegeben.
In allen weiteren Betrachtungen wird die Objektgenerierung als abgeschlossen betrachtet. Alle potentiellen Objekte seien generiert worden und nur noch die Relationen über diesen Objekten werden analysiert. Von der Klassenzugehörigkeit der Objekte wird im Weiteren abstrahiert. Sie ist nach der Aggregation der Objekte in keiner Weise mehr von Belang.
Aus dem Entwurfsparadigma folgen unmittelbar zwei Regeln, die eine symmetrische, binäre Relation über Objekten definieren. Diese Relation gibt an, welche Objekte nicht gleichzeitig zu einer gültigen **Interpretation** einer Vorlage gehören dürfen.

Erzeugungsregel. *Ist ein Objekt Teil zweier Objekte, so sind diese beiden Objekte unvereinbar.*

Vererbungsregel. *Sind zwei Objekte unvereinbar und begründet eines dieser Objekte ein drittes Objekt, so überträgt sich die Unvereinbarkeit auf dieses Objekt.*

Die formale, rekursive Definition der *conflict*-Relation, die diesen Regeln folgt, lautet:

Definition 23. *Die Relation **conflict** ist eine Teilmenge von O×O. Es gilt:*

$$conflict := \left\{ (o,o') \middle| \begin{array}{l} (\exists o'' : o''\ part\_of\ o \wedge o''\ part\_of\ o' \wedge o \neq o') \vee \\ (\exists o'' : o''\ causes\ o \wedge o''\ conflict\ o') \end{array} \right\} . \quad \textbf{(3-61)}$$

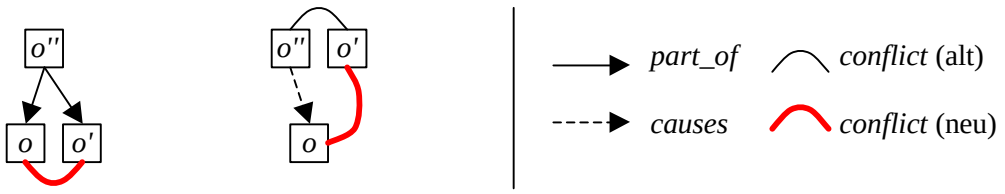


Abb. 3-25 Erzeugung und Vererbung von Konflikten

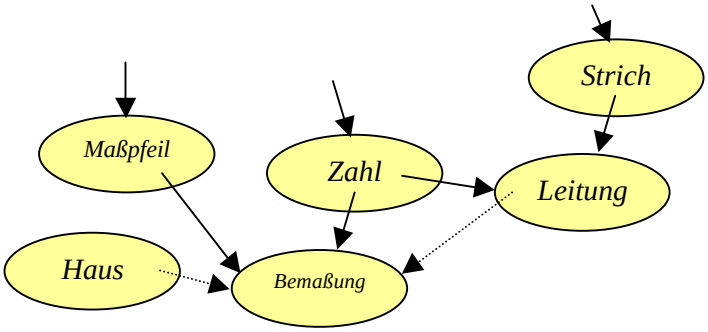
Die *conflict*-Relation impliziert wegen des Entwurfsparadigmas die stochastische Unvereinbarkeit:

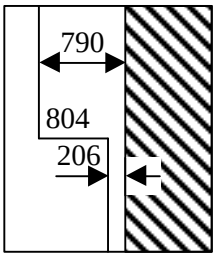
$$o \; conflict \; o' \Rightarrow P(o,o') = 0 \quad \textbf{(3-62)}$$

gilt für alle Objekte *o* und *o'*.

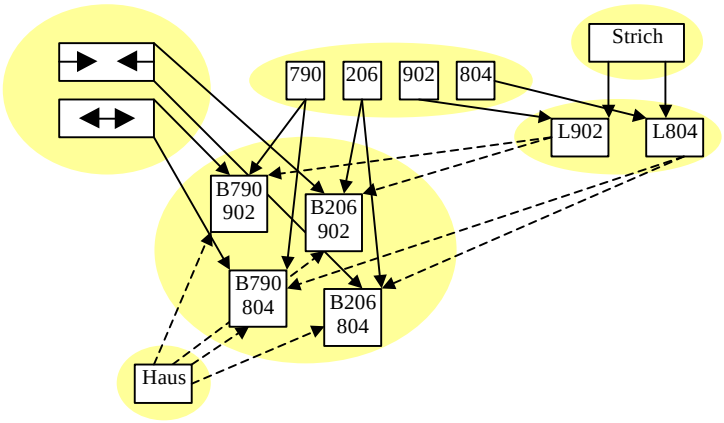
An einem komplexen Beispiel, eine Zusammenfassung der Beispiele „Entwurfsparadigma“ und „Leitung“, kann die Plausibilität der Definition von *conflict* geprüft werden.

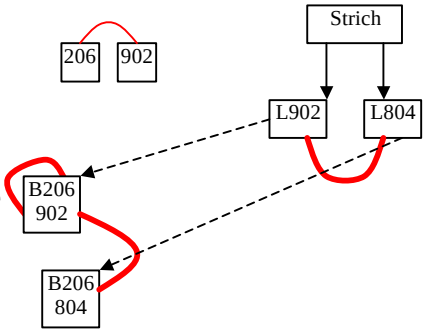
Beispiel *conflict*-Relation. Ein Ausschnitt aus einem Leitungsplan könnte die folgende Struktur aufweisen: Es wurden u. a. die Bestandteile einer Klasse *Bemaßung* modelliert. Die Klasse *Zahl* geht einerseits als Leitungsdurchmesser in die Klasse *Leitung* und andererseits als Maßzahl in *Bemaßung*sobjekte ein. Die Zeichenkette „206“ ist auch kopfstehend, dann als Leitungsdurchmesser 902, interpretierbar. *Maßpfeil*e seien als Generalisierung von inneren und äußeren Maßpfeilen modelliert worden. Die Maßzahl stehe immer über dem *Maßpfeil*.





Auf der Ebene der Objekte ergibt sich der nebenstehende Graph. Unten sind einige Konflikte dargestellt, die sich aufgrund der Erzeugungs- und der Vererbungsregel ableiten lassen. Der aus früheren Aggregationen stammende Konflikt zwischen den Zahlen 206 und 902 ist etwas dünner dargestellt. Der Konflikt zwischen den *Leitung*en L902 und L804 beruht auf der überschneidenden Verwendung des Striches (Erzeugungsregel). Der Konflikt zwischen den *Bemaßung*sobjekten B206/902 und B206/804 kann dann nach zweimaliger Anwendung der Vererbungsregel begründet werden. Allerdings braucht das Objekt B206/902 eigentlich gar nicht mehr in die Betrachtungen eingeschlossen werden, denn es steht zu sich selbst in Konflikt. Es kann also nicht existieren! Die Unvereinbarkeit der Zahlen 206 und 902 ist auch für diesen Selbstkonflikt die indirekte Ursache.





Dagegen sind die *Bemaßung*en B790/804 und B206/804 durchaus vereinbar, obwohl für beide Objekte ein und das selbe *Leitung*sobjekt L804 eine Bedingung darstellt.
Würde man auf die Unterscheidung zwischen den Konzepten *part_of* und *causes* verzichten, wäre theoretisch eine vollständige Modellierung von technischen Zeichnungen möglich. Dann müßte man jedoch alle *Bemaßung*en, *Leitung*en und *Häuse*r der Vorlage zu einem einzigen *Leitungs_Bemaßungs_Haus_System* aggregieren, wenn diese alle (indirekt) miteinander verbunden sind und man garantieren will, daß nur *Bemaßung*en, die tatsächlich etwas bemaßen, zulässig sind.

Der Zusammenhang zwischen Entwurfsparadigma, Modellierung, Objekterzeugung, Konflikten und Konfliktregeln kann anschaulich so beschrieben werden:

Die Konfliktregeln sind die formale Konsequenz des Entwurfsparadigmas.
Das Paradigma beschreibt, in welchem *causes/parts* Verhältnis die Klassen zueinander stehen dürfen. Der Modellierende muß beim Design der Klassen und ihrer Relationen auf die Einhaltung des Paradigmas achten. Die Objekterzeugung weiß von alledem nichts. Sie generiert einfach lokal-plausible Objekte. Die Konfliktregeln und die daraus erwachsenen Konflikte protokollieren die dabei auftretenden Verstöße gegen das Paradigma. Später werden die Konflikte der Definition von gültigen **Interpretationen** einer Zeichnung dienen und gemäß eines **globalen Optimalitätskriteriums** gelöst werden.

Konflikte zwischen Objekten der primitiven Klassen müssen durch Aufzählung vorgegeben werden. Sind beispielsweise die *Schriftzeichen* eines Planes als primitive Klasse modelliert worden, dann müssen auch die Konflikte zwischen alternativen Objekten dieser Klasse dem System Y mitgeteilt werden. Dies geschieht nachdem alle primitiven Objekte, das heißt alle prinzipiell möglichen *Schriftzeichen*, an das System übergeben worden sind.

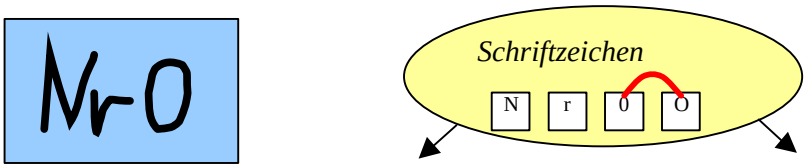


Abb. 3-26 Primitive Objekte und Konflikte einer Zeichnung

Die Konflikte zwischen Objekten sind im Gegensatz zur *part_of*- und *causes*-Beziehung nicht offensichtlich. Der folgende Abschnitt beantwortet die Frage, wie die relevanten Konflikte effektiv aufgedeckt werden können.
Aus der Definition der *conflict*-Relation folgt, daß sehr viele Objekte zueinander in Konflikt stehen können. Bedeutsam sind aber nur die Konflikte zwischen Objekten der Ausgabeschnittstelle C_{out}.
Für eine effektive, automatische Erzeugung der *conflict*-Relation aus den explizit bekannten *causes*- und *part_of*-Relationen zwischen den Objekten, benötigt man den Begriff ***path*** zwischen zwei Objekten und einen darauf basierenden Satz.

Definition 24. *Ein **path** ist eine binäre Relation über Objekten. Zwei Objekte o_1 und o_n sind durch einen path verbunden, wenn es ist eine Folge von Objekten $(o_1,o_2,...o_n)$ gibt, wobei für alle Nachbarn o_i und o_{i+1} in dieser Folge gilt: o_i causes o_{i+1}. Sind die beiden Objekte o_1 und o_n identisch, so sind auch sie durch einen path verbunden.*

Ein Objekt ist also mit sich selbst stets durch einen *path* verbunden.

Definition 25. ***conflict*$_{path}$** *ist eine binäre Relation über Objekten. Zwei Objekte o_m und o_n stehen in dieser Relation, genau wenn es ein weiteres Objekt o gibt, das durch path$(o,o_1,...o_m)$ und path$(o,o_2,...o_n)$ mit o_m und o_n verbunden ist und ferner o part_of o_1, o part_of o_2* und *o_1 ¹ o_2 gilt.*

Satz 2. *Zwei Objekte o und o' stehen in Konflikt zueinander, genau wenn o conflict$_{path}$ o‘ gilt oder wenn es zwei Objekte o'' und o''' gibt mit: o'' path o und o''' path o' und o'' conflict o'''.*

Beweis. Die formale Fassung des Satzes lautet:

$$o \text{ conflict } o' \Leftrightarrow \left(\begin{array}{l} \left(o \text{ conflict}_{path} \; o'\right) \vee \\ \left(\exists o'' : \exists o''' : o'' \text{ path } o \; \wedge \; o''' \text{ path } o' \; \wedge \; o'' \text{ conflict } o'''\right) \end{array} \right). \qquad \textbf{(3-63)}$$

Zunächst wird gezeigt, daß folgendes gilt:

$$o \; conflict \; o' \Leftarrow \left(o \; conflict_{path} \; o'\right). \qquad \textbf{(3-64)}$$

Es gibt also ein Objekt *o''* mit *o'' path o* und *o'' path o'*. Daraus folgt, daß es eine Folge von Objekten p_1 = (*o''*,*a*...*o*) gibt. Aus *o'' path o'* folgt, daß es eine Folge p_2 = (*o''*,*b*...*o'*) gibt.
Wegen *o'' part_of a*, *o'' part_of b* und aus dem ersten Teil der Definition der *conflict*-Relation folgt, *a conflict b*. Nach iterativer Anwendung des zweiten Teils der *conflict*-Definition, sieht man, daß auch *o* und *o'* zueinander in Konflikt stehen.

Weiterhin ist zu zeigen, daß auch der folgende Schluß richtig ist:

$$o \; conflict \; o' \Leftarrow \left(\exists o'' : \exists o''' : o'' \; path \; o \; \wedge \; o''' \; path \; o' \; \wedge \; o'' \; conflict \; o'''\right). \qquad \textbf{(3-65)}$$

Der rekursive zweite Teil der *conflict*-Definition führt unmittelbar zu dieser Aussage auch wenn *o''* und *o* oder *o'''* und *o'* identisch sind.

Der Rückschluß

$$o \; conflict \; o' \Rightarrow \left(\begin{array}{l}\left(o \; conflict_{path} \; o'\right) \vee \\ \left(\exists o'' : \exists o''' : o'' \; path \; o \; \wedge \; o''' \; path \; o' \; \wedge \; o'' \; conflict \; o'''\right)\end{array}\right) \qquad \textbf{(3-66)}$$

wird indirekt vollzogen. Angenommen wird, daß es eine andere Art der Konflikterzeugung als (3-64) oder (3-65) gibt. Dazu wird (3-66) negiert:

$$o \; conflict \; o' \wedge \left(\begin{array}{l}\neg\left(o \; conflict_{path} \; o'\right) \wedge \\ \left(\forall o'' : \forall o''' : \neg o'' \; path \; o \; \vee \; \neg o''' \; path \; o' \; \vee \; \neg o'' \; conflict \; o'''\right)\end{array}\right). \qquad \textbf{(3-67)}$$

Im zweiten Teil der Annahme

$$o \; conflict \; o' \wedge \left(\forall o'' : \forall o''' : \neg o'' \; path \; o \; \vee \; \neg o''' \; path \; o' \; \vee \; \neg o'' \; conflict \; o'''\right) \qquad \textbf{(3-68)}$$

gelangt man sofort zum Widerspruch. Jede der drei Teilaussagen in der Disjunktion liefert einen Wahrheitswert *False*, wenn man für *o''* das Objekt *o* und für *o'''* das Objekt *o'* einsetzt. Ein solches Einsetzen müßte aufgrund der Allquantoren jedoch möglich sein.
■

Die folgende Abbildung illustriert diesen Satz in seiner Anwendung für die Ausgabeobjekte O_{out}.

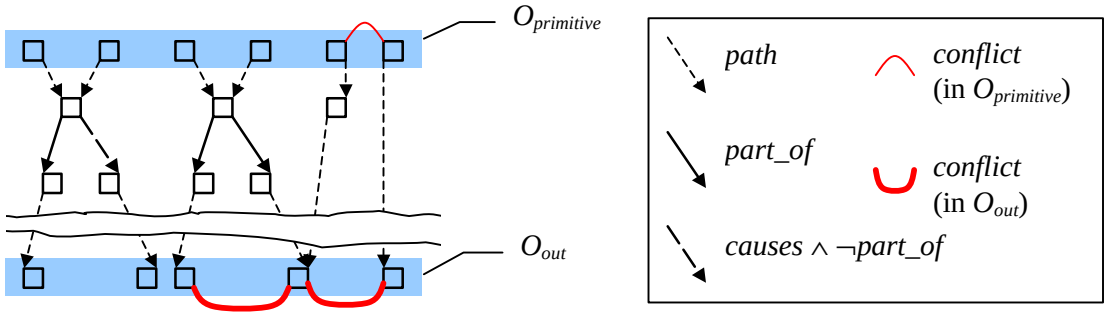


Abb. 3-27 Relevante Konflikte und *path*es für die Objekte der Ausgabeschnittstelle

Satz 2 stellt eine alternative Definition der *conflict*-Relation dar. Alle Konflikte in den Ausgabeobjekten O_{out} können direkt auf ihre Ursachen in der *causes*-Struktur zurückgeführt werden. Gäbe es keine Konflikte bei den primitiven Objekten, dann ließen sich alle Konflikte auf ein gemeinsam verwendetes Objekt zurückführen. Der Satz 2 würde dann lauten:

$$o \; conflict \; o' \Leftrightarrow \left(o \; conflict_{path} \; o'\right). \quad \textbf{(3-69)}$$

Der Algorithmus für die automatische Generierung der Konflikte innerhalb der Menge der Ausgabeobjekte $conflict_{out}$ kann nun angegeben werden. Die formale Spezifikation ist:

$$Konfliktgenerierung := (conflict_{out} := \{(o,o') \mid o \; conflict \; o' \wedge o,o' \in O_{out}\}) \quad \textbf{(3-70)}$$

Es wird ein Datentyp *ConflictObjects* benötigt. Dieser Typ sei als Menge von Mengen von Verweisen auf Objekte implementiert:

$$ConflictObjects := 2^O. \quad \textbf{(3-71)}$$

Neben den Mengenoperationen, seien zwei Funktionen über *ConflictObjects* definiert. Die erste Funktion

$$flatten : ConflictObjects \rightarrow ConflictObjects \quad \textbf{(3-72)}$$

vereinigt alle inneren Mengen zu einer einzigen:

$$flatten(co) := \left\{ \bigcup_{a \in co} a \right\}. \quad \textbf{(3-73)}$$

Beispielsweise liefert der Ausdruck $flatten(\{\{o_1,o_2,o_3\},\{o_4\},\{o_5,o_6\}\})$ die „geglättete“ Menge $\{\{o_1,o_2,o_3,o_4,o_5,o_6\}\}$.
Eine weitere Funktion

$$makeConflicts : ConflictObjects \rightarrow O \times O \quad \textbf{(3-74)}$$

erzeugt eine symmetrische Relation über den Objekten, die – wie man später sehen wird – einen Teil der $conflict_{out}$-Relation bildet:

$$makeConflicts(co) := \bigcup_{(a,b) \in co \times co} \{(a \times b),(b \times a)\} \quad \textbf{(3-75)}$$

Der Ausdruck $makeConflicts(\{\{o_1,o_2,o_3\},\{o_4\},\{o_5,o_6\}\})$ erzeugt die Menge der Objektpaare
$\{(o_1,o_4),(o_1,o_5),(o_1,o_6),(o_2,o_4),(o_2,o_5),(o_2,o_6),(o_3,o_4),(o_3,o_5),(o_3,o_6),(o_4,o_5),(o_4,o_6)\}$
und die dazu gespiegelten geordneten Paare $\{(o_4,o_1),(o_5,o_1)...\}$.

Es wird jedem Objekt *o* ein Datenfeld *o.co* vom Typ *ConflictObjects* zugeordnet.
Die Mengen *co* aller Objekte seien beim Starten des Algorithmus leer. Die Menge der Objekte, die im Sinne von Satz 2 einen Konflikt auslösen, sei mit O_{multi} bezeichnet. Alle Objekte, die Teil von mindestens zwei anderen Objekten sind, gehören zu dieser Menge.

$$O_{multi} := \{o \mid \exists o' : \exists o'' : o' \neq o'' \wedge o \; part\_of \; o' \wedge o \; part\_of \; o''\} \quad \textbf{(3-76)}$$

Die Menge O_{multi} kann während der Objektgenerierung sofort ermittelt werden.

Die Konfliktgenerierung erfolgt in zwei Schritten. Zunächst werden alle Konflikte erzeugt, die direkt aus den bereits existierenden Konflikten resultieren. Anschließend werden alle weiteren Konflikte hinzugefügt.

Algorithmus 8 Konfliktgenerierung.

In: Relation *conflict* in $O_{primitive}$, Objekte O_{multi}, Klassen C_{out}	**Out:** Relation $conflict_{out}$
$conflict_{out} := \varnothing$ $primitveKonflikte()$ $\forall o \in O_{multi}$ $\quad down(o)$	**(3-77)**

Dieser Rahmenalgorithmus benutzt die globale Variable $conflict_{out}$. Er ruft zunächst die Funktion *primitiveKonflikte* auf. Diese erzeugt alle Konflikte in der Ausgabemenge O_{out}, welche aufgrund der bereits existierenden Konflikte entstehen. Diese Konflikte waren durch Aufzählung vorgegeben worden. (vgl. 2.3 Vorverarbeitung und Vektorisierung).

Algorithmus 9 primitiveKonflikte.

In: Relation *conflict* in $O_{primitive}$, Klassen C_{out}	**In/Out:** Teilmenge der Relation $conflict_{out}$
$\forall (o, o') \in conflict$ $\quad conflict_{out} := conflict_{out} \cup makeConflicts(down(o), down(o'))$	**(3-78)**

Den Kern der Konfliktgenerierung bildet die rekursive Funktion *down*(*o*). Sie liefert alle Ausgabeobjekte, mit denen *o* durch einen *path* verbunden ist. Sollte *o* ebenfalls ein Ausgabeobjekt sein, dann wird also auch *o* zurückgegeben. Weiterhin werden in dieser Funktion die neuen Konflikte erzeugt. Immer wenn *o* durch einen $path_{part\_of}$ mit zwei Objekten der Ausgabeklassen verbunden ist, wird ein Konflikt zwischen diesen der Menge der Konflikte hinzugefügt.
Der Algorithmus *down* benutzt zwei komplementäre Funktionen $child_{part\_of}$ und $child_{not\_part\_of}$. Diese sind Funktionen eines Objektes und liefern dessen direkte Nachfolger:

$$child_{part\_of}(o) := \{o' \mid o \ part\_of \ o'\}$$
$$child_{not\_part\_of}(o) := \{o' \mid o \ causes \ o' \wedge \neg o \ part\_of \ o'\} \qquad \textbf{(3-79)}$$

Algorithmus 10 down.

In: Objekt o, Klassen C_{out}	**Out:** *ConflictObjekts co*	**In/OUT:** $conflict_{out}$
$if\ (o.co \neq \varnothing)$	1	
$\quad return\ co$	2	
$\quad stop$	3	
$\forall o' \in childs_{part\_of}(o)$	4	
$\quad o.co := o.co \cup down(o')$	5	
$conflicts_{out} := conflicts_{out} \cup makeConflicts(co)$	6	**(3-80)**
$\forall o' \in childs_{not\_part\_of}(o)$	7	
$\quad o.co := o.co \cup down(o')$	8	
$if\ (o \in O_{out})$	9	
$\quad o.co := o.co \cup \{\{o\}\}$	10	
$co := flatten(co)$	11	
$return\ co$	12	

Zur Diskussion des Algorithmus wird ein Beispiel betrachtet:

Beispiel Konfliktgenerierung. In der Abbildung ist eine *causes*-Struktur dargestellt. Die Konflikte zwischen primitiven Objekten und die Menge O_{multi} sind bereits gegeben. Die Konflikte innerhalb O_{out} sind zu bestimmen.

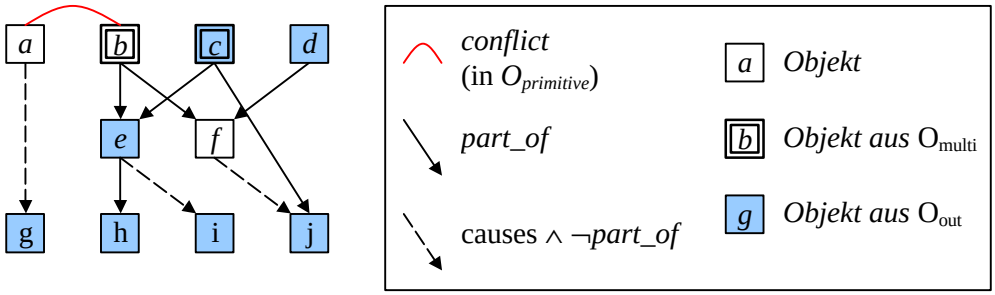


Abb. 3-28 Objektgraph mit einem vorgegeben Konflikt

Der Aufruf der Funktionen geschieht in der folgenden Reihenfolge (Die Aufrufe von *makeConflicts*() werden nur in die Auflistung eingetragen, wenn als Argument eine nichtleere Menge auftritt. Andernfalls werden sowieso keine Konflikte generiert):

Konfliktgenerierung()
  primitiveKonflikte()
    down(*a*)→{{*g*}}
      down(*g*) →{{*g*}}
    down(*b*) →{{*h, i, e, j*}}
      down(*e*) →{{*h, i, e*}}
        down(*h*) →{{*h*}}
        down(*i*) →{{*i*}}
      down(*f*) →{{*j*}}
        down(*j*) →{{*j*}}
      makeConflicts({{*h, i, e*}, {*j*}})→{(*h, j*), (*i, j*), (*e, j*)}
    makeConflicts({{*g*}, { *h, i, e, j*}})→{(*g, h*), (*g, i*), (*g, e*), (*g, j*)}
  down(*b*)
  down(*c*)
    down(*e*) →{{*h, i, e*}}
    down(*j*) →{{*j*}}
    makeConflicts({{*h, i, e*}, {*j*}})→{(*h, j*), (*i, j*), (*e, j*)}

Es werden insgesamt die folgenden Konflikte in C_{out} erzeugt:
(g,h),(g,i),(g,j),(g,e),(h,j),(i,j),(e,j).

Einige Eigenschaften des Algorithmus lassen sich an diesem Beispiel beobachten:

- O_{out} und $O_{primitive}$ und demzufolge C_{out} und $C_{primitive}$ müssen nicht disjunkt sein. Die Objekte *c* und *d* gehören zu beiden Objektmengen.
- Ein Konflikt mit mehreren, verschiedenen Ursachen wird mehrfach erzeugt. So erfolgt der Aufruf *makeConflicts*({{*h, i, e*}, {*j*}}) im Beispiel gleich zweimal: das erstemal bei der Erzeugung aller Konflikte, die aus *a conflicts b* resultieren und noch einmal innerhalb des Aufrufes von *down*(*c*). Dagegen führt der Aufruf von *down*(*b*) zu keinem Mehraufwand, weil *b* bereits in *primitiveKonflikte* betrachtet worden war.
- Nicht alle Objekte von O_{out} werden angesteuert. So z. B. muß Objekt *d* nie in einem Aufruf *down*(*d*) auftreten.
- Nichtzusammenhängende (bezüglich der Relation *conflict* ∪ *causes*) Teilgraphen werden isoliert betrachtet.
- Eine leichte Optimierungsmöglichkeit ergibt sich wie folgt:
  Wenn ein Objekt *o* keine Pfade *path* zu Objekten von O_{out} besitzt und auch selbst nicht Element von O_{out} ist, dann wird bei jedem Aufruf *down*(*o*) ein rekursiver Abstieg vorgenommen. Das geschieht weil die Abfrage *if*(*o.co* ≠ ∅)... nicht hinreichend für die Feststellung ist, daß *o* bereits angesteuert worden war. Eine zusätzliche boolesche Variable *visited* kann leicht Abhilfe schaffen.

Beweis. Die Korrektheit des Algorithmus kann sofort gezeigt werden, wenn man zunächst davon ausgeht, daß *down* korrekt ist. Dann erzeugt *down*(*o*) alle Konflikte in O_{out}, die (indirekt) durch *o* verursacht werden und liefert alle Objekte aus O_{out}, mit denen *o* durch einen *path* verbunden ist. Formal gilt also folgende Spezifikation der Funktion *down*:

$$down(o) \coloneqq (o.co \coloneqq \{o' \mid o \ path \ o' \wedge o' \in O_{out}\}, \\ \{(o', o'') \mid o' \ conflict \ o'' \wedge o', o'' \in o.co\}). \qquad \textbf{(3-81)}$$

Als Funktionswert wird nur der erste Teil des Paares zurückgegeben. Die zweite Menge wird als Nebeneffekt des Funktionsaufrufes der Menge $conflict_{out}$ hinzugefügt.

Unter diesen Voraussetzungen wird nun der Algorithmus *Konfliktgenerierung* analysiert: Es gibt nach Satz 2 zwei Ursachenklassen für die Existenz von Konflikten. Diese werden im Algorithmus nacheinander abgehandelt:

(1) Für alle bereits vorliegenden Konflikte (*o''*,*o'''*) wird in *primitiveKonflikte* mit *down*(*o''*)×*down*(*o'''*) sowie *down*(*o'''*)×*down*(*o''*) die Menge aller Konflikte erzeugt, die direkt aus (*o''*,*o'''*) folgen. Das resultiert aus (3-65):

$$o \; conflict \; o' \Leftarrow \left(\exists o'' : \exists o''' : o'' \; path \; o \; \wedge \; o''' \; path \; o' \; \wedge \; o'' \; conflict \; o'''\right).$$

(2) Die übrigen Konflikte werden aus den Elementen *o*" von O_{multi} nach (3-64) erzeugt, indem für alle *o*" ∈ O_{multi} in *Konfliktgenerierung* die Funktion *down*(*o*") aufgerufen wird.

$$o \; conflict \; o' \Leftarrow \left(o \; conflict_{path} \; o'\right).$$

Nach Satz 2 gibt es keine weiteren Möglichkeiten der Konfliktgenerierung.

Nun muß gezeigt werden, daß *down*(*o*) das Erforderliche leistet. Die Funktion wird nur dann sofort verlassen, wenn sie schon einmal für *o* aufgerufen wurde (*o.co* ≠ ∅). Induktiv kann gezeigt werden, daß *down* auch andernfalls korrekt arbeitet.

Induktionsanfang: Wenn *down*(*o*) für ein Objekt ohne Nachfolger ($child_x(o) = \emptyset$) aufgerufen wird, werden keine Konflikte erzeugt. Es gibt dann zwei Fälle für den Rückgabewert:

$$child_x(o) = \emptyset \;\Rightarrow\; down(o) = \begin{cases} \{\{o\}\} & o \in O_{out} \\ \emptyset & o \notin O_{out} \end{cases} \qquad \textbf{(3-82)}$$

Wenn *down*(*o*) für ein Objekt aufgerufen wird, welches Nachfolger besitzt, die aber selbst keine Nachfolger besitzen, enthält *o.co* = {{*o'*},{*o''*}...} nach Zeile 5 alle Ausgabeobjekte, jeweils in einelementigen Mengen, die paarweise in Konflikt zueinander stehen. Diese Konflikte werden dann in der Funktion *makeConflict*(*o.co*) erzeugt und *conflict* hinzugefügt. In Zeile 8 werden die übrigen Objekte *o'''*, für die *o causes o'''* gilt *o.co* hinzugefügt. Gegebenenfalls wird auch *o* selbst dieser Menge in Zeile 10 zugeordnet. *o.co* enthält somit alle Ausgabeobjekte zu denen ein *path* von *o* existiert. Durch Zeile 11 wird die Gruppierung der Objekte aufgehoben:

o.co := {{*o'*,*o''*...*o''*...*o*}} = *flatten*({{*o'*},{*o''*}...{*o'''*}...{*o*}).

Induktionsschritt: Angenommen *down*(*o*) erfüllt die Spezifikation (3-81) für alle ihre internen *down*-Aufrufe. Dann ist zu zeigen, daß sie auch selbst den Anforderungen genügt. Der erste Teil des geordneten Paares dieser Spezifikation, der Funktionsrückgabewert ist trivialerweise gewährleistet, weil die Vereinigung aller Rückgabewerte der internen *down*-Aufrufe (gegebenenfalls um *o* selbst erweitert) zurückgegeben wird. Für den zweiten Teil, den Beitrag von *down*(*o*) zu $conflict_{out}$, betrachtet man die Mengen *down*(*o'*) für jedes *o'*∈ $child_{part\_of}$ (*o*). Für jede Paarung ({...*o''*...}, {...*o'''*...}) dieser Mengen werden in *makeConflicts* alle Paare (*o''*,*o'''*) und (*o'''*,*o''*) erzeugt. Das sind genau die Konflikte, die nach (3-64) verursacht werden:

$$o'' \; conflict \; o''' \Leftarrow \left(o'' \; conflict_{path} \; o'''\right).$$

Diese Paare werden $conflict_{out}$ hinzugefügt. *down*(*o*) erzeugt also das spezifizierte Ergebnis.

Der gesamte Algorithmus *Konfliktgenerierung* terminiert, wenn die rekursive Funktion *down* stets terminiert. Aus der Zyklenfreiheit der Klassen bezüglich der Relation *causes* über Klassen folgt die Zyklenfreiheit der Objekte bezüglich der Objektrelation *causes*. Da *down* innerhalb einer Funktion *down*(*o*) nur für Objekte *o'* mit *o causes o'* aufgerufen wird, terminiert *down* immer.

■

3.3 Zusammenfassung

In diesem Kapitel wurde gezeigt, wie technische Zeichnungen modelliert werden können.
Im Umfeld der Begriffe Klasse und Objekt wurden folgende Konzepte formalisiert:

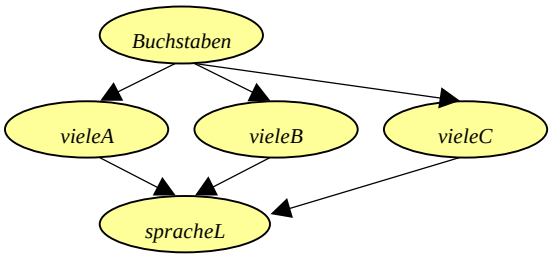
- **Stochastische Abhängigkeiten** werden mittels bedingter Wahrscheinlichkeiten und der *causes* Relation direkt modelliert.
- Die **Unvereinbarkeit** von Objekten wird durch die *conflict* Relation repräsentiert. Dabei werden auch Unvereinbarkeiten erkennbar, die der menschliche Modellierer nicht vorhergesehen hat (vgl. Beispiel „gerissene Linie“). Die Unvereinbarkeit wird nicht direkt modelliert, sondern ergibt sich automatisch aus den Konfliktregeln, die wiederum die Einhaltung des Entwurfsparadigmas voraussetzten.

Die Modellierung einer Sorte technischer Zeichnungen kann mit dem hier vorgeschlagenen Ansatz entsprechend der Zielsetzungen aus Abschnitt 0 erfolgen. Sie ist

- **modular** (Die einzelnen Klassen haben frei typisierbare Schnittstellen, die nur aus der logischen *causes*-Struktur resultieren.),
- **deklarativ** (Die Dynamik und Reihenfolge der Objekterzeugung ist für den Modellierenden nicht wichtig.) und
- **scharf** (Die inhärente Unschärfe der Mustererkennung steckt in gut definier- und bestimmbaren Wahrscheinlichkeiten.).

Die Mächtigkeit des hier vorgeschlagenen Modellierungssystems muß sich in der Praxis erweisen. Erste Untersuchungen [Bringmann Buchroithner Fuchs Seifert 1997] haben gezeigt, daß sich sehr wohl auch komplizierte Domänen wie z. B. Gas- und Elektroleitungspläne modellieren lassen.
Als Indiz für die Reichhaltigkeit des Konzeptes diene folgendes Beispiel, in dem eine **nicht kontextfreie** Grammatik umgesetzt wird.

Beispiel nicht kontextfreie Grammatik. Die Sprache $L=\{a^n\ b^n\ c^n : n \geq 1\}$ wird von der folgenden (monotonen) Grammatik erzeugt: Terminalsymbole sind $\{a,b,c\}$, Nichtterminalsymbole sind $\{B,S\}$. Die Ableitungsregeln lauten $\{S \rightarrow aSBc,\ S \rightarrow abc,\ cB \rightarrow Bc,\ bB \rightarrow bb\}$. Die Sprache L ist nicht kontextfrei [Bronstein Semedjajew 1995].



Sie läßt sich jedoch durch das hier vorgestellte Modellierungskonzept sehr leicht nachempfinden. Man definiert drei Klassen *vieleA*, *vieleB* und *vieleC*, die mittels der *Metaklasse* „Iteration 1“ einzelne identische *Buchstabe*n ‚*a*‘,‚*b*‘ und ‚*c*‘ in sich zusammenfassen. Die Funktion *better*(.) wird so implementiert, daß ein falscher Buchstabe sofort zur Beendigung der Aggregation führt. Ein Merkmal jeder dieser Klassen sei die Anzahl *anz* der zusammengefaßten *Buchstabe*n. In einer weiteren Klasse *spracheL* mit der zugeordneten *Metaklasse* „Umgebung“ werden anschließend Objekte der drei Klassen zusammengefaßt. Das signifikante Merkmal *m* sei eine natürliche Zahl $\max(anz_{vieleA}, anz_{vieleB}, anz_{vieleC}) - \min(anz_{vieleA}, anz_{vieleB}, anz_{vieleC})$. Da ein deterministisches Problem vorliegt ist die Wahrscheinlichkeit eines Objektes *o* der Klasse *spracheL*:

$$P(o \mid m) := \begin{cases} wenn\ m = 0 & 1 \\ sonst & 0. \end{cases} \qquad \textbf{(3-83)}$$

Die Klasse *spracheL* enthält damit nur Zeichenketten, die Element der Sprache *L* sind.

Durch die Unterscheidung von *causes* und *part_of* wird eine wesentliche Verfeinerung der Ausdrucksmittel der Modellierung erreicht.

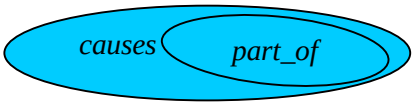


Abb. 3-29 Relationen zwischen Klassen. Durch den Vektor *parts* wird *causes* partitioniert.

Die freie Typisierung der Klassenparameter unterstützt die Umsetzung der hier vorgestellten Modellierungsmethodik in objektorientierten Programmiersprachen wie Java oder C++. Beispielsweise kann der Objektgenerator sehr intuitiv als Konstruktor der zugehörigen Klasse implementiert werden.

Es bleibt zu untersuchen, wie Zeichnungen aus so modellierten Domänen analysiert werden können. Man muß feststellen, ob mittels der bisher eingeführten Konzepte die Interpretation einer konkreten technischen Zeichnung definiert werden kann. Existieren mehrere Interpretationen, so soll die in einem gewissen Sinne optimale Interpretation auf effektive Art und Weise gefunden werden.
Ferner müssen die Grenzen, die logisch aus dem Konzept folgen, herausgearbeitet werden.

Wer anschaulich argumentiert, setzt sich leicht dem Vorwurf aus,
er würde gar nicht argumentieren, sondern nur gestikulieren...
Soll man deshalb allen anschaulichen Argumenten von
vornherein aus dem Wege gehen? Gewiß nicht.
Wenn man nur das bare Gold der strengen Beweise
immer als Deckung im Hintergrund hat,
dann ist das Papiergeld der Gesten ein unschätzbares Hilfsmittel
für schnelle Verständigung und raschen Gedankenumlauf.

K. Jänich

4 Optimale Interpretation Technischer Zeichnungen

4.1 Interpretation

Wäre die Modellierung einer Domäne vollständig, wäre also das gesamte einem Menschen zur Verfügung stehende Wissen modelliert worden, dann würde in den meisten Fällen die Ausgabemenge O_{out} lediglich ein einziges Objekt enthalten. Dieses Objekt würde den gesamten Inhalt der Zeichnung verkörpern und aufgrund seiner Komplexität auch gut und konfliktfrei erkannt werden. Nur wenn es tatsächlich völlig unabhängige Zeichnungsteile gäbe, wäre eine echte Menge von (dann auch konfliktfreien) Ausgabeobjekten das Ergebnis der Objekterzeugung. In der Praxis wird man jedoch bei weitem keine Vollständigkeit erreichen: So werden etwa Kaffeeflecken auf einer technischen Zeichnung kaum modelliert werden. Ein menschlicher Betrachter kann sein Allgemeinwissen nutzen, um den Kaffeefleck von einer Straßenbahnschiene zu unterscheiden. Er weiß auch, daß die vielen durchgestrichenen Straßennamen auf der alten Grundkarte von Dresden kein Artefakt, sondern Resultat von Straßenumbenennungen im Zuge der Wende in der DDR sind. Der Mensch wird also meist eine sehr singuläre und ideale Interpretation einer technischen Zeichnung erzeugen können. Er wird für jedes identifizierte Objekt eine Argumentationskette, die dessen Existenz eindeutig belegt, konstruieren können. Die inhärente Unvollständigkeit der Modellierung führt jedoch in einem System wie Y zu einer größeren Menge von Objekten in O_{out}. Diese sind ihrerseits auch weit weniger komplex als das oben erwähnte imaginäre Superobjekt. Durch die geringere Komplexität sind diese Objekte auch weniger stabil erkennbar, es gibt auch auf dem Niveau der Ausgabeobjekte Verwechslungsmöglichkeiten, d. h. Konflikte. So wird man die Modellierung von topologischen Karten auf dem Niveau der Legende und einiger zusätzlicher gut formalisierbarer Informationen („Höhenlinen beschreiben das Relief als Funktion der Lage.“, „Straßen bilden einen zusammenhängenden Graphen.“, „Flüsse bilden einen gerichteten Graphen ohne gerichtete Zyklen.“) belassen und Informationen, wie „In Deutschland existieren keine Berge über 8000 m.“ oder „Straßenbahnen fahren selten durch Gebäude“ vermutlich vergessen. Letzteres führt in nebenstehendem Ausschnitt aus dem Dresdner Stadtplan dazu, daß der Mensch die Fortsetzung der blauen Straßenbahnlinie (89) von Süd-Ost nicht in dem Bach, sondern sofort entlang der Straße sucht. Y wird dagegen die richtige Fortsetzung nicht „wissen“, sondern nur „vermuten“. Y wird verschiedene Varianten generieren und sich für die plausibelste Gesamtinterpretation (entsprechend einer numerischen Bewertung) entscheiden.

Unter Nutzung des *conflict* Begriffes kann definiert werden, was im Kontext dieser Arbeit unter der **Interpretation** einer Technischen Zeichnung verstanden wird.

Definition 26. *Eine **Interpretation** i ist eine konsistente, d. h. in sich konfliktfreie, Teilmenge von O_{out}:*

$$i := \left\{ o, o' \;\middle|\; \begin{array}{l} o, o' \in O_{out} \wedge \\ o \neq o' \wedge \\ \neg(o \; conflict \; o') \end{array} \right\} \qquad \textbf{(4-84)}$$

Weiterhin wird für eine Interpretation i gefordert, daß sie aus einer nicht erweiterungsfähigen Anzahl von Objekten besteht, das heißt:

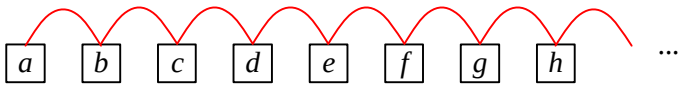
$$\forall o \in (O_{out} - i) : \exists o' \in i : o' \; conflict \; o \quad (Maximalitätsanforderung). \qquad \textbf{(4-85)}$$

Die Menge aller Interpretationen einer Zeichnung wird mit I bezeichnet.

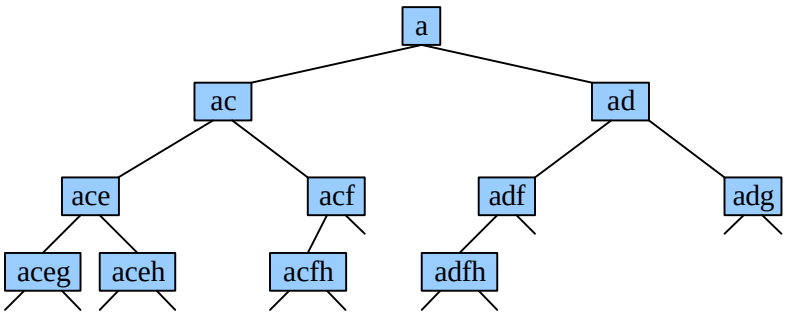
Aus der Literatur zu graphentheoretischen Problemen ist der hier mit Interpretation bezeichnete Gegenstand gut bekannt [Emden-Weinert Hougardy Kreuter Prömel Steger 1996]. Deutet man die Menge der Ausgabeobjekte O_{out} als Knoten eines Graphens und die Konfliktrelation als ungerichtete Kanten dieses Graphen, dann ist eine Interpretation eine **unabhängige** oder *stabile* Knotenmenge. Die Maximalitätsforderung (4-85) bewirkt jedoch, daß nicht jede **unabhängige** Knotenmenge eine Interpretation ist, sondern nur sogenannte **dominierende** Knotenmengen. Diese graphentheoretische Betrachtungsweise wird später von Nutzen sein, wenn untersucht wird, wie eine optimale Interpretation gefunden werden kann und wie komplex dieses Problem ist. Einen ersten intuitiven Anhaltspunkt für die Problemkomplexität liefert jedoch bereits der folgende Satz. Durch ihn wird deutlich, daß die explizite Aufzählung aller Interpretationen vermieden werden muß.

Satz 3. *Die Anzahl der möglichen Interpretationen ist im allgemeinen Fall exponentiell bezüglich der Anzahl der Objekte in O_{out}.*

Beweis. Um dies zu zeigen, betrachte man folgende Struktur von *n* Objekten aus O_{out} und deren Konflikte



Nach folgender Methode erhält man alle Interpretationen, die das erste Objekt *a* enthalten:
Wenn *a* in der Interpretation vorhanden ist, so ist entweder *c* oder *d* auch Element der Interpretation. Dies gilt wegen der Maximalitätsforderung für Interpretationen. Entscheidet man sich für *c*, dann ist auch entweder *e* oder *f* in der Interpretation... Es entsteht ein binärer Baum an dessen Blättern die Interpretationen stehen. Dieser Baum ist bis zu einer Tiefe von (*n*/4)-1 vollständig gefüllt. (Danach wird er von rechts beginnend immer dünner besetzt)



Das heißt es existieren weit mehr als $2^{\frac{n}{4}-1}$ Interpretationen, die *a* enthalten. Aus dem selben Grund gibt es mehr als $2^{\frac{n-1}{4}-1}$ Interpretationen, die *b* enthalten. Eine grobe Abschätzung für die Anzahl der Interpretationen beträgt also: $2^{\frac{n}{4}-1}+2^{\frac{n-1}{4}-1}>2^{\frac{n-1}{4}}$. (Bei 101 Objekten sind das >33554432 Interpretationen**)**

■

Die Forderung, daß eine Interpretation stets eine maximale Anzahl an Objekten enthalten soll, ist sehr wichtig und beruht auf folgender Überlegung:
Wenn innerhalb einer Interpretation nichts gegen die Existenz eines Objektes spricht (kein Konflikt zu diesem Objekt existiert), dann soll dieses Objekt existieren und ebenfalls zur Interpretation gehören. Dies gilt auch dann, wenn es nur eine geringe Wahrscheinlichkeit besitzt.
Beispielsweise könnte ein völlig isolierter kleiner Kreis in einem Gasleitungsplan als Zahl „0“ und als Zeichenkette „O“ sowie „o“ und auch als Symbol *Wassertopf* in die Menge der Ausgabeobjekte gelangt sein. Die Wahrscheinlichkeit sei für die ersten drei Klassen gleich 0.2 und für den *Wassertopf* 0.4. Damit ist allerdings die Wahrscheinlichkeit, daß kein *Wassertopf* vorliegt, recht groß (nämlich 0.6). Die Interpretation des gesamten Planes sollte jedoch den *Wassertopf* durchaus enthalten, da aufgrund der Isolation nichts gegen ihn spricht. Ist der *Wassertopf* doch das Objekt mit der maximalen Wahrscheinlichkeit.
Im Beispiel „*conflict*-Relation” gibt es unter anderem die Interpretationen I ={ i_1={B790/804, B206/804, Haus, L804}, i_2={B790/902, Haus, L902} ...}, wenn C_{out} :={*Haus*, *Leitung*, *Bemaßung*} festgelegt wird.

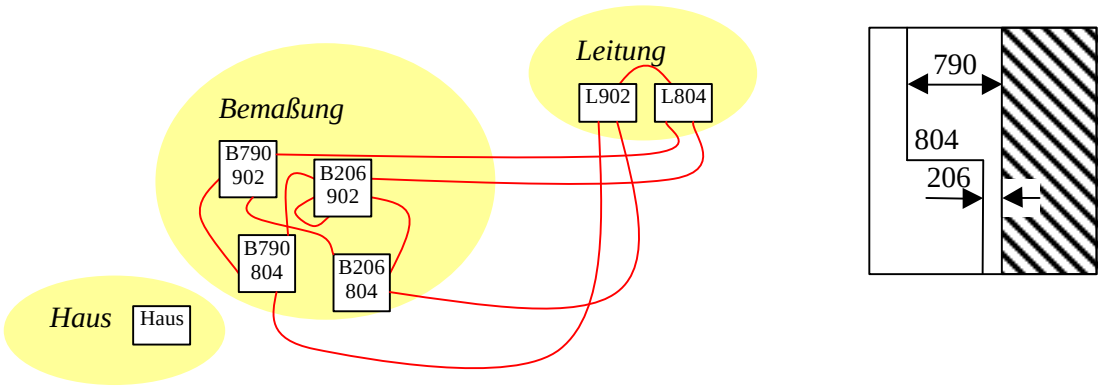


Abb. 4-30 Alle relevanten Konflikte aus dem Beispiel „*conflict*-Relation“

Das Objekt Haus steht zu keinem Objekt in Konflikt und gehört deshalb entsprechend der Maximalitätsforderung zu jeder Interpretation. Aus dem Beispiel geht hervor, daß es auch in der Praxis oft verschiedene Interpretationen einer Vorlage geben kann. Die finale Frage lautet daher: Wie werden die Interpretationen bewertet, und wie findet man eine optimale Interpretation?

4.2 Globales Optimalitätskriterium

In diesem Abschnitt wird zunächst ein abstraktes, globales Optimalitätskriterium *GO* für Interpretationen definiert. Auf dessen Grundlage wird ein kombinatorisches Optimierungsproblem formuliert und dessen NP-Vollständigkeit gezeigt.
Ein konkretes Optimalitätskriterium mit günstigen Eigenschaften wird vorgeschlagen. Anschließend wird untersucht, in welchem Verhältnis dieses zu wahrscheinlichkeitsbasierten Maßen steht. Dazu wird versucht, die *causes*-Struktur der Objekte einer technischen Zeichnung auf ein Bayesnetz abzubilden. Da dies nicht immer möglich ist, werden die Eigenschaften des konkreten Optimalitätskriteriums auch unter heuristischen Gesichtspunkten diskutiert.

4.2.1 Allgemeines Gütemaß *GO*

Bevor die Bewertung einer kompletten Interpretationen definiert werden kann, wird – zunächst nur abstrakt – die Bewertung potentieller Elemente einer Interpretation, also der Objekte aus O_{out} eingeführt.

Definition 27. *Die Funktionen* ***t***(o) *und* ***f***(o) *ordnen einem Objekt o jeweils eine nicht negative, reelle Zahl aus dem offenen Intervall*]0,1[*zu:*

$$t : O \rightarrow]0,1[\quad \textit{und} \quad f : O \rightarrow]0,1[\,. \qquad \textbf{(4-86)}$$

Ferner gelte die folgende Eigenschaft für alle Objekte:

$$t(o) > f(o) \,. \qquad \textbf{(4-87)}$$

Interpretierbar im Kontext dieser Arbeit ist $t(o)$ als Evidenz der Existenz von o (*true*). $f(o)$ ist entsprechend ein Maß für die Annahme, daß o nicht existiert (*false*). Mögliche Definitionen für diese Funktionen werden im Abschnitt 4.2.4 eingeführt und diskutiert.

Die Bewertung einer Interpretation wird als $GO : 2^{O_{out}} \rightarrow]0,1[$ definiert:

Definition 28. *Eine Interpretation i wird durch ein globales* ***Optimalitätskriterium***

$$GO(i) := \prod_{o \in i} t(o) \prod_{o' \in O_{out} - i} f(o') \qquad \textbf{(4-88)}$$

bewertet.

Die Bewertung *GO* ist also eine Funktion, die prinzipiell über allen Teilmengen von O_{out} definiert ist. Relevant ist sie jedoch nur für Interpretationen.

Beispiel *GO*. In der *conflict*-Struktur von Abb. 4-31 sind zwei Interpretationen

$$I = \{i_1 = \{a, c\}, i_2 = \{b\}\} \qquad \textbf{(4-89)}$$

definiert. Die Bewertungen dieser beiden Objektmengen mittels *GO* lauten:

$$GO(\{a,c\}) = t(a)\, f(b)\, t(c) \qquad \textbf{(4-90)}$$

$$GO(\{b\}) = f(a)\, t(b)\, f(c) \,. \qquad \textbf{(4-91)}$$

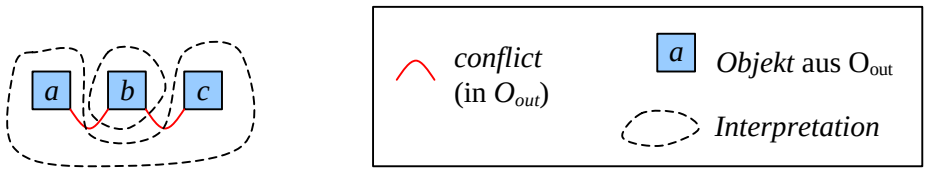


Abb. 4-31 Eine einfache *conflict*-Struktur und die beiden möglichen Interpretationen

Die Anzahl der in *GO* einfließenden Faktoren ist für verschiedene Interpretationen einer Vorlage immer gleich. Somit ist bereits anschaulich keine Normierung notwendig. Die *GO*s haben immer die selbe Größenordnung und ein Vergleich mehrerer Interpretationen mittels *GO* ist plausibel.

4.2.2 Optimierungsproblem Y

Definition 29. *Eine* ***optimale Interpretation*** *einer Zeichnung* i_{opt} *ist eine Lösung des Maximierungsproblems:*

$$i_{opt} := \arg\max_{i \in I} GO(i) \,. \qquad \textbf{(4-92)}$$

4.2.3 Komplexitätsbetrachtung

Der folgende Abschnitt beschäftigt sich mit der wichtigen Frage, ob es einen stets effizienten, d. h. polynomiellen Algorithmus geben kann, der eine optimale Interpretation i_{opt} liefert.
Um dieses Problem auf bekannte Erkenntnisse zurückführen zu können, werden in diesem Abschnitt einige graphentheoretische Konzepte benötigt [Emden-Weinert Hougardy Kreuter Prömel Steger 1996].

Definition 30. *Ein (endlicher) ungerichteter **Graph** ist ein Tupel $G=(V,E)$, wobei V eine (endliche) Menge und E eine binäre, symmetrische Relation über V, d. h. eine Teilmenge von $V\times V$ ist und $(a,b)\in E \Rightarrow (b,a)\in E$ gilt. Die Elemente von V heißen Knoten, die Elemente von E Kanten des Graphen. Ist (a,b) eine Kante, so heißen die Knoten a und b benachbart.*

Definition 31. *Eine Knotenmenge S eines Graphen G heißt **unabhängig** oder stabil, falls keine zwei Koten aus S benachbart sind.*

Definition 32. *Die Größe einer kardinalitätsmaximalen unabhängigen Knotenmenge eines Graphen G heißt **Unabhängigkeitszahl** $a(G)$.*

Definition 33. *Ein ungerichteter Graph $G=(V,E)$ ist **nicht zusammenhängend**, wenn es zwei disjunkte Teilmengen V_1 und V_2 von V gibt mit:*

$$(V_1 \times V_2) \cap E = \varnothing . \qquad \textbf{(4-93)}$$

Weiterhin werden in diesem Abschnitt einige Begriffe aus der Komplexitätstheorie verwendet.

Definition 34. *Ein **Problem** $\Pi(\Phi,\Gamma)$ ist ein Operator, durch den einer Menge von Eingangsinformationen Φ eine Menge von Ausgangsinformationen (oder Lösungen) Γ zugeordnet wird. Eine konkrete **Aufgabe** innerhalb des Problems Π ist $p(f,g)$ mit einer festgelegten Eingangsinformation (oder **Instanz**) $f\in\Phi$ und einer* zugehörigen *Ausgabeinformation $g\in\Gamma$.*

Definition 35. *Die Menge der Ausgabeinformationen eines **Entscheidungsproblems** ist {ja, nein}.*

Das folgende Entscheidungsproblem diene der Illustration und wird später noch eine wichtige Rolle spielen.

INDEPENDENTSET:
Gegeben sei ein ungerichteter Graph $G=(V, E)$ und eine natürliche Zahl k, die nicht größer als die Knotenanzahl $|V|$ ist.
Gilt $a(G) \geq k$?

Dieses Problem INDEPENDENTSET ist also formal ein Operator $\Pi(\Phi$ = Menge aller Graphen $\times$ N, Γ = {*ja, nein*}). Das Problem zu lösen bedeutet, einen Algorithmus anzugeben, der für alle Instanzen (G,k) die korrekte Antwort *ja* oder *nein* liefert.

Die Komplexitätstheorie ermöglicht eine Klasseneinteilung der Probleme nach den zur Lösung erforderlichen Ressourcen (Zeit und Speicher) vorzunehmen. Dabei müssen die Algorithmen zur Problemlösung nicht direkt betrachtet werden. Vielmehr werden Aussagen getroffen, welchen Verbrauch an Ressourcen der beste Algorithmus im schlechtesten Fall impliziert.

Die Problemklassen P, NP, NP-vollständig, sollen hier nur kurz und informell eingeführt werden [Großmann Terno 1993]:

- **P** (**p**olynomial deterministic) ist die Klasse von Problemen, für die es einen deterministischen Algorithmus gibt, der im schlechtesten Fall eine Zeitkomplexität aufweist, die durch ein Polynom der Größe der Eingangsinformation beschränkt ist.
- **NP** (**n**ondeterministic **p**olynomial) ist eine Obermenge von P. Dies sind Probleme für die ein nichtdeterministischer Algorithmus der folgenden Bauart existiert:
  (i) Orakelschritt: *Rate* einen Wert aus der Menge der möglichen Lösungen des Problems.
  (ii) Prüfschritt: *Prüfe* deterministisch in polynomieller Zeit, ob dieser Wert eine Lösung des Problems darstellt. Falls dies nicht zutrifft, gehe zu (i), sonst breche ab und gebe die Lösung aus.
  Es wird vermutet, daß $P \neq NP$ gilt.
- **NP-vollständig** sind Probleme, die - unter der Annahme $P \neq NP$ - zwar zu NP nicht aber zu P gehören. Sie stellen die schwierigsten Probleme innerhalb von NP dar. Es existieren vermutlich keine effizienten Algorithmen zu deren Lösung.

Die folgende Halbordnungsrelation wird zum Vergleich der Schwierigkeit zweier Problem, also der relativen Bewertung der Probleme, herangezogen.

Definition 36. *Ein Problem* $\Pi(\Phi,\Gamma)$ *heißt* ***polynomiell reduzierbar*** *auf ein Problem* $\Pi'(\Phi', \Gamma')$, *in Zeichen* $\Pi \propto \Pi'$, *wenn es eine polynomiell berechenbare Funktion* $f : \Phi \to \Phi'$ *gibt, so daß gilt:*

$$\forall f \in \Phi: \quad \exists \textit{ exponentiell berechenbare Funktion } g \,(\textit{mit } g : \Gamma' \to \Gamma): \\ g' \textit{ ist Ausgabe von } p'(f(f), g') \Rightarrow g(g') \textit{ ist Ausgabe von } p(f, g(g')). \qquad \textbf{(4-94)}$$

$\Pi \propto \Pi'$ deutet an, daß Π nicht schwerer als Π' ist, da jede Instanz p von Π anhand einer Instanz p' von Π' entschieden werden kann.

Definition 37. *Ein Problem* Π' *ist* **NP-hart**, *wenn es ein Problem* $\Pi \in$ NP-vollständig *gibt, mit* $\Pi \propto \Pi'$.

Probleme aus NP-hart sind nach dieser Definition zumindest genauso schwierig wie die aus NP-vollständig. Es existieren auch hier (vermutlich) keine effizienten, das heißt polynomiellen Algorithmen für deren Lösung.

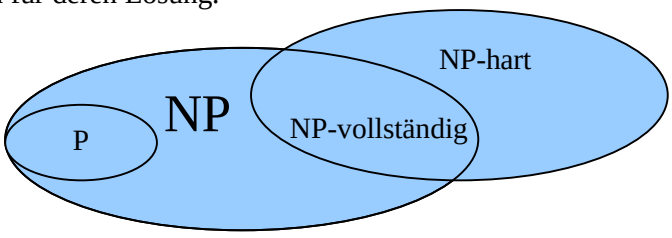


Abb. 4-32 Komplexitätsklassen; entnommen aus [Großmann Terno 1993]

Mit der Definition 37 ergibt sich nun auch eine elegante Möglichkeit, die NP-Härte eines Problems Π' nachzuweisen ohne die Menge aller Algorithmen zu dessen Lösung zu kennen. Man benötigt nur ein NP-vollständiges Problem Π, welches man auf Π' polynomiell reduzieren kann.
Eine wichtige Konsequenz für diese Arbeit lautet: Wenn man zeigen kann, daß die Bestimmung einer optimalen Interpretation einer technischen Zeichnung prinzipiell NP-hart ist, braucht man (falls $P \neq NP$) nicht versuchen, einen polynomiellen Algorithmus für die Lösung des Problems zu suchen.
INDEPENDENTSET ist ein Problem, welches sich zum Nachweis der NP-Härte der Suche nach einer optimalen Interpretation eignet, denn:

Satz 4. INDEPENDENTSET *ist* NP-vollständig.

Der Beweis erfolgt durch polynomielle Reduktion des Problems SAT (Erfüllbarkeit aussagenlogischer Formeln) auf INDEPENDENTSET und kann in [Emden-Weinert Hougardy Kreuter Prömel Steger 1996] nachgelesen werden. Der Nachweis für die NP-Vollständigkeit von SAT wurde 1971 durch Cook geführt.
Nun werden zwei neue Optimierungsprobleme OY und OYMULT eingeführt. OYMULT ist vollkommen äquivalent zum in Definition 29 genannten zentralen Problem dieser Arbeit. OY ist eine leichte Verallgemeinerung dieser Aufgabenstellung.
Man beachte, daß in der Formulierung dieser Optimierungsprobleme die Maximalitäts-eigenschaft (4-85), wie sie für eine Interpretation gelten soll, nicht erwähnt wird. Daß die Lösungen aller dieser Probleme diese Eigenschaft trotzdem besitzen, wird ebenfalls in diesem Abschnitt gezeigt.

OY:
Gegeben sei ein ungerichteter Graph $G=(V, E)$.
Eine beliebige total geordnete Menge R von möglichen Gewichten sei gegeben.
Über den Knoten sind jeweils Gewichte *t* und *f* definiert (*t* steht für true, *f* für false):

$$t, f : V \to \mathrm{R} . \qquad \textbf{(4-95)}$$

Es gelte folgende Einschränkung für die Gewichte aller Knoten $v \in V$:

$$t(v) > f(v) . \qquad \textbf{(4-96)}$$

Ferner existiere eine kommutative Funktion ⊗:

$$\otimes : R \times R \to R . \qquad \textbf{(4-97)}$$

Diese Funktion wird im weiteren als Operator verwendet:

$$a \otimes b \otimes c = \otimes(\otimes(a,b),c) . \qquad \textbf{(4-98)}$$

Ferner sei ⊗ in folgender Weise monoton:

$$a \otimes b = c \otimes d \wedge a > c \Rightarrow b < d . \qquad \textbf{(4-99)}$$

Das neutrale Element von ⊗ sei 1, d. h. $1 \otimes a = a$.
Eine Bewertungsfunktion $GO : 2^V \to \mathrm{R}$ einer Teilmenge *i* der Knoten des Graphen sei durch

$$GO(i) = \left(\bigotimes_{v \in i} t(v) \right) \otimes \left(\bigotimes_{v \in V - i} f(v) \right) \qquad \textbf{(4-100)}$$

gegeben. Gesucht ist eine *unabhängige* Knotenmenge i_{opt} mit maximaler Bewertung $GO(i_{opt})$.

Das folgende Problem ist eine praxisrelevante Instanz von OY, wobei R dem Intervall (0,1) rationaler Zahlen und ⊗ der Multiplikation entspricht:

OYMULT:
Dieses Problem ist eine Instanz von OY, wobei R dem offenen Intervall]0,1[rationaler Zahlen und ⊗ der Multiplikation entspricht:

$$GO(i) = \left(\prod_{v \in i} t(v) \right) \left(\prod_{v \in V - i} f(v) \right). \qquad \textbf{(4-101)}$$

Gesucht ist auch hier eine *unabhängige* Knotenmenge i_{opt} mit maximaler Bewertung $GO(i_{opt})$.

Bemerkung. OYMULT *ist nicht schwerer als* OY, *das heißt* OYMULT $\propto$ OY.

Auch OYMULT ist ebenso wie OY ein abstraktes Problem. Die Knotenbewertungen *t* und *f* bleiben undefiniert. Im nächsten Abschnitt werden verschiedene Konkretisierungen dieser Funktionen diskutiert.
Für die Ermittlung der Problemkomplexität wird das zugehörige Entscheidungsproblem zu OYMULT benötigt:

OYMULTDP:
Es sei (G=(V, E), t, f) eine Instanz von OYMULT.
Gibt es für eine vorgegebene rationale Zahl *go* eine *unabhängige* Knotenmenge i ($i \subseteq V$) mit einer Bewertung:

$$GO(i) \geq go\,? \qquad \textbf{(4-102)}$$

Eine Instanz von ist OYMULTDP also ein Tupel (G, t, f, *go*).

Lemma 1. OYMULTDP $\propto$ OYMULT.

Beweis. Kennt man durch Nutzung einer Lösung von OYMULT eine optimale Knotenmenge i_{opt}, dann kann man sofort auf das maximale *GO*, nämlich $GO_{max} = GO(i_{opt})$ schließen. Die Antwort auf das Entscheidungsproblem lautet also:

$$\text{OYMULTDP} = true \quad \Leftrightarrow \quad GO_{max} \geq go. \qquad \textbf{(4-103)}$$

Diese Reduktion von OYMULTDP auf OYMULT ist polynomiell.
■

Satz 5. OYMULTDP *ist* NP-hart.

Beweis. Wir wissen, daß INDEPENDENTSET NP-vollständig ist. Die Frage, ob $a(G) \geq k$ ist (G ist Graph und k ist eine natürliche Zahl, die kleiner gleich der Knotenanzahl von G ist), ist also im allgemeinen Fall nicht effizient zu beantworten.
Wir zeigen, daß INDEPENDENTSET $\propto$ OYMULTDP gilt, indem wir die allgemeine Instanz (G=(V,E), k) von INDEPENDENTSET polynomiell nach einer speziellen Instanz (G', t, f, *go*) von OYMULTDP transformieren.
Zu (G=(V,E), k) konstruieren wir folgende OYMULTDP-Instanz:
Der Graph wird einfach übernommen:

$$G' = G(V, E)\,. \qquad \textbf{(4-104)}$$

Es werden konstante Gewichte für alle Knoten $v \in V$ verwendet:

$$\begin{aligned} t(v) &= t \\ f(v) &= f\,. \end{aligned} \qquad \textbf{(4-105)}$$

Ferner sei die Konstante *go* folgendermaßen festgelegt:

$$go = t^k * f^{|V|-k}\,. \qquad \textbf{(4-106)}$$

Diese Reduktion von INDEPENDENTSET auf eine Instanz von OYMULTDP ist offensichtlich polynomiell.
Nun ist die folgende Identität zu zeigen:

$$\exists i \subseteq V, i \;\; ist \text{ unabhängig e } Knotenmenge : GO(i) \geq go \quad \Leftrightarrow \quad a(G) \geq k\,. \qquad \textbf{(4-107)}$$

In dieser speziellen OYMULTDP-Instanz ist $GO(i)$ - wegen der Konstanz der Gewichte - nur eine Funktion der Kardinalität von i:

$$\begin{aligned} GO(i) &= GO(|i|) \\ &= \left(\prod_{v\in i} t\right)\left(\prod_{v\in V-i} f\right) \\ &= t^{|i|} * f^{|V|-|i|}. \end{aligned} \qquad \textbf{(4-108)}$$

Die linke Seite von „⇔“ kann also äquivalent umgeformt werden:

$$\exists i \subseteq V, i \text{ ist unabhängige Knotenmenge}: t^{|i|} * f^{|V|-|i|} \geq t^k * f^{|V|-k}. \qquad \textbf{(4-109)}$$

Weil $t > f > 0$ für alle OYMULTDP-Instanzen gilt, ist $GO(|i|)$ eine streng monoton wachsende Funktion von $|i|$ und es kann daher folgender Koeffizientenvergleich vorgenommen werden:

$$\exists i \subseteq V, i \text{ ist unabhängige Knotenmenge}: |i| \geq k. \qquad \textbf{(4-110)}$$

Für eine offensichtlich immer existierende Knotenmenge i_{opt} mit $|i_{opt}| = a(G)$ ist diese Aussage äquivalent zu der rechten Seite von „⇔“: $a(G) \geq k$.
■

Bemerkung. OYMULT *ist nach* Lemma 1 *auch* NP-hart.

Der folgende Satz belegt, daß jede Lösung von OY (und damit jede Lösung von OYMULT) auch immer eine Interpretation im Sinne von Definition 26 ist. Die dort geforderte und inhaltlich motivierte Maximalitätsforderung (4-85) kann garantiert werden. Eine Lösung von OY ist eine optimale Interpretation einer technischen Zeichnung.

Satz 6. *Eine Lösung i_{opt} von* OY *kann kardinalitätsmäßig nicht mehr vergrößert werden, d. h. für i_{opt} und eine* OY*-Instanz* $(G(V, E), t, f)$ *gilt:*

$$\forall v \in (V - i_{opt}): \exists v' \in i_{opt}: (v', v) \in E. \qquad \textbf{(4-111)}$$

Beweis. Man muß zeigen, daß man zu einer *nicht* kardinalitätsmaximalen *unabhängigen* Knotenmenge i stets eine *unabhängige* Menge j findet, die i ganz enthält und eine größere Bewertung aufweist:

$$\forall i: \exists j: i \subset j \subseteq V, GO(j) > GO(i). \qquad \textbf{(4-112)}$$

Die Bewertung von i ist per Definition

$$GO(i) = \left(\bigotimes_{v\in i} t(v)\right) \otimes \left(\bigotimes_{v\in V-i} f(v)\right). \qquad \textbf{(4-113)}$$

Jede größere Menge j weist mindestens einen weiteren Knoten v' auf. Die Bewertung von j berechnet sich in Abhängigkeit von $GO(i)$, indem das erste Produkt um einen Faktor erweitert und das zweite Produkt um einen Faktor verringert wird. Das Verringern des zweiten Produkts um $f(v')$ erfolgt indirekt, indem zu $GO(j)$ dieser Faktor $f(v')$ hinzugefügt wird:

$$\begin{aligned} f(v') \otimes GO(j) &= \left(t(v') \otimes \left(\bigotimes_{v\in i} t(v)\right)\right) \otimes \left(\bigotimes_{v\in V-i} f(v)\right) \\ &= t(v') \otimes GO(i). \end{aligned} \qquad \textbf{(4-114)}$$

Weil $t(v')$ stets größer als $f(v')$ ist, ist die Bewertung $GO(j)$ wegen der Monotonie von ⊗ stets größer als $GO(i)$.

Jede gegenüber *i* umfangreichere Menge *j* hat also eine größere Bewertung als *i*. Da *i* nicht kardinalitätsmaximal ist, kann immer ein Knoten *v´* gefunden werden, der die Unabhängigkeitsforderung nicht verletzt und eine Menge *j* mit $j = i \cup \{v'\}$ erzeugt.
■

Eine Lösung von OY ist im Algorithmus „Interpretation“ in Abschnitt 4.3.1 implementiert.
Die beiden Probleme OY und OYMULT wurden aus folgendem Grunde unterschieden:
Da die im Weiteren betrachteten Konkretisierungen von *GO* für den Operator ⊗ immer die Multiplikation verwenden, wurde die Hartnäckigkeit des Problems auch für den Sonderfall OYMULT gezeigt. Das heißt, durch die Einschränkung auf die Multiplikation kann die NP-Härte nicht gebrochen werden. Der später dargestellte Algorithmus macht jedoch nur von den in OY geforderten algebraischen Eigenschaften von ⊗ Gebrauch.

4.2.4 Konkretisierungen von *GO*

Das Bewertungsmaß *GO* und insbesondere konkrete Funktionen *f* und *t* müssen unter zwei Gesichtspunkten diskutiert werden:
1. *GO* muß daraufhin geprüft werden, ob es in der Lage ist, „gute“ von „schlechten“ Interpretationen zu unterscheiden.
2. *GO* muß geeignet sein, die NP-harte Suche nach der besten Interpretation im durchschnittlichen Fall drastisch zu beschleunigen.

Für die Definition und Diskussion eines konkreten globalen Optimalitätskriteriums *GO*(*i*) einer Interpretation *i* werden einige neue Begriffe benötigt.

Definition 38. *Die Funktion component*(*o*) *liefert alle Objekte, die das Objekt o direkt beeinflussen:*

$$component(o) := \{o' \mid o' \text{ causes } o\}. \qquad \textbf{(4-115)}$$

Die bedingten Wahrscheinlichkeiten der Objekte P_o werden mittels eines Operators $\langle P_o \rangle$ auf ein Intervall]1/2, 1.0[abgebildet.

Definition 39.

$$\langle x \rangle := \frac{x+1}{2}. \qquad \textbf{(4-116)}$$

Durch diese Transformation ist sichergestellt, daß stets $\langle o \rangle$ > 1-$\langle o \rangle$ gilt. Nun kann ein erstes einfaches *GO* definiert werden, indem die noch nicht festgelegten objektspezifischen Bewertungen *t* und *f* angegeben werden.

Definition 40. *Für ein einfaches GO=GO*$_{simple}$ *wird definiert:*

$$t_{simple}(o) := \begin{cases} \langle P_o \rangle \prod_{o' \in component(o)} t_{simple}(o') & wenn\ \ component(o) \neq \emptyset \\ \langle P_o \rangle & sonst \end{cases} \qquad \textbf{(4-117)}$$

Analog werden die inversen Evidenzen durch

$$f_{simple}(o) := \begin{cases} (1 - \langle P_o \rangle) \prod_{o' \in component(o)} f_{simple}(o') & wenn\ \ component(o) \neq \emptyset \\ 1 - \langle P_o \rangle & sonst \end{cases} \qquad \textbf{(4-118)}$$

zusammengefaßt.

Die Forderung an die Gewichte $t_{simple}(o) > f_{simple}(o)$ ist durch die Konstruktion von $\langle o \rangle$ erfüllt.

Beispiel GO_{simple}. In der *causes*-Struktur von Abb. 4-33 sind zwei Interpretationen definiert. Die Bewertungen dieser beiden Objektmengen mittels *GO* lauten:

$$GO(\{e,g\}) = t_{simple}(e)\ f_{simple}(f)\ t_{simple}(g)$$
$$= \langle P_e\rangle\langle P_b\rangle\langle P_c\rangle\left(1-\langle P_f\rangle\right)\left(1-\langle P_c\rangle\right)\left(1-\langle P_d\rangle\right)\left(1-\langle P_a\rangle\right)\langle P_g\rangle\langle P_d\rangle\langle P_a\rangle \text{ bzw.} \quad \textbf{(4-119)}$$

$$GO(\{f\}) = f_{simple}(e)\, t_{simple}(f)\ f_{simple}(g). \quad \textbf{(4-120)}$$

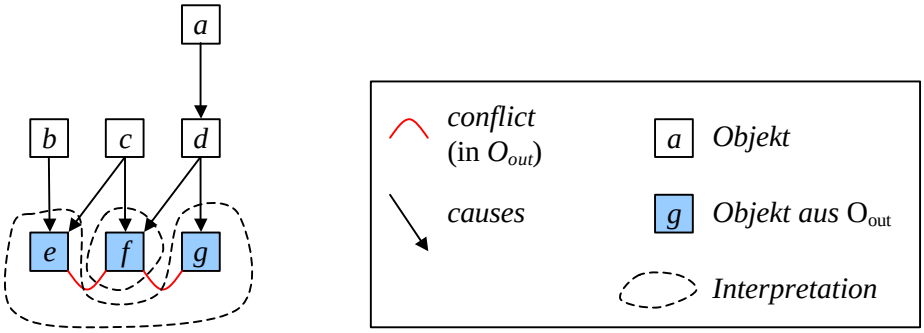


Abb. 4-33 Eine einfache *causes*-Struktur und die zwei möglichen Interpretationen

In diesem Beispiel wird auch offensichtlich, daß eine unvollständige Interpretation $\{e\}$ durch die Transformation der Wahrscheinlichkeiten $\langle . \rangle$ immer geringer als eine vollständige Interpretation $\{e, g\}$ bewertet wird. Das bedeutet, daß man bei der Lösung der Optimierungsaufgabe in Definition 29 die Maximalitätseigenschaft 4-2 nicht beachten muß.

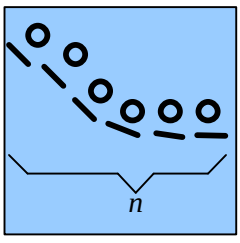
4.2.4.1 Gütemaß *GO* und Bayesnetz

Bei der Bewertung der Interpretationen ist nicht die Maßzahl der Bewertung von Interesse. Relevant ist nur die durch die Bewertung gegebene Ordnung der Menge der Interpretationen einer konkreten Zeichnung.
Das Kriterium *GO* wurde also genau dann vernünftig gewählt, wenn die Wahrscheinlichkeit der mittels *GO* gefundenen optimalen Interpretation größer ist, als die aller anderen Interpretationen. Etwas strenger ist folgende Monotonieforderung an ein optimales Kriterium GO_{opt}:

$$\forall i_1, i_2 \in I : P(i_1) > P(i_2) \Rightarrow GO_{opt}(i_1) > GO_{opt}(i_2). \quad \textbf{(4-121)}$$

Es ist zu untersuchen, ob oder unter welchen Bedingungen *GO* ein solches Kriterium GO_{opt} darstellt.
Die Berechnung der Wahrscheinlichkeit eine Interpretation erfordert im allgemeinen Fall Kenntnisse über bedingte Wahrscheinlichkeiten, die nicht vorliegen.

Beispiel IO. Die folgende Grafik stellt eine Zeichnung einer fiktiven Domäne dar. Die Primitive sind *Striche* und *Kreise*. Die Domäne besteht aus *Linien* mit verschiedenen Signaturen (ooo und ---) und aus *Zeichenketten*. Damit ergeben sich zumindest die folgenden beiden Interpretationen der Szene (o.B.d.A. n = 3): i_1={ooo , ---} und i_2={„IO“, „IO“, „IO“}.



Der Parallellauf der beiden *Linien* aus i_1 sei nicht modelliert worden. Es gibt keine Klasse, die verschiedene *Linien* z. B. aufgrund ihres Abstandes kombiniert. Das selbe gilt für die offensichtlich nicht zufällige Formation der *Zeichenketten* „IO“. Es gilt vermutlich nicht, daß das Auftreten zweier solcher

*Zeichenkette*n unabhängig voneinander ist. Damit läßt sich die Verbundwahrscheinlichkeit

$P(i_2)=P(\text{„I“}|\text{„O“ „IO“„IO“})* P(\text{„O“}|\text{„IO“ „IO“})* P(\text{„I“}|\text{„O“ „IO“})...$ **(4-122)**

nicht weiter vereinfachen. Die hier auftretenden bedingten Wahrscheinlichkeiten sind jedoch nicht bekannt. Die Wahrscheinlichkeiten $P(i_1)$ und $P(i_2)$ können also mit der modellierten Aggregationsstruktur und den berechneten Merkmalen nicht ermittelt werden.

4.2.4.1.1 Definition Bayesnetz

Um die Wahrscheinlichkeiten $P(i)$ aus (4-121) wenigstens in einigen Fällen aus den zur Verfügung stehenden bedingten Wahrscheinlichkeiten ableiten zu können, wird das Konzept der **Bayes**- oder **Beliefnetze** verwendet. [Russel Norvig 1995]

Definition 41. *Ein **Bayesnetz** ist ein gerichteter Graph $G=(V,E)$ ohne gerichtete Zyklen. Die Knotenmenge V ist eine Menge von Zufallsvariablen $V = \{A ...\}$. Die Variablen können jeweils diskrete Werte annehmen: Eine Variable A ist also mit genau einem der alternativen Werte ihres Definitionbereiches $Def(A) = \{a ...\}$ instanziiert:*

$$\sum_{a\in Def(A)} P(A=a)=1. \qquad \textbf{(4-123)}$$

Die Kanten E sind eine binäre Relation über den Knoten $E\subset V\times V$. Sie definieren eine Funktion $parents(A) := \{X \mid (X, A)\in E\}$. Zu jeder Variablen A existiert eine Wahrscheinlichkeitstabelle:

$$P(A \mid parents(A)): \underset{X\in(parents(A)\cup A)}{\times} Def(X) \rightarrow]0,1[. \qquad \textbf{(4-124)}$$

Für Knoten A ohne Eltern ($parents(A) = \varnothing$) entarten diese bedingten Wahrscheinlichkeiten $P(A \mid parents(A))$ zu den sogenannten a-priori-Wahrscheinlichkeiten $P(A)$.

Durch die gerichteten Kanten E ist weiterhin eine partielle Ordnung über den Variablen festgelegt. Die Kanten E zwischen den Variablen repräsentieren die direkten, kausalen Zusammenhänge der Variablen. Formal bedeutet dies, daß die folgende bedingte Unabhängigkeit bei der Betrachtung einer Variablen A_i gelten muß:

$$parents(A_i)\subseteq\{A_{i-1}\dots A_1\}\Rightarrow P(A_i \mid A_{i-1}\dots A_1)=P(A_i \mid parents(A_i)). \textbf{(4-125)}$$

Diese Aussage muß für alle Ordnungen der Variablen (A_1, A_2 ... A_{i-1}) gelten, die konsistent zur partiellen Ordnung der Variablen sind. In der nachfolgenden Abbildung sind die Ordnungen (D,B,C,A) und (D,C,B,A) konsistent. Bei Betrachtung der Variable B muß also analysiert werden, ob das Auftreten der Ereignisse in B nur abhängig ist vom Zustand der Variable D, wenn die Zustände der Variablen $\{A_{i-1}\dots A_1\} := \{D,C\}$ bekannt sind.

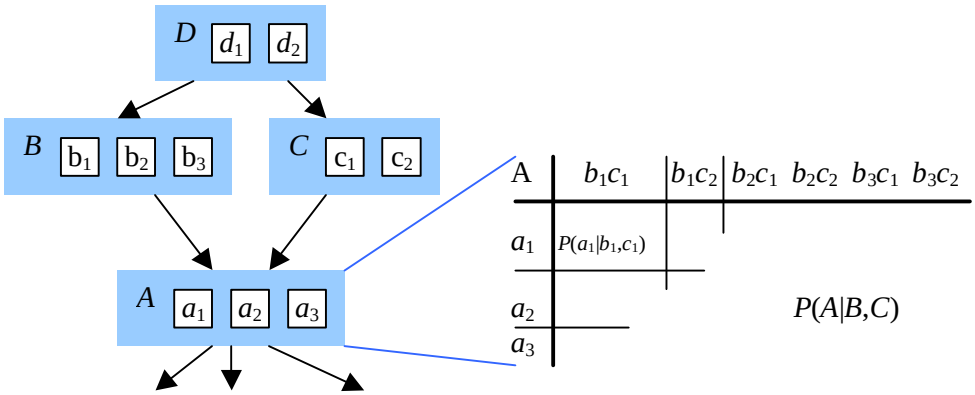


Abb. 4-34 Teil eines Bayesnetzes und die Wahrscheinlichkeitstab. der Variablen *A*

Ein Bayesnetz repräsentiert alle stochastischen Zusammenhänge der Zufallsvariablen in kompakter Form. Zu jedem möglichen Ereignis innerhalb der durch das Bayesnetz charakterisierten Domäne kann eine Wahrscheinlichkeit ermittelt werden:
Zunächst soll hier untersucht werden, wie die Wahrscheinlichkeiten $P(V)$, also die Verbundwahrscheinlichkeiten berechnet werden können. Aus Gleichung (4-125) folgt unmittelbar die Zerlegung

$$P(V) =$$

$$P(A_1 \ldots A_n) = \prod_i^n P(A_i \mid A_{i-1} \ldots A_1) \qquad \textbf{(4-126)}$$

$$= \prod_i^n P(A_i \mid parents(A_i)),$$

wenn (A_1, A_2 ...) eine Ordnung der Variablen darstellt, die konsistent zur partiellen Ordnung des zugrunde liegenden Graphen ist.
Gesucht sei nun eine Wahrscheinlichkeit $P(Ask)$, wobei *Ask* eine Teilmenge der Variablenmenge V sei und jede Variable in *Ask* einen konkreten Wert annimmt. Dann muß über alle möglichen Belegungen der restlichen Variablen *V-Ask* summiert werden:

$$P(Ask) = \sum_{a \in A} \cdots \sum_{b \in B} P(V) \text{ mit } V\text{-}Ask = \{A \cdots B\}. \qquad \textbf{(4-127)}$$

V in $P(V)$ nimmt in dieser Gleichung jeweils den Wert $V = \{A=a \ldots B=b \ldots\} \cup Ask$ an (siehe weiter unten Beispiel Bayesnetz).

Eine allgemeine Wahrscheinlichkeit in einem Bayesnetz ist von der Form $P(Ask \mid Tell)$, wobei *Ask* und *Tell* auch wieder Teilmengen der Variablenmenge V sind. Diese Wahrscheinlichkeiten ergeben sich aus der Definition der bedingten Wahrscheinlichkeit:

$$P(Ask \mid Tell) = \frac{P(Ask, Tell)}{P(Tell)} = \frac{P(Ask \cup Tell)}{P(Tell)}. \qquad \textbf{(4-128)}$$

Die Wahrscheinlichkeiten in Zähler und Nenner lassen sich wiederum entsprechend (4-127) berechnen. Das hier vorgestellte Verfahren ermöglicht zwar prinzipiell die Berechnung der Wahrscheinlichkeit aller durch das Bayesnetz beschriebenen Ereignisse, ist aber nicht effizient. Insbesondere für Bayesnetze mit einer Polytree-Struktur können Algorithmen mit polynomialer Komplexität angegeben werden [Russel Norvig 1995].

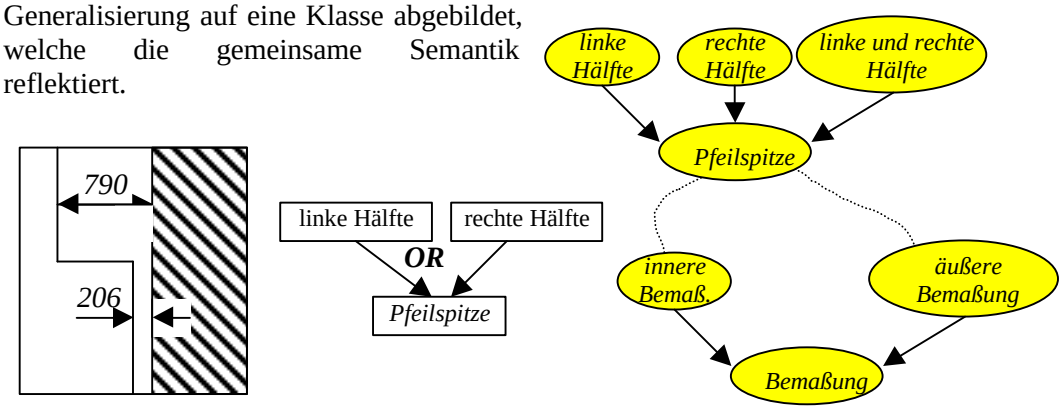
> **Beispiel Bayesnetz.** Im Bayesnetz von Abb. 4-34 soll eine sogenannte diagnostische (d. h. von der Wirkung zur Ursache verlaufende) Inferenz vorgenommen werden. Es ist beispielsweise die Wahrscheinlichkeit $P(d_1|a_2)$ aus den gegebenen Wahrscheinlichkeitstabellen zu bestimmen:

$$
\begin{aligned}
P(d_1|a_2) &= P\big(D = d_1 \mid A = a_2\big) \\
&= \frac{P(D,A)}{P(A)} \\
&= \frac{\sum_{b\in B}\sum_{c\in C} P(a_2 \mid b,c)P(b \mid d_1)P(c \mid d_1)P(d_1)}{P(A)} \\
&= \frac{\sum_{b\in B}\sum_{c\in C} P(a_2 \mid b,c)P(b \mid d_1)P(c \mid d_1)P(d_1)}{\sum_{b\in B}\sum_{c\in C}\sum_{d\in D} P(a_2 \mid b,c)P(b \mid d)P(c \mid d)P(d)}.
\end{aligned}
\qquad \textbf{(4-129)}
$$

4.2.4.1.2 Abbildung der *causes*-Struktur auf ein Bayesnetz

Bayesnetze sind ein mächtiges Beschreibungsmittel für die Unschärfe in komplexen Domänen. Das in dieser Arbeit vorgeschlagene Modellierungs- und Interpretationssystem für technische Zeichnungen Y geht einerseits über die Anwendbarkeit dieses strengen mathematischen Hilfsmittels hinaus. Insbesondere das Konzept der Zufallsvariablen wird verallgemeinert. Andererseits werden nicht alle Ausdrucksmöglichkeiten der Bayesnetze durch Y abgedeckt.:
Bayesnetze können unterschiedliche – nicht zwangsläufig alternative – Ursachen für ein Ereignis ausdrücken. Übersetzt man dies in die hier verwendete Terminologie hieße das: Es gäbe verschiedene Klassen, die ein und dasselbe Objekt erzeugten (Zum Beispiel könnte eine ‚8' aus zwei *Kreis*en, und gleichzeitig aus zwei ‚∈'-Kämmen aggregiert werden). Dies widerspräche dem Entwurfsparadigma und würde viel zusätzlichen Modellierungsaufwand (Wann sind zwei Objekte der Klasse *Acht* als identisch anzusehen?) nach sich ziehen. Hier wird davon ausgegangen, daß das Paradigma eingehalten wird. Nicht-alternative Objekterzeugungsmethoden müssen in einer einzigen Klasse gebündelt werden. Durch die Nutzung der Metaklasse Generalisierung wird dieses Vorgehen unterstützt. Alternative Objektarten können damit auf einem abstrakteren Niveau als identisch betrachtet werden.

Beispiel Bemaßung. Die möglicherweise fehlende Hälfte einer Pfeilspitze wäre in der Modellierung ein Kandidat für nicht-alternative Klassen: *linke Hälfte* bzw. *rechte Hälfte*. Man kann ein solches Verhalten auch in dem hier vorgeschlagenen System durch Aufzählung aller Varianten und der Metaklasse Generalisierung erreichen.
Die unterschiedlichen Bemaßungsarten (*innen* und *außen*) werden ebenfalls durch Generalisierung auf eine Klasse abgebildet, welche die gemeinsame Semantik reflektiert.



Lemma 2. *Die causes-Struktur einer technischen Zeichnung bildet ein Bayesnetz, wenn alle Objekte Teil genau einer Zufallsvariablen sind und die Unabhängigkeitsforderung* (**III**) *erfüllt ist.*

Beweis. Die Werte einer Zufallsvariable bilden ein Gesamtereignis. Jedes Objekt wird mit genau dem Wert einer Zufallsvariablen identifiziert: Man findet nach der Voraussetzung des Satzes eine Zerlegung V von O, so daß

$$\bigcup_{o\in O} V = O \quad \wedge \quad \forall (v,v') \in V \times V : v \cap v = \varnothing , \tag{4-130}$$

$$\forall v \in V : \left(\sum_{o\in v} P_o = 1 \quad \wedge \quad \forall (o,o') \in v \times v : o \; conflict \; o' \right) \tag{4-131}$$

gilt. Der Schwellwert bei der Objektgenerierung b_c muß wegen $\sum_{o\in v} P_o = 1$ für alle Klassen Null sein. Es werden alle potentiellen Objekte erzeugt. Die Modellierung ist bezüglich eines bestimmten semantischen Niveaus vollständig.
Die bedingte Unabhängigkeit (**III**) ist äquivalent zu (4-125) aus der Bayesnetzdefinition. Damit bilden die Objektmengen V die Zufallsvariablen eines Bayesnetzes. Die Kanten E des Bayesnetzes ergeben sich trivial aus der *causes*-Relation:

$$(v,v') \in E \Leftrightarrow \exists (o,o') \in v \times v' : o \; causes \; o'. \tag{4-132}$$

■

Ein solches Bayesnetz besitzt eine Wald-Struktur, denn jedes Objekt darf nur Objekte einer Zufallsvariablen beeinflussen (*causes*-Relation). Andernfalls würden die zwei Nachfolgerobjekte verschiedener Variablen zueinander in *conflict* stehen. Die beiden Variablen wären dann nicht mehr unabhängig voneinander.
Die Wurzeln der Bäume beinhalten die alternativen Ausgabeobjekte O_{out}. Das Ganze funktioniert, wenn die Modellierung der Domäne in den Klassen zu einer hierarchischen Segmentierung der Vorlage führt. Die Konfliktgraphen zwischen den Objekten sind dann jeweils vollvernetzt. Sie bilden also Zufallsvariablen entsprechend der Bayesnetz-Definition.
In der Abb. 4-35 wurde eine fiktive *causes*-Struktur mit den Objekten a_1 ... f_2 auf ein Bayesnetz abgebildet. Die breiten Pfeile mit den großen Spitzen bezeichnen die Kanten des resultierenden Bayesnetzes. Die alternativen Interpretationen lauten $\{a_1\}$ $\{a_2\}$ und $\{a_3\}$. Auch sie bilden eine Zufallsvariable (A). In einem zusammenhängenden Graphen (Baum) gibt es also nur einelementige Interpretationen. In einem nichtzusammenhängenden Graphen (Wald) hat jede Interpretation so viele Elemente wie der Graph Zusammenhangskomponenten.
Die Konflikte zwischen den primitiven und den Ausgabeobjekten wurden in der Abbildung ebenfalls eingezeichnet.

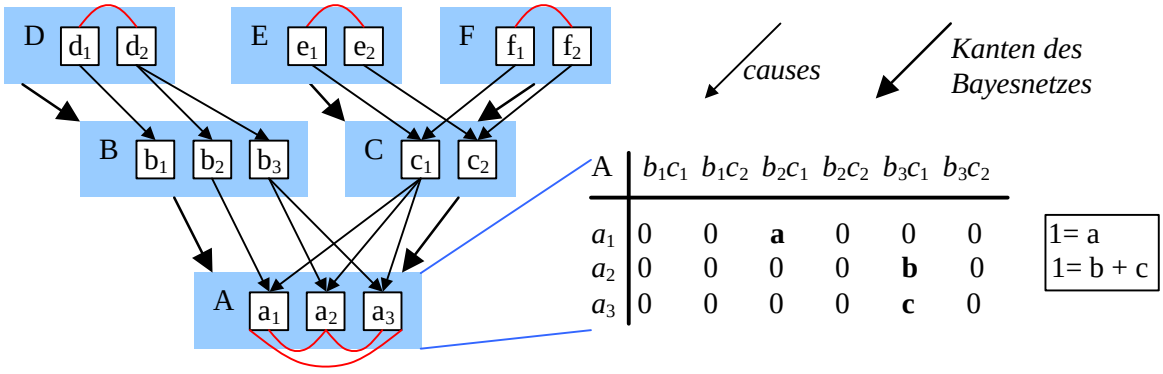


Abb. 4-35 Betrachtung einer geeigneten *causes*-Struktur als Bayesnetz

Die Wahrscheinlichkeitstabellen sind nur dünn besetzt und weisen folgende Eigenschaften auf:

1. Die Spaltensumme ist gleich Eins. Alle Objekte werden generiert. Die Schwellwerte *b* sind Null.
2. Jede Zeile besitzt nur genau einen Eintrag. Die Objekte können nur auf genau eine Art aus primitiveren Objekten aggregiert werden. Das folgt aus der Definition von *causes*.
3. Alle Werte ungleich Null entsprechen den gegebenen bedingten Wahrscheinlichkeiten P_o der Objekte *o*.

Satz 7. *Kann die causes-Struktur einer interpretierten Vorlage auf ein Bayesnetz abgebildet werden, ergibt sich die Wahrscheinlichkeit einer Interpretation i aus dem Produkt der bedingten Wahrscheinlichkeiten ihrer Elemente und derer (indirekten) Komponenten:*

$$P(i) = \prod_{o \in (i \cup \{o' \in O \mid \exists o'' \in i : o' \overline{causes}\ o''\})} P_o \quad . \qquad \textbf{(4-133)}$$

Beweis. Die letzte Variable des Bayesnetzes A_n enthalte alle Interpretationen I. Sie nimmt also als Wert das einzige Objekt o in i an. Die Wahrscheinlichkeit wird entsprechend (4-127) als Summe dargestellt:

$$\begin{aligned} P(i) &= P(A_n = o) \text{ mit } o \in i \\ &= \sum_{a \in A} \cdots \sum_{b \in B} P(V) \text{ mit } A...B \in V\text{-}A_n . \end{aligned} \qquad \textbf{(4-134)}$$

Nur der die *causes*-Struktur von o repräsentierende Summand verschwindet in dieser Summe nicht. Alle anderen Verbundwahrscheinlichkeiten enthalten mindestens einen Faktor Null (wegen der 2. Bemerkung zu den Wahrscheinlichkeitstabellen):

$$P(i) = P(A_n \in i, A_{n-1} causes\ A_n ... A_1 \overline{causes}\ A_n) . \qquad \textbf{(4-135)}$$

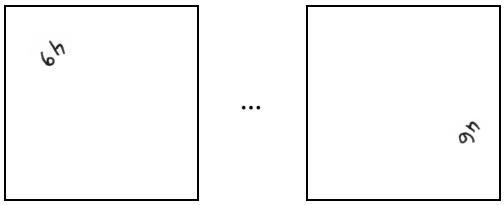
Wird nun die Gleichung (4-126) eingesetzt, erhält man die Aussage des Satzes:

$$\begin{aligned} P(i) &= \prod_{i}^{n} P(A_i \mid parents(A_i)) \\ &= \prod_{o \in (i \cup \{o' \in O \mid \exists o'' \in i : o' \overline{causes}\ o''\})} P_o \quad . \end{aligned} \qquad \textbf{(4-136)}$$

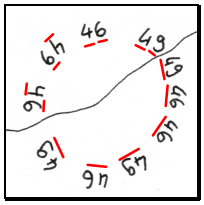
■

Das folgende ausführliche Beispiel demonstriert, daß es tatsächlich Domänen gibt, die sich in der beschrieben Art und Weise stochastisch modellieren und analysieren lassen und die technischen Zeichnungen zumindest ähneln. Es zeigt auch, wie eine unvollständige Modellierung zu Unsicherheit führt. Ausgehend von einer konstruierten, deterministischen „Welt“ wird eine eingeschränkte und damit zwangsläufig nichtdeterministische „Weltsicht“ erfunden.

Beispiel Zahlenkreis. Ein Zeichnung enthält stets genau zwei nebeneinanderstehende Ziffern. Diese können eine oder zwei Zahlen bilden. Die linke Ziffer ist immer eine ‚4‘, die rechte entweder eine ‚6‘ oder eine ‚9‘. Die gesamte Domäne beinhalte zehn solcher Zeichnungen. Überlagert man alle diese zehn gültigen Zeichnungen, entsteht ein Kreis von Zahlen. Die Aufzählung aller gültigen Zeichnungen lautet:



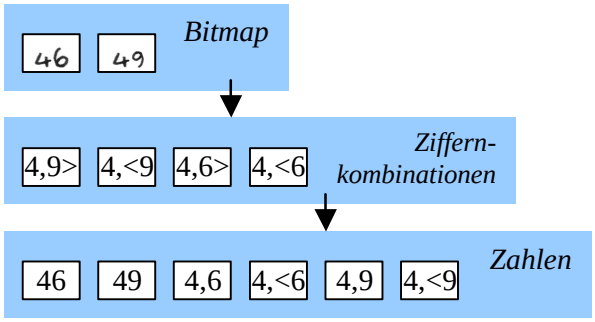
Überlagerung aller gültigen Zeichnungen und wahre Interpretation:

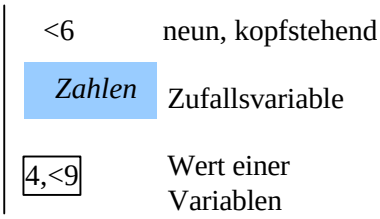


Das folgende Wissen sei **nicht** modelliert worden, und führt zu Unsicherheit bei der Interpretationen einer Zeichnung: Die Ziffern ‚6‘ und ‚9‘ haben ihren Fußpunkt immer zur Kreismitte. Über der imaginären Linie (siehe rechte Abbildung) befinden sich nur einstellige Zahlen, unter der Linie nur zweistellige. In der rechten Abbildung sind außerdem die Grundlinien der Zahlen dargestellt.
Damit ergibt sich folgende Verteilung der Wahrscheinlichkeit (Häufigkeit) für die wirkliche Semantik einer Zeichnung:

Interpretation *i*	**46**	**49**	**4,9**	**4,6**
***P*(*i*)**	3/10	3/10	2/10	2/10

Diese Domäne unterscheidet sich fundamental von realen Zeichnungsdomänen: So gibt es tatsächlich eine Wahrscheinlichkeit für jede konkrete Zeichnungsinterpretation und die Anwendung eines Bayesnetzes ist gerechtfertigt.
Zur Interpretation der Vorlagen wird folgendes Bayesnetz verwendet:



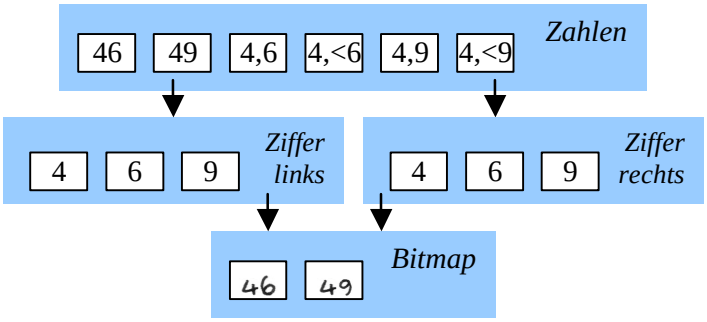


Die Entscheidung, ob eine Ziffer ‚6' oder ‚9' vorliegt, ist nicht unabhängig von der Richtung der benachbarten ‚4', denn zweistellige Zahlen sind wahrscheinlicher als einstellige. Deshalb kann das Erkennen der Ziffern nicht lokal (in einer eigenen Zufallsvariable *Ziffer* mit *Def*(*Ziffer*) ={‚4',‚6',‚9'}) erfolgen. Die notwendigen bedingten Wahrscheinlichkeiten für die kausale Inferenz *P*(*Zahlen*|*Bitmap*) lauten:

Ziffernkombi.	46	49
4,9>	0	4/5
4,<9	0	1/5
4,6>	4/5	0
4,<6	1/5	0

Zahlen	**4,9>**	**4,<9**	**4,6>**	**4,<6**
46	0	0	3/4	0
49	3/4	0	0	0
4,6	0	0	1/4	0
4,<6	0	0	0	1
4,9	1/4	0	0	0
4,<9	0	1	0	0

Dieses Netz gibt erwartungsgemäß folgende Wahrscheinlichkeiten aus:
P(*Zahlen* = 46|*Bitmap* =46) = P(*Zahlen* = 49|*Bitmap* =49) = 3/4*4/5 = 3/5
P(*Zahlen* = 4,6|*Bitmap* =46) = P(*Zahlen* = 4,9|*Bitmap* =49) = 1/4*4/5 = 1/5
P(*Zahlen* = 4,<6|*Bitmap* =46) = P(*Zahlen* = 4,<9|*Bitmap* =49) = 1*1/5 = 1/5
P(*Zahlen* = 46|*Bitmap* =49) = P(*Zahlen* = 4,6|*Bitmap* =49) = 0 ...
Bei jeder Vorlage würde also entweder die Zahl 46 oder die Zahl 49 als beste Interpretation ausgegeben werden, was nach dem im Netz gespeicherten Wissen optimal (wenn auch nicht immer richtig) ist.
Elegant und natürlich läßt sich ein Bayesnetz formulieren, wenn die wirkliche Kausalität, d. h. wenn Top-Down modelliert wird. Für jede der beiden Ziffern wird eine eigene Zufallsvariable verwendet.



Ziffer r.	**46**	**49**	**4,6**	**4,<6**	**4,9**	**4,<9**
4				0		
6	1	0	1	0	0	1
9	0	1	0	1	1	0

Ziffer l.	**46**	**49**	**4,6**	**4,<6**	**4,9**	**4,<9**
4				1		
6	0	0	0	0	0	0
9	0	0	0	0	0	0

Bitmap	**4;6**	**4;9**	**6;4**	**...**
46	1	0	0	
49	0	1		

Zahlen	
46	3/10
49	3/10
4,6	
4,<6	1/10
4,9	
4,<9	

Hier ist nun eine diagnostische Inferenz durchzuführen. Beispielsweise:

$$
\begin{aligned}
P(Zahlen = 46 \mid Bitmap = 46) &= P(46, 46) * 1/P(46) \\
&= P(46|46)*P(46)*1/P(46) \\
&= P(46|46)*P(46)*1/\,\Sigma\big(P(46|46)*P(46), \\
&\qquad P(46|4,6)*P(4,6),\ P(46|4,<6)*P(4,<6)\big) \\
&= 1*3/10*1/\,\Sigma(3/10+1/10+1/10) = 3/5.
\end{aligned}
$$

Die beschriebene unvollständige Modellierung der „Welt“ führt auch bei dieser Variante zu einer wahrscheinlichkeitsoptimalen, aber nicht immer zu einer richtigen Interpretation der Vorlage.

Beide im Beispiel vorgenommenen Modellierungen sind unbefriedigend. Diese Beobachtung kann zumindest anschaulich verallgemeinert werden:

- **Bottom Up.** Es entsteht ein sehr unnatürliches Netz. Die im Bayesnetz einzuhaltenden bedingten Unabhängigkeiten verhindern ein modulares Aufteilen der Erkennung. Alle potentiell einander beeinflussende Klassen müssen in einem Schritt erkannt werden. Vorteilhaft ist, daß die bei dieser Modellierung notwendigen bedingten Wahrscheinlichkeiten (die P_o) bekannt sind. Diese werden bereits bei der Objekterzeugung verwendet.
- **Top Down.** Bei dieser den Objekterzeugungsprozeß nachvollziehenden Modellierung sind die bedingten Wahrscheinlichkeiten immer konstant Null oder Eins. Dies liegt daran, daß jeder konkrete Zeichnungsinhalt immer nur auf einem einzigen Weg entstehen kann. Diese Einschränkung wurde bereits in 4.2.4.1.2 motiviert. Die Unschärfe steckt nur in den – in der Praxis schwierig definier- und bestimmbaren – a-priori-Wahrscheinlichkeiten der Top-Objekte.

Trotzdem die beiden dargestellten stochastischen Modellierungsansätze nicht zu einem direkt verwertbaren Bewertungsschema für Interpretationen führen, ist das Bayesnetzkonzept dennoch hilfreich, um die Güte eines globalen Optimalitätskriterums wenigstens in einigen Fällen zu belegen.

Satz 8. *Für alle causes-Strukturen, die sich auf ein Bayesnetz abbilden lassen, ist die Aussage*

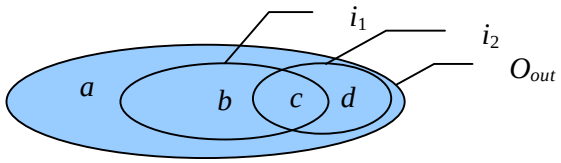
$$\forall i_1, i_2 \in I : \left(P(i_1) > P(i_2) \Rightarrow GO(i_1) > GO(i_2)\right) \quad \textbf{(4-137)}$$

wahr, wenn $\prod_{o \in i_1 - i_2} f_{simple}(o) < \prod_{o \in i_2 - i_1} f_{simple}(o)$ *gilt.*

Die Bedingung für die Gültigkeit der Monotonieforderung ist nicht sehr anschaulich. Eine Simulation hat ergeben, daß sie jedoch mit einer Häufigkeit von ca. 99% erfüllt ist. Dazu wurden zufällig eine Million causes-Strukturen wie in Abb. 4-34 erzeugt. Die Anzahl der Objekte wurde als im Intervall [1,1000] gleichverteilt angenommen. Die notwendigen bedingten Wahrscheinlichkeiten P_o wurden ebenfalls gleichverteilt generiert.

Beweis. Es wird o.B.d.A. folgende Partitionierung von O_{out} eingeführt:

$$\begin{aligned} a &:= (O_{out} - i_1) - i_2 \\ b &:= i_1 - i_2 \\ c &:= i_1 \cap i_2 \\ d &:= i_2 - i_1. \end{aligned} \quad \textbf{(4-138)}$$



Die rechte Seite von „⇒“ kann damit zunächst separat umgeformt werden:

$$\prod_{o \in b} t_{simple}(o) \prod_{o \in d} f_{simple}(o) > \prod_{o \in d} t_{simple}(o) \prod_{o \in b} f_{simple}(o). \quad \textbf{(4-139)}$$

Für die Umformung der linken Seite wird von Satz 8 und der Monotonie des $\langle\ \rangle$ Operators Gebrauch gemacht:

$$\prod_{o \in b} t_{simple}(o) > \prod_{o \in d} t_{simple}(o). \quad \textbf{(4-140)}$$

Insgesamt ist damit folgende Aussage zum „⇒“ Ausdruck äquivalent:

$$\begin{aligned} &\prod_{o \in b} t_{simple}(o) > \prod_{o \in d} t_{simple}(o) \Rightarrow \\ &\prod_{o \in b} t_{simple}(o) \prod_{o \in d} f_{simple}(o) > \prod_{o \in d} t_{simple}(o) \prod_{o \in b} f_{simple}(o). \end{aligned} \quad \textbf{(4-141)}$$

Damit ergibt sich eine einzige Voraussetzung für das Gelten der Monotonieeigenschaft:

$$\prod_{o \in d} f_{simple}(o) > \prod_{o \in b} f_{simple}(o). \quad \textbf{(4-142)}$$

■

4.2.4.2 Gütemaß *GO* und Heuristik

Der streng wahrscheinlichkeitstheoretische Zugang zu *GO* scheint dem Autor für die praktische Anwendung nicht besonders hilfreich zu sein, weil das in den Objektwahrscheinlichkeiten gespeicherte Wissen nicht für eine stochastische Bewertung der gesamten Interpretation ausreicht. Ein allgemeines, eine hinreichend wichtige Menge von Domänen abdeckendes, probabilistische Konzept konnte nicht gefunden werden.

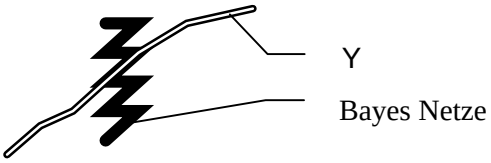


Abb. 4-36 Modellierbare Domänen; Überlappung von Y und Bayesnetzen

In diesem Abschnitt werden die Objektwahrscheinlichkeiten P_o nur noch als gleichberechtigte Gewichte der einzelnen Objekte aufgefaßt. Ihre Bedeutung als Wahrscheinlichkeit wird nicht mehr verwertet.

Dagegen wird die hervorragende Eigenschaft von *GO* – die Normierung durch die konstante Faktorenanzahl in diesem Abschnitt verallgemeinert. Es wird eine neue Konstruktion der Teilbewertungen *t* und *f* angegeben. In *GO* erreicht man die Normierung durch eine konstante Anzahl von beteiligten Faktoren für jede Interpretation einer Zeichnung. Bei dem Beispiel IO zeigen sich die Nachteile von GO_{simple}:

Beispiel IO-2. Angenommen die Wahrscheinlichkeit der Objekterzeugung ist für *alle* Objekte in Beispiel „IO“ konstant:

$$P(ooo \mid o,o,o) = P(---\mid -,-,-) = P("IO"\mid 'I','O') = p \qquad \textbf{(4-143)}$$

Dann ergibt sich für die Interpretationen i_1 und i_2 folgende Bewertungen:

$$\begin{aligned} P(i_1) &= t(ooo)t(---)(f("IO"))^n \\ &= \langle p\rangle^2(1-\langle p\rangle)^n \\ P(i_2) &= f(ooo)f(---)(t("IO"))^n \\ &= (1-\langle p\rangle)^2\langle p\rangle^n . \end{aligned} \qquad \textbf{(4-144)}$$

Wenn $n > 2$ wird immer i_2 immer bevorteilt.

Folgende Eigenschaften soll ein heuristisches, normiertes GO_{norm} aufweisen:

- Interpretationen aus wenigen „großen“ Objekten sollen gleiche Chancen besitzen wie Interpretationen aus vielen „kleinen“ Objekten.
- Alternative Modellierungen eines Sachverhaltes sollen den Ausgang der Optimierung nicht beeinflussen. (Wenn beispielsweise eine Klasse *c* über *m* Zwischenklassen aus einer anderen Klasse *c´* komponiert wird, sollen die *m* Zwischenobjekte nicht automatisch zu einer Verringerung der Bewertung eines Objektes aus *c* gegenüber einer Modellierung, bei der *c* direkt aus *c´* erzeugt wird, führen.)

Daraus ergibt sich eine gut formalisierbare Normierungsidee: Die Bewertung einer Interpretation wird auf den Anteil, der durch sie verwerteten Primitive bezogen. Man geht von *gleichwichtigen* Primitiven aus.

Definition 42. *Die Funktion* ***primitive***(*o*) *liefert alle primitiven Objekte, die im Objekt o (indirekt) enthalten sind:*

$$primitive(o) := \{o' \mid o'\ \overline{part\_of}\ o \wedge o' \in O_{primitive}\}. \qquad \textbf{(4-145)}$$

Die Funktion ***component***(*o*) *liefert alle Objekte, die das Objekt o (indirekt) beeinflussen und o selbst:*

$$component(o) := \{o' \mid o'\ \overline{causes}\ o\} \cup \{o\} \qquad \textbf{(4-146)}$$

Definition 43. *Die objektspezifischen Bewertungen von* $\boldsymbol{GO_{norm}}$ *lauten:*

$$t_{norm}(o) := \left\langle \sqrt[|component(o)|]{\prod_{o' \in component(o)} P_{o'}} \right\rangle^{|primitive(o)|} \quad \textbf{(4-147)}$$

$$f_{norm}(o) := \left(1 - \left\langle \sqrt[|component(o)|]{\prod_{o' \in component(o)} P_{o'}} \right\rangle \right)^{|primitive(o)|} \quad \textbf{(4-148)}$$

t_{norm} und f_{norm} genügen offensichtlich der Bedingung $t_{norm}(o) > f_{norm}(o)$.

Die Objektwahrscheinlichkeiten $P_{o'}$ werden zunächst durch geometrische Mittelwertbildung normiert. Anschließend wird die Anzahl der eingehenden primitiven Objekte einbezogen und so der gewünschte Effekt erzielt.

Satz 9. *Für alle Zeichnungen und Modelle gilt: Wenn alle Objekte aus O mit der selben Wahrscheinlichkeit erzeugt wurden, dann werden auch alle Interpretationen, die die selbe Zahl an Primitiven überdecken, mit* GO_{norm} *gleich bewertet.*

Beweis. Die identische Wahrscheinlichkeit aller Objekte *o* sei $P_o = P$. Dann sind also die lokalen Bewertungen

$$t_{norm}(o) = \langle P \rangle^{|primitive(o)|} \text{ und } f_{norm}(o) = (1 - \langle P \rangle)^{|primitive(o)|} \quad \textbf{(4-149)}$$

unabhängig von *component*(*o*) und damit auch unabhängig vom Weg der Aggregation der Primitive zum Objekt *o*. Alternative Modellierungen, die eine fixierte Menge von Primitiven zu einem Objekt *o* vereinigen, führen zur selben lokalen Bewertung. Die Bewertung einer ganzen Interpretation *i* ist nur von der Anzahl *a* der überdeckten primitiven Objekte

$$\begin{aligned} a(i) &= \left| \bigcup_{v \in i} primitive(v) \right| \\ &= \sum_{v \in i} |primitive(v)| \end{aligned} \quad \textbf{(4-150)}$$

abhängig:

$$\begin{aligned} GO_{norm}(i) &= \left(\prod_{v \in i} t_{norm}(v) \right) \left(\prod_{v \in V - i} f_{norm}(v) \right) \\ &= \langle P \rangle^{a(i)} (1 - \langle P \rangle)^{|V| - a(i)}. \end{aligned} \quad \textbf{(4-151)}$$

■

Bei der Verwendung GO_{norm} von wird nun auch das Problem aus Beispiel IO-2 zugänglich. Es gilt:

$GO_{norm}(i_1) = GO_{norm}(i_2)$.

4.3 Suche einer Optimalen Interpretation

„Branch and bound ist eine sehr flexible Technik, um effektive Lösungsverfahren ... zu entwickeln. ... branch and bound [ist] die angepaßte Lösungsmethodik für diskrete NP-harte Optimierungsprobleme. In der Methode branch and Bound wird die mathematische Technik der Fallunterscheidung kultiviert, Anschaulichkeit wird durch die graphentheoretische Baumstruktur erreicht, eine effektive Implementierung gehört direkt zur Methodik und kommt ohne gute Informatikkenntnisse nicht aus.“ (Einleitung von Kapitel 10 in [Großmann Terno 1993])

In diesem Abschnitt wird kein allgemeiner Zugang zu branch and bound (siehe dazu auch [Lengauer 1990]) sondern direkt ein entsprechendes Verfahren zur Lösung des Optimierungsproblems OY und damit OYMULT gegeben. Von diesen wurde NP-Härte durch den Beweis von Satz 5 und Lemma 1 bereits nachgewiesen und damit die Verwendung eines branch and bound Verfahrens motiviert.

Zunächst wird eine sehr wichtige Optimierungsmöglichkeit außerhalb der branch and bound Technik beschrieben: Eine Instanz von OY kann nämlich in vielen Fällen so zerlegt werden, daß die Lösung des eigentlichen Problems auf OY-Probleme mit wesentlich verringerter Knotenkardinalität reduziert werden kann. Die mit geringem Zeitaufwand verbundene Vereinigung der Teillösungen ist die Gesamtlösung.

Satz 10. *Separierbarkeit der Optimierung. Besteht der Graph G einer Instanz* $(G(V,E),t,f)$ *von* OY *aus n nicht zusammenhängenden Teilgraphen* $G_1 \ldots G_n$*, erzeugen diese Graphen n weitere Instanzen von* OY*. Die Lösung* i_{opt} *der ursprünglichen Instanz ergibt sich dann aus der Vereinigung der Lösungen* $i_{opt,1} \ldots i_{opt,n}$ *der neuen Instanzen* $(G_1(V_1,E_1),t,f) \ldots (G_n(V_n,E_n),t,f)$.

Beweis. Die Vereinigung

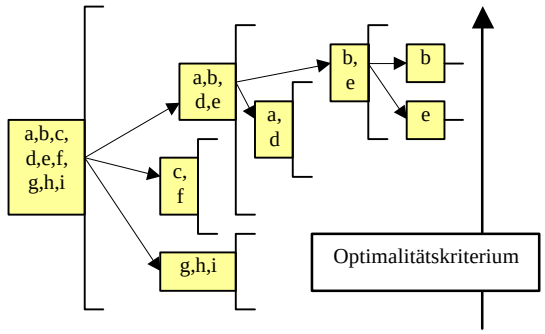
$$i = \bigcup_{j=1}^{n} i_{opt,j} \qquad \textbf{(4-152)}$$

ist eine unabhängige Knotenmenge von G, weil dies bereits für alle $i_{opt,j}$ in ihren jeweiligen Subgraphen $G_1 \ldots G_j \ldots G_n$ gilt und die Vereinigung i wiederum eine unabhängige Knotenmenge ist. Es bleibt zu zeigen, daß das komplette Austauschen einer $i_{opt,j}$ gegen eine andere unabhängige Knotenmenge von G_j keine Verbesserung der Bewertung bringen kann. Die Bewertung GO von i ist implizit eine Funktion des Graphen und lautet per Definition:

$$\begin{aligned} GO(i) &= \left(\bigotimes_{v \in i} t(v) \right) \otimes \left(\bigotimes_{v \in V - i} f(v) \right) \\ &= \left(\bigotimes_{v \in i - i_j} t(v) \right) \otimes \left(\bigotimes_{v \in (V - V_j) - i} f(v) \right) \otimes \left(\bigotimes_{v \in i_j} t(v) \right) \otimes \left(\bigotimes_{v \in V_j - i} f(v) \right) \\ &= \left(\bigotimes_{v \in i - i_j} t(v) \right) \otimes \left(\bigotimes_{v \in (V - V_j) - i} f(v) \right) \otimes GO(i_j) \\ &= \bigotimes_{j=1}^{n} GO(i_j) \end{aligned} \qquad \textbf{(4-153)}$$

(⊗ ist ein wie in OY definierter Operator.)
Eine Verschlechterung der Bewertung $GO(i_j)$ eines Teilgraphens bewirkt also immer auch eine Verschlechterung von $GO(i)$.
■

Die Grundidee der hier verwendeten branch and bound-Variante A* ist folgende: Man partitioniere den Raum aller potentiellen Lösungen. Für jeden Teil dieser Zerlegung bestimme man eine obere und eine untere Schranke für die Bewertung der enthaltenen Lösungskandidaten. Man streiche alle Teile des Lösungsraums, die aufgrund ihrer Schranken keine optimale Lösung enthalten können. Die Teilung des Lösungsraumes wird stets bei demjenigen Teil fortgesetzt, dessen optimistische Schranke am besten ist.



Beispiel A*. Die Menge aller Lösungen {a...i} wird disjunkt zerlegt. Für jede Teilmenge wird eine obere und untere Schranke des Gütemaßes bestimmt (als Klammer in der Abbildung dargestellt). Nach der ersten Teilung ist klar, daß {g,h,i} keine Lösung enthält, weil die untere Schranke von {a,b,d,e} bereits besser als die obere Schranke von {g,h,i} ist. D. h. der Raum {g,h,i} muß nicht weiter geteilt werden. Nun wird der Raum mit der besten oberen Schranke geteilt ({a,b,d,e} → {b,e} und {a,d}) und die Schranken werden bestimmt. {c,f} kann damit auch gelöscht werden. Nach der Verzweigung von {b,e} ist bereits eine optimale Lösung gefunden. Alle noch vorhandenen Teilräume {e} und {a,d} haben schlechte obere Schranken als der nun bekannte Bewertung von {b}. Die obere und untere Schranke fallen bei einer einzelnen Lösung zusammen.
Durch den A* Algorithmus werden nur die beiden Lösungen b und e explizit generiert. Trotzdem ist man sicher, die optimale Lösung (nämlich b) gefunden zu haben.

4.3.1 Algorithmus Interpretation

Es werden wegen Satz 10 nur Instanzen $(G(V,E),\otimes,t,f)$ von OY betrachtet, deren Graph G zusammenhängend ist.
Eine unabhängige Knotenmenge des Graphen wird in diesem Abschnitt naheliegend mit Interpretationsfragment, kurz Fragment, bezeichnet. Durch die Bezeichnung Fragment wird angedeutet, daß eine solche Knotenmenge nicht die Maximalitätsforderung einer Interpretation erfüllen muß. Es werden zwei Datenstrukturen *Zustand* und *Liste<Zustand>* benötigt:

Zustand
Eine Datenstruktur *Zustand* besteht aus zwei Mengen von Verweisen auf Knoten. Er beschreibt eine Partitionierung der Knotenmenge V in drei Teile. Der sogenannte *inInter* Teil ist die Menge von Knoten, die zur durch die *Zustand*-Instanz repräsentierten Fragmentenmenge gehören. Der Teil *notInter* ist die Menge von Knoten, die definitiv nicht zu den Fragmenten gehören. Die verbleibenden Knoten, die sich also weder in *inInter* noch in *notInter* befinden, repräsentieren Knoten über deren Zugehörigkeit noch keine Aussage getroffen worden ist. Ist diese Restmenge nicht leer, verbleibt eine gewisse Unbestimmtheit. Dann wird also durch einen *Zustand* nicht genau ein Fragment sondern implizit eine Menge von Fragmenten beschrieben.

$$\begin{aligned} &Zustand := 2^V \times 2^V \\ &z \in Zustand \\ &z = (inInter, notInter) \\ &inInter \cap notInter = \varnothing \end{aligned} \qquad \textbf{(4-154)}$$

Damit die Unabhängigkeitseigenschaft gewährleistet wird, enthält *inInter* keine durch Kanten verbundenen Knoten:

$$inInter^2 \cap E = \varnothing \qquad \textbf{(4-155)}$$

Die Menge *notInter* enthält auch immer alle diejenigen Knoten, für die es einen benachbarten Knoten aus *inInter* gibt:

$$\forall v \in V : \exists v' \in inInter \wedge (v', v) \in E \quad \Rightarrow \quad v \in notInter \qquad \textbf{(4-156)}$$

Man kann sich die *Zustände* als einen Baum veranschulichen. Die Wurzel des Baumes ist ein *Zustand zWurzel*=(∅,∅), welcher die Menge aller Fragmente kodiert. Die Wurzel und alle folgenden Zustände werden immer weiter verzweigt bis an den Blättern nur Zustände existieren, für die $inInter_{zBlatt} \cup notInter_{zBlatt} = V$ gilt. Die Blattknoten kodieren also jeweils nur noch genau ein Fragment, d. h. jeweils eine unabhängige Knotenmenge.

Die Verzweigung eines *Zustand*es *z* erfolgt, indem alle *Zustände z´* generiert werden, die zu $inInter_z$ einen noch möglichen Knoten aus der Restmenge V-($inInter_z \cup notInter_z$) hinzufügen. Für den Zustand z‘ sei das der Knoten *v*. $notInter_{z'}$ wird so gebildet, daß alle *Zustände z‘* einen dijunkten Lösungsraum beschreiben. Anschließend wird *z´* bereinigt, indem alle zu *v* benachbarten Knoten zu $notInter_{z'}$ hinzugefügt werden, weil diese niemals gleichzeitig mit *v* in einer unabhängigen Knotenmenge sein können (4-156). Die formale Fassung der Verzweigungsprozedur wird weiter unten beschrieben.

In der Abb. 4-37 wird der *Zustand z‘*=({*c*},{*a,b,d,f*}) auf folgende Weise ermittelt (siehe Algorithmus Interpretation):
inInter ergibt sich durch Vereinigung der leeren Menge *inInter* des Vorgänger*zustand*s mit dem in diesem Teilungsschritt noch nicht betrachteten Objekt {*c*}. *notInter* ergibt sich zunächst auch aus der leeren Menge *notInter* des Vorgänger*zustand*s. Diese wird um die in diesem Teilungsschritt bereits betrachteten Knoten {*a,b*} erweitert. Dadurch wird sichergestellt, daß *z‘* kein Fragment kodiert, das bereits durch die gerade erzeugten *Zustände* ({*a*},{*b,d,e*}) und ({*b*},{*a,c,f*}) kodiert wird. Zu $notInter_{z'}$ wird außerdem der Knoten *f* hinzugefügt, weil er zu *c* eine Kante aufweist. *f* kann nie gleichzeitig mit *c* zu einem Fragment gehören (Bereinigung). ({*c*},{*a,b,d,f*}) kodiert somit implizit die Fragmente {*c*} und {*c,e*}.

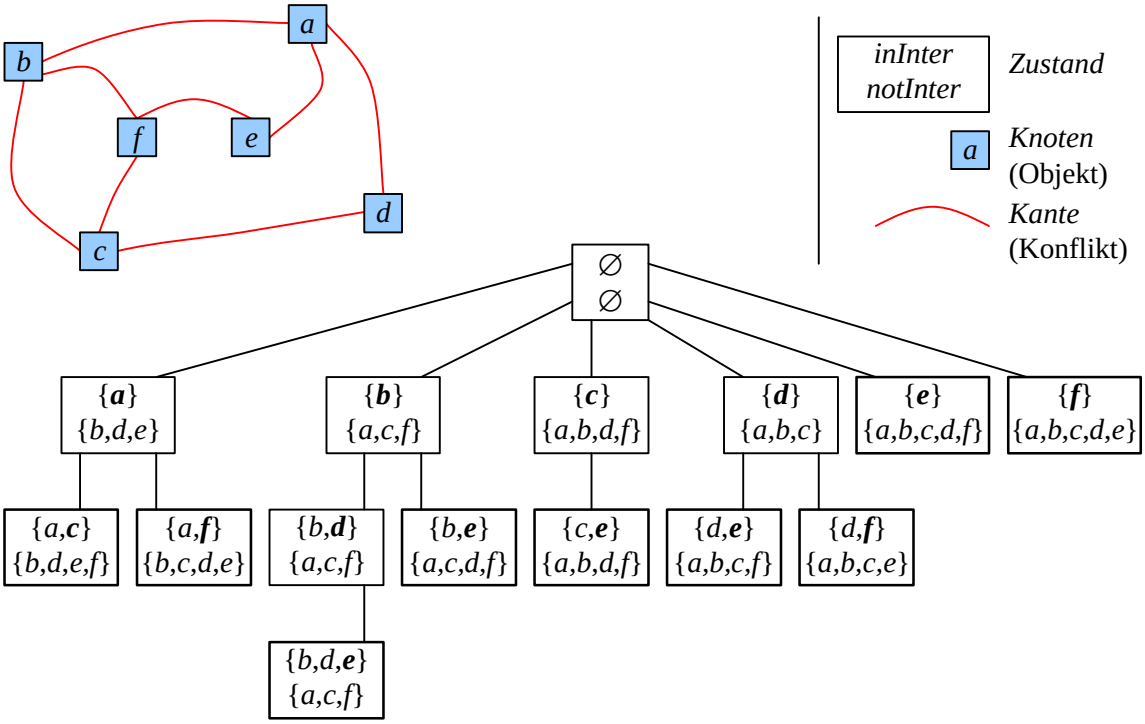


Abb. 4-37 Ein Graph und der zugehörige vollständige Zustandsbaum. Der *Zustand* ({a},{b,d,e}) kodiert indirekt die Menge {{a,c},{a,f},{a}} unabhängiger Knotenmengen.

Liste<Zustand>
Eine Liste von *Zustände*n beinhaltet alle zu einem bestimmten Zeitpunkt zwar generierten aber noch nicht verzweigten *Zustände*. Der Algorithmus benötigt nur eine Instanz dieser Datenstruktur.

Bewertung eines *Zustand*s
Die nicht durch *inInter* und *notInter* abgedeckten Knoten eines *Zustand*es *z* werden durch eine Funktion

$$rest(z) := V - (inInter_z \cup notInter_z) \qquad \textbf{(4-157)}$$

ermittelt. Für Zustände wird ein Prädikat vereinbart:

$$blattzustand(z) := \begin{cases} wenn\ rest(z) = \varnothing & true \\ sonst & false. \end{cases} \qquad \textbf{(4-158)}$$

Die Mengen *inInter* und *notInter* beschreiben den bereits feststehenden Teil eines Fragments (d. h. einer potentiellen Interpretation). Die beiden Mengen können genutzt werden, um ein partielle Bewertung entsprechend *GO* vorzunehmen. Diese Bewertung ist eine Funktion g : *Zustand*$\rightarrow$ R und wird wie folgt definiert:

$$g(z) := \begin{cases} wenn\ V - rest(z) \neq \varnothing & \left(\bigotimes_{v \in inInter_z} t(v) \right) \otimes \left(\bigotimes_{v \in notInter_z} f(v) \right). \\ sonst & 1 \end{cases} \qquad \textbf{(4-159)}$$

$\otimes$ ist ein wie in OY definierter Operator. Das neutrale Element von $\otimes$ ist vereinbarungsgemäß 1. Vier weitere Funktionen h_{opt}, h_{pess}, f_{opt}, f_{pess} : *Zustand*$\rightarrow$R dienen der Bestimmung einer oberen und einer unteren Bewertungsschranke für die durch den *Zustand* kodierte Menge an Fragmenten. Diese Funktionen bewerten die Knoten der Restmenge:

$$h_{opt}(z) := \begin{cases} wenn\ rest(z) \neq \varnothing & \bigotimes_{v \in rest(z)} \max(t(v), f(v)) = \bigotimes_{v \in rest(z)} t(v) \\ sonst & 1 \end{cases}$$
$$h_{pess}(z) := \begin{cases} wenn\ rest(z) \neq \varnothing & \bigotimes_{v \in rest(z)} \min(t(v), f(v)) = \bigotimes_{v \in rest(z)} f(v) \\ sonst & 1 \end{cases} \qquad \textbf{(4-160)}$$

$$f_{opt}(z) := g(z) \otimes h_{opt}(z)$$
$$f_{pess}(z) := g(z) \otimes h_{pess}(z). \qquad \textbf{(4-161)}$$

Satz 11. *Für alle (auch indirekten) Nachfolger z´ eines Zustands z gilt:*

$$f_{pess}(z) \leq f_{pess}(z') \leq f_{opt}(z') \leq f_{opt}(z). \qquad \textbf{(4-162)}$$

Für alle in z kodierten BlattZustände zBlatt gilt:

$$f_{pess}(z) \leq f_{pess}(zBlatt) = GO(inInter_{zBlatt}) = f_{opt}(zBlatt) \leq f_{opt}(z). \qquad \textbf{(4-163)}$$

Beweis. (4-162) gilt aufgrund der Monotonie der Multiplikation $\otimes$ von Elementen aus R und der Tatsache, daß für alle Knoten $f(v) < t(v)$ gilt.
Für einen Blatt*Zustand zBlatt* gilt außerdem: *rest*(*zBlatt*) = $\varnothing$. Daraus folgt

$$\begin{aligned} f_{pess}(zBlatt) &= g(zBlatt) \otimes 1 \\ &= \left(\bigotimes_{v \in inInter_{zBlatt}} t(v) \right) \otimes \left(\bigotimes_{v \in notInter_{zBlatt}} f(v) \right) \\ &= GO(inInter_{zBlatt}). \end{aligned} \qquad \textbf{(4-164)}$$

Entsprechendes gilt für $f_{opt}(zBlatt)$.
■

Anschaulich bedeutet das: Besser (d. h. größer) als $f_{opt}(z)$ kann kein durch z kodiertes Fragment werden und $f_{pess}(z)$ ist immer mindestens genauso schlecht das schlechteste durch z kodierte Fragment. Daher müssen nicht immer alle Zustände verzweigt werden. Ein Knoten z zu dem man einen Knoten z' findet, dessen $f_{pess}(z')$ besser (größer, gleich) als $f_{opt}(z)$ ist, muß nicht weiter untersucht, d. h. verzweigt werden.
Nun kann der Algorithmus zur Generierung einer optimalen Interpretation angegeben werden. In der Praxis wird der Definitionsbereich R durch]0,1[, der Operator $\otimes$ durch die gewöhnliche Multiplikation $a \otimes b \rightarrow a*b$ oder durch die Addition von Logarithmen $a \otimes b \rightarrow \log(a)+\log(b)$ ersetzt.

Algorithmus 11 Interpretation.

In: Instanz $(G=(V,E),t,f)$ von OY mit zusammenhängenden Graph G **Out:** i_{opt}

$schlange := Liste < Zustand > (\emptyset)$
$einsortiere(schlange, Zustand(\emptyset,\emptyset))$
$z_{opt} := Zustand(\emptyset,\emptyset)$ **(4-165)**
$fGlobal_{pess} := 0$

wiederhole solange $nichtLeer(schlange)$
  $z := holeMax(schlange)$
  $ausschluss := \emptyset$
  $\forall v \in rest(z)):$
    $z' := Zustand(inInter_z \cup \{v\}, notInter_z \cup ausschluss)$ **(4-166)**
    $ausschluss := ausschluss \cup \{v\}$
    $Bereinige(z', v)$
    $Bewerte(z')$
return $inInter_{z_{opt}}$

Die Funktion *bewerte*(.) berechnet für den Zustand die Schranken und modifiziert den Wert $fGlobal_{pess}$. Dies ist der Wert, der durch einen der bereits geöffneten Knoten auf jeden Fall erreicht werden kann.

Algorithmus 12 Bewerte.

In: *Zustand z* **In/Out:** *schlange,* $fGlobal_{pess}$, z_{opt}

$fGlobal_{pess} := Max(fGlobal_{pess}, f_{pess}(z))$
wenn $blattzustand(z)$
  wenn $f_{opt}(z_{opt}) < f_{opt}(z)$
    $z_{opt} := z$
    $löscheKleinere(schlange, f_{opt}(z_{opt}))$ **(4-167)**
sonst
  $löscheKleinere(schlange, fGlobal_{pess})$
  $einsortiere(schlange, z)$

Die Funktion *einsortiere*(*schlange*, *z*) sortiert den *Zustand z* entsprechend seiner Bewertung $f_{opt}(z)$ in die *schlange* ein. Entsprechend löst die Funktion *holeMax*(*schlange*) den *Zustand* mit der größten Bewertung f_{opt} aus der *schlange* heraus.
Mittels der Funktion *löscheKleinere*(*schlange*,*wert*) werden alle *Zustände z* aus der sortierten *schlange* entfernt, deren $f_{opt}(z)$ kleiner als der übergebene Wert ist. Dies kann aus zwei Gründen geschehen: Zum einen, wenn ein neuer bester *Zustand* z_{opt} gefunden wurde (mit dem *wert* = $f_{opt}(z_{opt})$), oder aber wenn der schlechteste auf jeden Fall erreichbare *Zustand* (mit dem *wert* = $fGlobal_{pess}$) größer ist als die Bewertung $f_{opt}(z)$.
Die Funktion *Bereinige*(*z*,*v*) stellt die geforderte Konsistenz des Zustands *z* her, indem alle zum betrachteten Knoten *v* in Konflikt stehenden Knoten *notInter* hinzugefügt werden.

Algorithmus 13 Bereinige.

In: *Knoten v*	**In/Out:** *Zustand z*
$\forall v' \in rest(z):$ *wenn* $(v', v) \in E$ $notInter_z := notInter_z \cup \{v'\}$	**(4-168)**

In einer realen Implementierung des Algorithmus Interpretation wird man noch folgende Optimierungen vornehmen:

- die Initialisierung von z_{opt} erfolgt nicht mit dem Wurzel*Zustand* (∅,∅), sondern mit irgendeinem einem Blatt*Zustand*, der eine unabhängige, dominierende Knotenmenge *inInter* enthält. Einen solchen Blatt*Zustand* findet man leicht durch eine Tiefensuche. Dem entsprechend wird $fGlobal_{pess}$ mit $f_{pess}(z_{opt})$ initialisiert. Dadurch ist bereits von Anfang an eine wesentlich bessere Schranke als 0 im Einsatz.
- Ein branch and bound Verfahren ist im schlechtesten Fall exponentiell. Um ein Terminieren des Verfahrens auch in solchen pathologischen Fällen zu erzwingen, sollte eine sinnvolle Zeitbegrenzung implementiert werden. Bei Überschreitung dieser Grenze wird die bis dahin gefundene beste Lösung z_{opt} ausgegeben. Diese Lösung kann als Näherungslösung interpretiert werden. Ein sehr langsames Konvergieren des Verfahrens deutet darauf hin, daß es viele sehr ähnlich bewertete Interpretationen der Vorlage gibt, die Qualität der Lösungen sich also nicht sehr unterscheiden. Damit kann irgendeine eine Näherungslösung als Repräsentant der Menge dieser guten Lösungen akzeptiert werden.
  Beispiel. Wenn eine Zeichnung eine sehr lange Zeichenkette „abcdefgh...“ enthält und alle Wörter als gleichwahrscheinlich betrachtet werden, ergibt sich auch für alle Interpretationen (beliebige Wortkombinationen {„a“, „bcdefgh...“},{ „a“, „b“, „cdefgh...“} usw.usf.) ein gleichgroßes Gütemaß *GO*. Es existiert eine große Menge von Interpretationen dieses Teils der Zeichnung, die durch den Algorithmus vollständig durchmustert werden muß. Da es offenbar keine ausgezeichneten Wörter gibt, ist jede beliebige Interpretation gleichermaßen korrekt. Der Algorithmus sollte sich anderen zusammenhängenden Konfliktgraphen, d. h. anderen Teilen der Zeichnung zuwenden.
- In jedem *Zustand z* wird $h_{opt}(z)$, $h_{pess}(z)$, $f_{opt}(z)$, $f_{pess}(z)$ und $g(z)$ abgespeichert. Dadurch können die Bewertungen der neuen aus *z* durch Verzweigung entstehenden *Zustände* durch Nutzung der gespeicherten Werte von *z* bestimmt werden.
- Man sollte mit logarithmierten Werten für die Knotenbewertung $f(v)$ und $t(v)$ rechnen, wenn es sich bei OY um OYMULT handelt, und so die Gleitkommamultiplikation auf eine schnelle Addition zurückführen.
- Die Reihenfolge in der die Knoten aus der Knotenmenge *V* im Algorithmus mittels $\forall v \in rest(z)$ herausgelöst werden, ist nicht unwesentlich. Durch ein Vorsortieren der Menge *V* entsprechend der Bewertung $f(v)$ wird das hier verwendete best-first-search unterstützt. Die Suche erfolgt tendenziell gleich in der Nähe der Blätter des *Zustand*baumes, d. h.

Blätter werden rasch und gute, d. h. restriktive Schranken werden gleich am Anfang der Suche gefunden.

Satz 12. *Der Algorithmus Interpretation löst die Probleme* OY *und* OYMULT.

Beweis. Da OYMULT eine Instanz von OY ist, muß nur OY betrachtet werden:
(1) Das Terminieren des Algorithmus kann leicht gezeigt werden: Jeder Durchlauf der Verzweigung, d. h. der beiden äußeren Schleifen, der zur Generierung eines Blattzustandes (*rest*(*z´*)=∅) führt, reduziert die *schlange* um einen Eintrag. Neue Einträge in die *schlange* entstehen durch die nur endliche Bildung unabhängiger Knotenmengen *V*. Damit wird spätestens nach Aufzählung aller unabhängigen Teilmengen von *V* ein Ende des Verfahrens erreicht.

(2) Es ist zu zeigen, daß nur unabhängige Knotenmengen als *blattzustand* generiert werden.
Da für jeden Knoten *v*, der *inInter* hinzugefügt wird, die Menge von Knoten, die zu *v* eine Kante aufweisen, in *notInter* gespeichert werden (Funktion *Bereinige*()), werden nur unabhängige Knotenmengen als *blattknoten* generiert.

(3) Nachzuweisen ist, daß alle unabhängigen Knotenmengen als *blattzustand* durch den Algorithmus erzeugt werden, wenn keine *Zustände* aus der *schlange* gelöscht werden würden.
Jeder Zustand $z=(inInter_z, notInter_z)$ wird offenbar solange verzweigt, bis die Menge seiner Blätter aus allen unabhängigen Knotenmengen besteht, die $inInter_z$ als Teilmenge enthalten.

(4) Weiterhin ist zu zeigen, daß im Laufe der Abarbeitung des Algorithmus kein *Zustand* aus der *schlange* gelöscht wird, der eine Lösung von OY kodiert, es sei denn z_{opt} enthält bereits eine Lösung:
Mittels der Funktion *löscheKleinere*(*schlange*,*wert*) werden alle Zustände *z* aus der *schlange* entfernt, deren $f_{opt}(z)$ kleiner als der übergebene Wert ist. Dies kann aus zwei Gründen geschehen:

- Zum einen, wenn ein neuer bester *Zustand* z_{opt} gefunden wurde,
- oder aber wenn der schlechteste auf jeden Fall erreichbare *Zustand* (mit dem Wert $fGlobal_{pess}$) größer ist als die Bewertung $f_{opt}(z)$.

z_{opt} ist immer besser als alle aus der *schlange* gelöschten Zustände.
Die zweite Aufrufvariante *löscheKleinere*(*schlange*, $fGlobal_{pess}$) setzt berechtigter Weise voraus, daß jeder *Zustand* zu einem Blattknoten erweitert werden kann. Nach Satz 11 können auch Knoten, die keine Blattknoten sind, gelöscht werden wenn ihre optimistische Schranke schlechter als ein bereits gefundenes f_{pess} ist, denn auch ihre Nachfolger werden nicht besser werden.

(5) Letztendlich muß untersucht werden, ob der zum Schluß ausgegebene *Zustand* z_{opt} (genauer: dessen Knotenmenge *inInter*) tatsächlich die unabhängige Teilmenge i_{opt} = *inInter* mit maximaler Bewertung $GO(i_{opt})$ darstellt und *inIter* kardinalitätsmaximal ist:
Da nach Satz 6 zu jeder unabhängigen, nicht kardinaliätsmaximale Knotenmenge *m* eine kardinalitätsmaximale unabhängige Knotenmenge existiert, die *m* als Teilmenge enthält und ein besseres *GO* aufweist, können die *m* nicht zu maximalen *GO* führen. Ein optimaler Zustand repräsentiert stets auch eine kardinaliätsmaximale Knotenmenge.
Da beim Terminieren des Algorithmus die *schlange* leer ist, also keine teilungsfähigen *Zustände* mehr existieren, gibt es nach (4) auch keine besseren *Zustände* als z_{opt}.
■

Nach Satz 12 und Satz 6 erzeugt der Algorithmus Interpretation eine optimale Interpretation einer technischen Zeichnung im Sinne der Definition 29.

4.3.2 Komplexitätsbetrachtung

Die Zeitkomplexität des zugrunde liegenden kombinatorischen Optimierungsproblems ist von Torsten Wierschin in seiner Diplomarbeit [Wierschin 2000] untersucht worden. Die

Berechnungen, die einen *Zustand* und dessen Verzweigung betreffen, sind polynomiell bezüglich der Knotenanzahl $|V|$ und sollen hier vernachlässigt werden. Somit ist nur die Anzahl der *Zustände* im *Zustand*sbaum von Belang. Sie ist beim kombinatorischen Optimierungsproblem gleich der Anzahl der unabhängigen Knotenmengen $\mathcal{I}(G=(V,E))$, weil die Reduktion durch die Bewertung der *Zustände* nicht betrachtet wird. Die Extrema der Zeitkomplexität liegen bei den extremen Graphen: Für den leeren Graphen gibt es $\mathcal{I}(G=(V, \varnothing)) = 2^{|V|}$ solcher unabhängigen Mengen. Der vollständige Graph enthält $\mathcal{I}(G=(V, V\times V)) = |V|+1$ unabhängige Mengen. Für einen zufälligen Graphen $G_{\boldsymbol{random}}$ – mit einer Wahrscheinlichkeit von ½ für jede Kante – ergibt sich für die Anzahl unabhängiger Knotenmengen folgender Erwartungswert:

$$\mathcal{I}(G_{random}) = \sum_{k=0}^{|V|} \binom{|V|}{k} 2^{-\binom{k}{2}}. \qquad \textbf{(4-169)}$$

Die Abschätzung der Ordnung der mittleren Zeitkomplexität ist subexponentiell und beträgt:

$$\mathcal{I}(G_{random}) = O\left(|V|^{\log(|V|)}\right). \qquad \textbf{(4-170)}$$

Diese Abschätzung ist eher pessimistisch. Sie setzt zwar einen durchschnittlichen Graphen voraus aber sie entspricht bezüglich einer Bewertung *GO* dem schlechtesten Fall, in welchem *alle Zustände* bis hin zu den Blättern verzweigt werden müssen.

4.4 Der Algorithmus Y

Der Algorithmus zur automatischen Analyse einer technischen Zeichnung unter Nutzung des Models *M* einer Domäne lautet:

Algorithmus 14 Y.

In: M, $O=O_{primitive}$,,Konflikte in $O_{primitive}$, konkretes GO_{method}	**Out:** i_{opt}
$C := Objektgenerierung(O_{primitive})$ $conflict := Konfliktgenerierung(O_{primitive}, conflict\ in\ O_{primitive})$ $i_{opt} := \varnothing$ $\forall\ bzgl.\ conflict\ zusammenhängenden\ Teilmengen\ cc\ von\ O_{out}:$ $\quad Bilde\ Graph\ G(cc, conflict\ in\ cc)$ $\quad i_{opt} := i_{opt} \cup Interpretation((G, t_{method}, f_{method}))$ $return\ i_{opt}$	**(4-171)**

Perfection is achieved
only on the point of collapse.

C.N.Parkinson

5 Lernen der Wahrscheinlichkeiten

Für die „Objektgenerierung“ und auch für die hier vorgeschlagenen Versionen des globalen Optimalitätskriteriums GO_{simple} und GO_{norm} im Algorithmus „Interpretation“ werden die bedingten Wahrscheinlichkeiten P_c aller Klassen $c \in C$ benötigt. Der Wertebereich der Funktion $const_c$ ist der Definitionsbereich von P_c. Dieser Wertebereich wird in jeder Dimension geeignet durch eine unstetige Funktion *discr* diskretisiert und begrenzt. Es entsteht so ein endliches Feld mit ebenso vielen Dimensionen, wie dieser Wertebereich aufweist. Die Begrenzung des Feldes soll so erfolgen, daß sich die Wahrscheinlichkeit an den Grenzen des Feldes nicht mehr ändert.
Der Eintrag an einer Position

$$index(o) := discr(const_{c_o}(p_o)) \qquad \textbf{(5-172)}$$

dieses Feldes sei ein Paar positiver natürlicher Zahlen $(positive, all)_{index(o)}$. Diese Zahlen repräsentieren Häufigkeiten:

- *all*: Häufigkeit des Auftretens der vorliegenden signifikanten Merkmale $const_{c_o}(p_o)$,
- *positive*: Anzahl der Objekte der Klasse *c* bei den vorliegenden signifikanten Merkmalen.

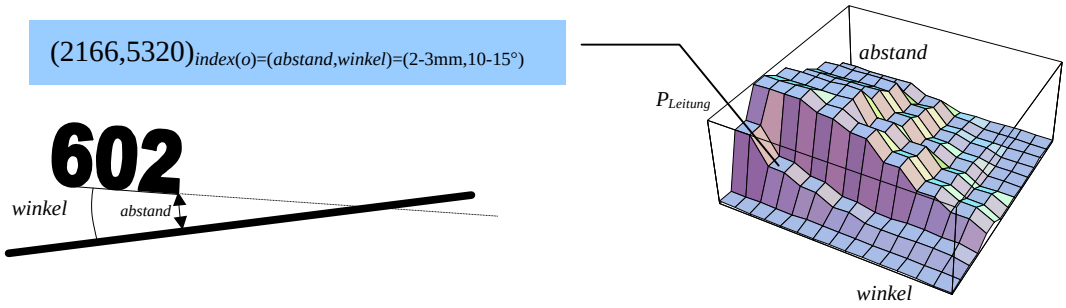


Abb. 5-38 Ein Objekt *o* mit c_o = *Leitung* hat eine bestimmte Konfiguration signifikanter Merkmale und eine bedingte Wkt. von ca. 0.4 (2166/5320)

Wurde eine repräsentative Stichprobe von Zeichnungen (wie weiter unten dargestellt) analysiert, gilt folgendes:

$$P_o = P(o \mid index(o)) = \frac{positive_{index(o)}}{all_{index(o)}}. \qquad \textbf{(5-173)}$$

Nachdem alle eine Domäne beschreibenden Klassen entwickelt wurden und bevor die automatische Analyse von Vorlagen beginnen kann, müssen sämtliche Feldeinträge (*positiv*, *gesamt*) aller Klassen bestimmt werden. Dazu wird das folgende einfache, überwachte Lernverfahren vorgeschlagen:

1. Zuerst erfolgt eine **Initialisierung** aller Einträge der Felder durch einen Experten mit subjektiven Wahrscheinlichkeiten. Für einen bestimmten Eintrag sei diese Wahrscheinlichkeit *p* ($0<p<1$). Diese wird auf (*positive*, *all*) aufgeteilt. *all* ist frei wählbar und bestimmt die Wichtung dieses Initialwertes. *positive* ergibt sich aus dem Produkt *all***p*.
2. Anschließend wird eine **Stichprobe** von Zeichnungen der Reihe nach dem System vorgelegt und der Algorithmus „Objekterzeugung“ für alle Klassen und die „Interpretation“ ausgeführt. Jede Zeichnung führt zu einer Veränderung der Feldeinträge. Dabei muß der Operateur nur bei Fehlleistungen eingreifen und irrtümlich erzeugte oder nicht erzeugte Objekte benennen. Die Modifikation der Einträge erfolgt so:

Objekt *o* (indirekt) in einem Objekt aus der Interpretation i_{opt} enthalten	*o* existiert nach Experten-meinung	Änderung des Feldeintrages	
ja	ja	automatisch	$positive_{index(o)} := positive_{index(o)}+1$ $all_{index(o)} := all_{index(o)}+1$
nein	ja	manuell	$positive_{index(o)} := positive_{index(o)}+1$ $all_{index(o)} := all_{index(o)}+1$
ja	nein	manuell	$all_{index(o)} := all_{index(o)}+1$
nein	nein	automatisch	$all_{index(o)} := all_{index(o)}+1$

Durch dieses Verfahren werden die bedingten Häufigkeiten bestimmt. Ist die Stichprobe repräsentativ, so können die so ermittelten Quotienten *positive*/*all* als bedingte Wahrscheinlichkeiten interpretiert und bei der automatischen Zeichnungsanalyse verwendet werden. Die Quotienten werden stets im offenen Intervall]0,1[liegen und somit die Anforderungen eines *GO*s erfüllen.
Die Schwellwerte sind für die so erzielten Wahrscheinlichkeiten offensichtlich ohne Belang. Sie können während der Lernphase stets so eingestellt werden, daß der Operateur möglichst wenig eingreifen muß.
Theoretisch kann auf eine Initialisierung mit subjektiven Wahrscheinlichkeiten verzichtet werden. Legte man statt dessen alle Einträge auf ½ fest, würde das Lernverfahren gegen die selben Wahrscheinlichkeiten konvergieren. Ein solches auf den ersten Blick bequemes Vorgehen hat jedoch zwei entscheidende Nachteile:

- Der Operateur muß in der Anfangsphase des Lernens sehr häufig eingreifen, weil sehr viele fehlerhafte Klassenzuordnungen auftreten werden.
- Die Optimierung im Algorithmus „Interpretation“ kann dann in der Anfangsphase des Lernens leicht zu einem exponentiellen Zeitbedarf führen. Die den Suchraum einschränkende Wirkung von *GO* wird nicht genutzt. Alle Interpretation haben ähnliche *GO*-Werte. Diesen Nachteil könnte man ausschließen, würde man in der Lernphase nur die „Objektgenerierung“ nicht aber die „Interpretation“ durchführen. Die Entlastung des Operateurs durch die selektive Wirkung der Optimierung würde dadurch allerdings auch entfallen [Messmer Bunke 1996], [Thrun 1995]).

Hallo World!

Folklore

6 Implementierung und Ergebnisse

Die hier verwendeten Algorithmen sind ihrem Zeitverhalten nicht immer angenehm. So ist das Problem der Suche einer optimalen Interpretation NP-hart, was in ungünstigen Fällen zu einem exponentiellen Zeitbedarf des zugehörigen Algorithmus führt. Auch die in der Modellierungsphase zu entwerfenden Objektgeneratoren werden nicht immer trivial sein und so können sie – auch wenn sie durch die Metaklassen nur polynomiell oft aufgerufen werden – einige Ressourcen binden. Aus diesen Gründen seien einige grundlegende Bemerkungen zur Kodierung von Y in realer Software gestattet.

Für die Implementierung des hier vorgestellten Modellierungskonzeptes gibt es prinzipiell zwei Möglichkeiten:

1. Entweder man definiert eine eigene, eventuell graphische Beschreibungsprache und implementiert einen dazugehörigen Compiler oder Interpreter oder
2. man läßt die Modelle direkt in der Hochsprache, in der auch das Analysesystem (hier C++) programmiert ist, entwickeln.

Da moderne Programmiersprachen über sehr ausgereifte Mechanismen zum Kapseln von Funktionalität verfügen und sich auch sonst sehr an eine spezialisierte Aufgabenstellung anpassen lassen, ist der zweite Ansatz zu favorisieren. Man erhält die volle Leistungsfähigkeit professioneller Compiler und die scharfsinnigen Sprachmittel auch für die Anwendung der Modelle während der Analyse einer großen komplexen Zeichnung. Beispielsweise kann in der Implementierung eines Objektgenerators ein komplizierter und aufwendiger Vorgang wie Graphenmatching erfolgen. Auch können die Parametertypen der Klassen entsprechend der Hochsprache völlig frei gewählt werden (z. B. kann in einem Objektgenerator ein Graph an ein Objekt der Nachfolgerklasse als Parameter übergeben werden). Die Nachteile dieser Herangehensweise sind, daß der Modellierende zumindest in den Grundzügen die jeweilige Programmiersprache beherrschen muß und eine Modelländerung immer zu einem zusätzlichen Übersetzungs- und Linkaufwand führt. Hat sich später ein bestimmter Kern an Klassen und Metaklassen herauskristallisiert, sollte dieses Problem neu überdacht werden. Für eine eingeschränkte Menge von verwandten Domänen (z. B. topographische Karten) kann es durchaus sinnvoll sein, eine spezialisierte und sehr einfache Modellierungssprache zu entwerfen.

Wenn man das System Y als Blackbox betrachtet, ergibt sich der folgende Datenfluß bei der automatischen Analyse einer technischen Zeichnung:

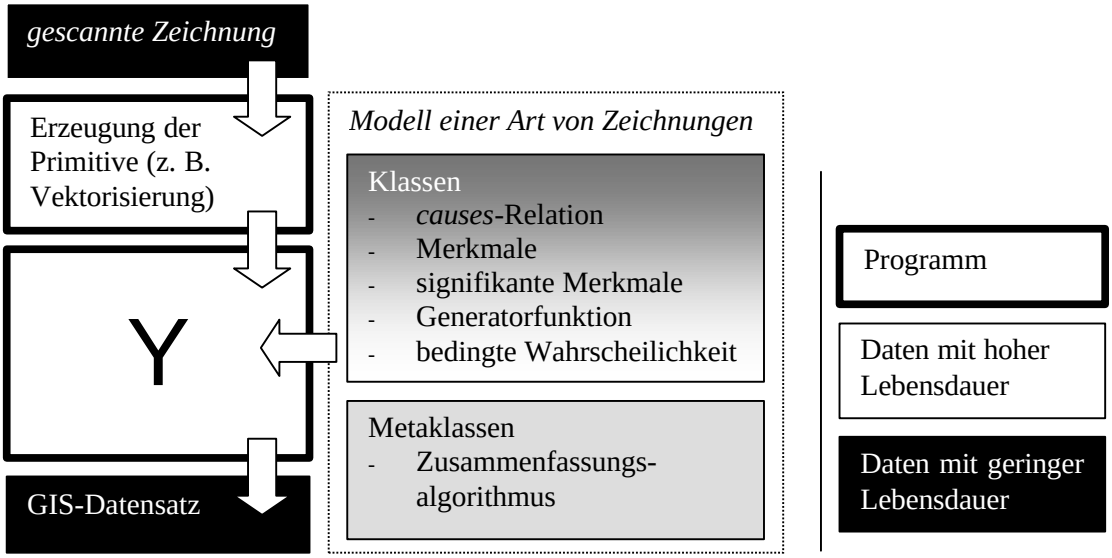


Abb. 6-39 Grobe Systemarchitektur mit Datenfluß

Die Häufigkeit der Änderung der Systemkomponenten wird durch die Helligkeit der jeweiligen Box symbolisiert. Während die Implementierung einer neuen Metaklassen nur selten erforderlich ist, kann aufgrund der hohen Spezialisierung der Modelle eine häufige Anpassung einiger Klassen erforderlich sein. Aber zum Glück bleiben auch viele Klassen, wie etwa die Modelle geometrischer Grundbausteine (*Linie*n, *Zeichenkette*n) unverändert, auch wenn neue Zeichnungsarten interpretiert werden müssen. Diese wichtige Bemerkung zur Skalierbarkeit des Ansatzes soll am folgenden Beispiel erläutert werden:

Beispiel Gasleitungen. In einer Art von Zeichnungen seien Leitungen mit Beschriftungen dargestellt. Die Beschriftungen bestehen ausschließlich aus Zahlen. Eine Leitung kann nicht unterschiedliche Beschriftungen aufweisen.

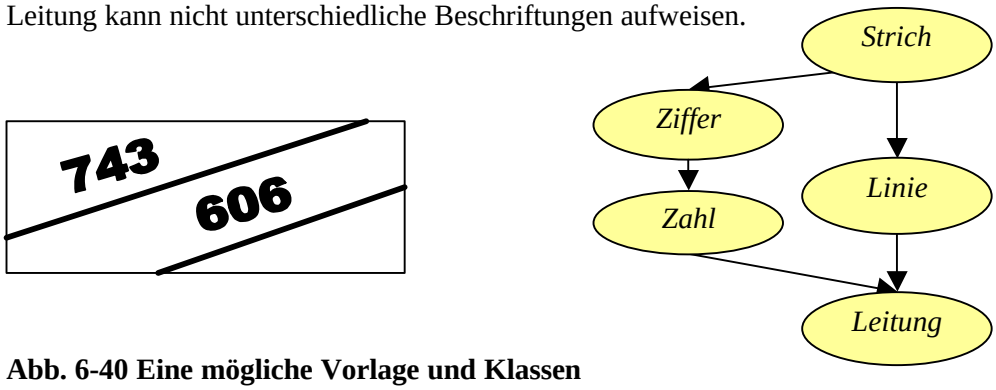


Abb. 6-40 Eine mögliche Vorlage und Klassen

Man kann dem Gesamtmodell Klassen hinzufügen, indem man die Klassen anpaßt, die über die *causes* Relation mit der neuen Klasse verbunden werden müssen. Weiterhin müssen die Wahrscheinlichkeiten P_C neu gelernt werden. Die übrigen Klassen müssen nicht modifiziert werden.

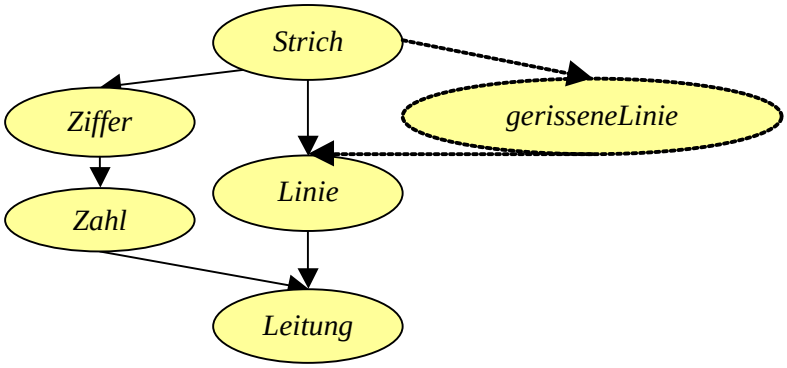


Abb. 6-41 Das graphische Konzept *gerisseneLinie* wird hinzugefügt

Angenommen es wird eine Klasse *gerisseneLinie* dem Gesamtmodell „Gasleitungen“ hinzugefügt. Die Klasse *Linie* muß so modifiziert werden, daß sie eine Generalisierung der Klassen *Strich* und *gerisseneLine* darstellt. Wenn nun die Analyse gestartet wird, erkennt eine robust gestaltete Klasse *gerisseneLinie* in der Zahl 606 irrtümlich zwei Instanzen. Diese konkurrieren mit der sehr wahrscheinlichen Interpretation $\{L_{743}, L_{606}\}$ und werden nicht vom System Y ausgegeben:

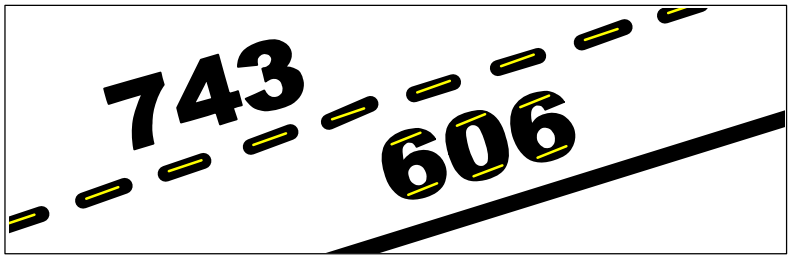


Abb. 6-42 *gerisseneLinie*n in der Zahl 606 können sich global nicht durchsetzen

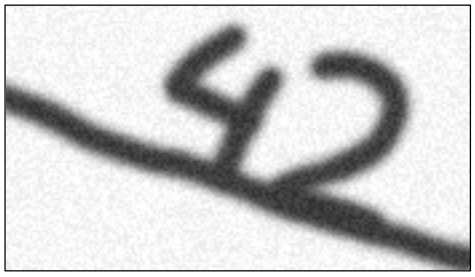
Obwohl die alten Klassen *Ziffer* und *Zahl* mit dem neuen Modell *gerisseneLinie* konkurrieren, müssen sie nicht angepaßt werden. Das System wird durch Hinzufügen oder Ändern von einzelnen Klassen nicht instabil.

Bei anderen Ansätzen [Pasternak 1996], [Kulschewski 1997] besteht eine Dialektik zwischen Fehlern erster und zweiter Art. Die Minimierung beider Fehlerarten ist nicht gleichzeitig möglich.

Fehler erster Art: *Es wird ein Objekt nicht erzeugt, obwohl es eigentlich existiert.*
Fehler zweiter Art: *Es wird ein Objekt erzeugt, obwohl es eigentlich nicht existiert.*

Beim Verfahren Y führen Fehler zweiter Art „nur“ zu einer Laufzeiterhöhung. Die Qualität der Interpretation wird dadurch nicht verringert. Fehler erster Art können prinzipiell auf Kosten von Fehlern zweiter Art minimiert werden.

Die prinzipiellen Nachteile von Y liegen im Bottom-Up-Charakter der Modellierung und damit auch der Analyse begründet. Die so erreichte hohe Flexibilität der Modellierung, geht zu Lasten der Leistung: Einmal nicht erkannte Objekte können auch später, wenn eventuell mehr Kontext und ein starker Verdacht vorliegt, nicht erzeugt werden. Wird die ‚2‘ in nebenstehender Zeichnung nicht durch die Symbolerzeugung der Vektorisierung und damit auch nicht in späteren Analyseschritten von Y erzeugt, kann auch keine vollständige fehlerfreie Interpretation generiert werden. In der vorliegenden Implementierung wird auch die ‚4‘ nicht erzeugt, weil die Symbolerkennung nur isolierte Zeichen erkennt.



Neben der Vektorisierung gescannter Vorlagen kann eine weitere, viel versprechende Anwendung für das hier vorgeschlagene Verfahren gesehen werden. Heute vorliegende digitale Geometiedatenbestände sind oft wenig oder nur für eine Spezialanwendung optimal strukturiert. Die allgemeine Veredlung oder die Anpassung der Daten an eine neue spezielle Aufgabe ist eine Herausforderung der näheren Zukunft und Verfahren wie Y können dort Wichtiges leisten. Da digitale Datenbestände als wohldefiniert und im weiteren Sinne auch als korrekt und vollständig angesehen werden können, ist der oben genannte Nachteil von Y weniger schwerwiegend. Fehlleistungen einer unterliegenden Vektorisierung müssen nicht ausgeglichen werden.
Beispielsweise können digitale Grundkarten durch eine derartige Analyse in einen topologisch korrekten, und semantisch höherwertigen Bestand migriert werden. Auch andere CAD-Zeichnungs-Domänen aus den Bereichen Architektur, Maschinenbau etc. sind Kandidaten für die Modellierung und Analyse mittels Y.

Die folgenden Abbildung zeigt Ergebnisse der vorliegenden Implementierung von Y in C++. Exemplarisch wurden einige Sachverhalte in einer Sorte von Gasleitungsplänen modelliert: Linien mit verschiedenen Signaturen, Buchstaben, Ziffern, (Jahres-)Zahlen, Materialkonstanten, Leitungen mit eventuell mehrzeiliger Beschriftung.

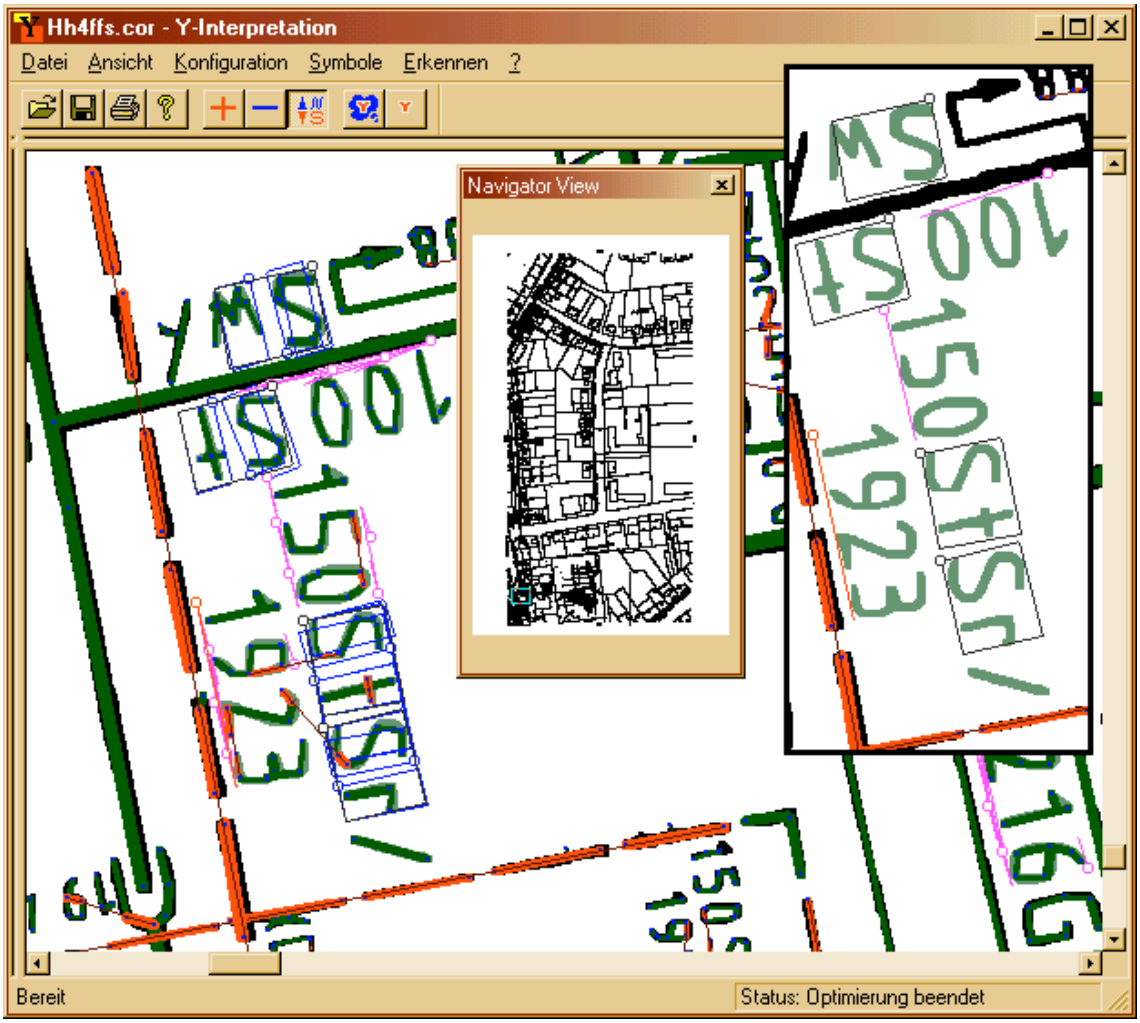


Abb. 6-43 Hauptfenster: Visualisierung aller erkannten Objekte, im Navigator: der gesamte A1-Plan mit markiertem Ausschnitt links unten, im schwarzen Rahmen: optimale Interpretation des Planes

In der Abbildung sind nicht alle extrahierten semantischen Bezüge dargestellt. Gut erkennbar sind die Klassen:

- *String* (blau),
- *Zahl* (violett),
- *Jahreszahl* (rot),
- *Materialstring*(schwarz) und
- *gerisseneLinie* (rot).

Die relativ abstrakten Klassen *Beschriftung* und *Leitung* (mit *Beschriftung*) sind in der Visualisierung nicht sichtbar, liegen aber in der Datenstruktur korrekt vor. Gut sichtbar ist im Hauptfenster, daß exzessiv Objekte erzeugt werden, um Fehler erster Art zu vermeiden. Beispielsweise werden sehr viele Strings und gerissene Linien in dem Plan entdeckt. Durch die globale Optimierung werden jedoch die dadurch entstehenden zahlreichen Konflikte perfekt gelöst (schwarzer Rahmen rechts im Bild).

Die Interpretation des gesamten Planes im Format A1 benötigt auf einem Notebook (266 MHz Pentium, 64 Megabyte RAM, Windows 98) insgesamt 26 Sekunden. Die Suche nach der optimalen Interpretation in der Menge aller Schnittstellenobjekte benötigt davon 5 Sekunden. Das Ergebnis ist bezüglich der modellierten Semantik fehlerfrei.

Danksagung

Ich danke meinem Vater und den vielen anderen, die mich zu diesem Text ermutigt und befähigt haben.

Erklärung

Ich erkläre hiermit, daß ich die vorliegende Arbeit selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen wörtlich oder sinngemäß übernommen Gedanken sind als solche kenntlich gemacht.

Druckkonventionen

Begriffe, die eine definierte Bedeutung in dieser Arbeit besitzen, werden, wenn sie vor ihrer Definition verwendet werden, fett gedruckt. Zum Beispiel werden **Objekt** und **Klasse** erst in einer intuitiven Bedeutung benutzt. In der Definition wird die erstmalige Erwähnung des zu definierenden Begriffs ebenfalls fett gedruckt.
In den Beispielen werden die Modelle der Planbestandteile kursiv gedruckt (z. B. *Bemaßung* oder *Leitung*).
Klassen werden in graphischen Darstellungen als getönte Ellipse, **Objekte** als farbloses Rechteck symbolisiert.
Die Bezeichner konkreter **Klassen** werden im Singular gebraucht, es sei denn, eine einzelne Instanz umfaßt eine ganze Menge von Entitäten.
Relationen, Variablen (die Elemente eines Raumes) und Funktionen werden klein, Räume und Wahrscheinlichkeiten weitgehend groß geschrieben.
Funktionen $f(x)$ einer **Klasse** c oder eines **Objektes** o werden unter Nutzung der Indexschreibweise notiert

$$f_c\,(x) := f(c, x) \text{ bzw. } f_o\,(x) := f(o, x).$$

In C++ würde man statt dessen c::f(x) bzw. o.f(x) schreiben.
Damit wird ein objektorientierter Entwurf suggeriert:

- **Klassen** sind Klassen im Sinne der Objektorientierung,
- **Objekte** sind deren Instanzen,
- f_o sind die objektspezifischen Methoden und
- f_c sind klassenspezifische (in C++: statische) Methoden

Stichwortverzeichnis

Verzeichnis der Beispiele

Verzeichnis der Algorithmen

Abbildungsverzeichnis

Literaturverzeichnis

[Albat Boersch Schirmer 1994] Albat, H., Boersch, I., Schirmer, H. (1994). *Prototyp eines Formularinterpreters für Blinde*. Wissenschaftliche Beiträge zur Informatik, 7(3).

[Bandemer Bellmann 1979] Bandemer, Bellmann (1979). Statistische Versuchsplanung. *MINÖL*, 19/2. BSB B.G.Teubner Verlagsgesellschaft, Leipzig.

[Bayer 1976] Bayer, O. (1976). Wahrscheinlichkeitsrechnung und mathematische Statistik. *MINÖL*, 17. BSB B.G.Teubner Verlagsgesellschaft, Leipzig.

[Booch 1994] Booch, G. (1994). *Object-oriented Analysis and Design. With Applications*. Benjamin/Cummings Pubishing Company, Inc.

[Bringmann Buchroithner Fuchs Seifert 1997] Bringmann, O., Buchroithner, M., Fuchs, S., Seifert, H.-H. (1997). Probabilistic Modelling and Analysis of Technical Drawings. *Digital Image Processing and Computer Graphics, SPIE Proceedings*. Wenger, E., Dimitrov, L.I. (eds) 3346.

[Bringmann Buchroithner Seifert 1997] Bringmann, O., Buchroithner, M., Seifert, H.-H. (1997). Unscharfe hierarchische Interpretation komplexer analoger Pläne für semantisch – adäquate Visualisierungen. *Nachrichten aus dem Karten- und Vermessungswesen*. Reihe I, 117.

[Bronstein Semedjajew 1995] Bronstein, I. N., Semedjajew K. A. (1996) Taschenbuch der Mathematik Teil II, B.G. Teubner, Stuttgart, Leipzig.

[Bronstein Semedjajew 1996] Bronstein, I. N., Semedjajew K. A. (1996) Taschenbuch der Mathematik, B.G. Teubner, Stuttgart, Leipzig.

[Crichton 1993] Crichton, M. (1993). *Congo.* Ballantine Books.

[Dori Velkovich Wenyin 1996] Dori, D., Velkovich, Y.,Wenyin, L. (1996). Object-process based recognition of ANSI and ISO standard dimensioning texts. *Lecture Notes on Comp. Sci.-Graphics Recognition.* Kasturi R., Tombre K. (eds), 1072.

[Emden-Weinert Hougardy Kreuter Prömel Steger 1996] Emden-Weinert, T., Hougardy, S., Kreuter, B., Prömel, H.J., Steger, A. (1996) *Einführung in Graphen und Algorithmen.* unveröffentlicht, www.informatik.hu-berlin.de/Institut/Struktur/Algorithmen/ga/

[Fuchs Bringmann Oganowski 1999] Fuchs, S., Bringmann, O., Oganowski W. (1999) Context Sensitive Analysis of Urban Main Network Plans. *ISAS/Sci*, Orlando, Florida.

[Gas 1995] (1995). *Zeichenvorschrift für Bestandspläne im Maßstab 1:500*. Gas GmbH, Dresden.

[Gibson 1950] Gibson, J.J. (1950). *The perception of the visual world*. Boston.

[Großmann Terno 1993] Großmann, Ch., Terno J. (1993). *Numerik der Optimierung*. B.G. Teubner, Stuttgart.

[Ihle Fuchs 1996] Ihle, T., Fuchs, S. (1996). Relational structures for hierarchical abstraction – applied to analysis of printed forms. *International Journal of Pattern Recognition and Artificial Intelligence*.

[Jänich 1996] Jänich, K. (1996). Topologie. Springer-Verlag, Berlin, Heidelberg.

[Köster 1995] Köster, M. (1995). Kontextsensitive Bildinterpretation mit Markoff-Zufallsfeldern. Verlag der Akademie der Wissenschaften, München.

[Kovalevski 1990] Kovalewski, V.A. (1990). New definition and fast recognition of digital straight segments and arcs. *IEEE Comuter Science Conferenceon Pattern Recognition*.

[Kulschewski 1997] Kulschewski, K. (1997). Building Recognition with Baysian Networks. *Semantic Modelling for the Acquisition of Topographic Information from Images and Maps*. Förstner, W., Plümer, L. (eds). Birkhäuser.

[Lengauer 1990] Lengauer, T. (1990). *Combinatorial Algorithms for Integrated Circuit Layout.* B.G.Teubner, Stuttgart.

[Mayer 1984] Mayer, H. (1984). Automatische, wissensbasierte Extraktion von semantischer Information aus gescannten Karten. *Geodätische Komission München*. C417.

[Messmer Bunke 1996] Messmer, B.T., Bunke H. (1996). Automatic lerning and recognition of graphical symbols in engineering drawings. *Lecture Notes on Comp. Sci.-Graphics Recognition*. Kasturi R., Tombre K. (eds), 1072.

[Pasternak 1996] Pasternak, B. (1996). *Adaptierbares Kernsystem zur Interpretation von Zeichnungen*. Dissertation, Universität Hamburg.

[Pavlidis 1980] Pavlidis, T. (1980). A thinning algorithm for discrete binary images. *Computer graphics and image prcessing*. 13.

[Pogio 1986] Poggio, T. (1986). Wie Computer und Menschen sehen. *Spektrum der Wissenschaft, Wahrnehmung und Visuelles System.* Spektrum der Wissenschaft Verlagsgesellschaft. Heidelberg.

[Pölzleitner Wechsler 1990] Pölzleitner,W., Wechsler, H. (1990). Invariant Pattern Recognition Using Associative Memory. *DIBAG Report 48.* Graz.

[Prechtel Bringmann 1998] Prechtel N., Bringmann, O. (1998). Near-Real-Time Road Extraction from Satellite Images Using Vector Reference Data. *ISPRS Proceedings of the Comm II Symposium: Data Integration: Systems and Techniques*. Cambridge (England).

[Russel Norvig 1995] Russel, S., Norvig, P. (1995). *Artificial intelligence – a modern approach*. Prentice Hall, Inc.

[Sanfeliu 1993] Sanfeliu, A. (1993). Maching Methods. *Proceedings of 8th Scandinavian Conference on Image Analysis*. Tromsø, Norwegen.

[Schoppe 1991] Schoppe, A., (1991). Behandlungsmöglichkeiten der Unschärfe von Daten und relationen. Unitext-Verlag, Göttingen.

[Smeulders Kate 1996] Smeulders, A.W.M., Kate, T. (1996). Software system design for paper map conversion. *Lecture Notes on Comp. Sci.-Graphics Recognition*. Kasturi R., Tombre K. (eds), 1072.

[Steinhagen Fuchs 1976] Steinhagen, H.-E., Fuchs, S. (1976). *Objekterkennung*. Verlag Technik, Berlin.

[Stroustrup 2000] Stroustrup; B. (2000). *The C++ Programming Language*. Addison Wesley.

[Tanimoto 1990] Tanimoto, S. L. (1990). *KI: Die Grundlagen*. R.Oldenbourg Verlag, München.

[Thomas 1996] Thomas, P et al (1996). A Combined High and Low Level Approach to Interpreting Scanned Engeneering Drawings. *Lecture Notes on Comp. Sci.-Graphics Recognition*. Kasturi R., Tombre K. (eds), 1072.

[Thrun 1995] Thrun, S. (1995). Explanation Based Neural Network Lerning, A Lifelong Lerning Approach. Kluwer.

[Tombre 1996] Tombre, K. (1996). Structural and Syntactic Methods in Line Drawing Analysis: To which Extent do they Work?. *Proceedings of SSPR*'96, Leipzig.

[Tombre 1998] Tombre, K. (1998). Ten Years of Research in the Analysis of Graphics Documents: Achievements and Open Problems. *Proceedings of 10th Portuguese Conference on Pattern recognition*, Lissabon.

[Wierschin 2000] Wierschin, T. (2000). Bewertung eines interaktiven Tools zur Interpretation von Leitungsplänen. *Diplomarbeit*, Technische Universität Dresden.

Symbolverzeichnis

		Seite
$\langle o \rangle$	Operator, der die bedingte Wahrscheinlichkeit von *o* auf das offene Intervall]0.5,1[abbildet	61
$\overline{relation}$	transitive Hülle einer Relation *relation*	14
$\otimes$	multiplikativer Operator	58
$\Pi(\Phi, \Gamma)$	Problem, Aufgabe; Operator, der Eingangsdaten Φ auf Ausgangsdaten Γabbildet	56
$\Pi \propto \Pi'$	Problem Π ist polynomiell auf Π' reduzierbar; Π ist nicht schwerer als Π'	57
$a(G)$	Unabhängigkeitszahl eines Graphen *G*	56
b_c	Schwellwert für die Erzeugungswahrscheinlichkeit der Objekte der Klasse *c*	37
b_{opt}	optimaler Schwellwert für die Erzeugungswahrscheinlichkeit der Objekte mehrerer Klassen	38
$]a,b[$	offenes Intervall; Menge der Zahlen $\{c: a<c<b\}$	
$\|Menge\|$	Kardinalität einer *Menge*	
2^V	Potenzmenge der Menge *V*	
$better_c$	abstrakte Funktion der Metaklassen **Iteration1** und **IterationN²**, die in den konkreten Klassen jeweils spezifiziert werden muß	29
c	Klasse; Menge von Objekten	12
C	Menge aller Klassen	
causes	binäre Relation je nach Kontext über Klassen oder über Objekten	14
c_o	Klasse, die das Objekt *o* enthält	13
component(.)	Menge der Bestandteile eines Objektes; Funktion eines Objektes	61
conflict	Unvereinbarkeit zweier Objekte; binäre Relation über Objekten	41
$conflict_{path}$	Teilmenge von *conflict*, die durch gemeinsame Pfade *path* zu den Objekten erzeugt wird	43
$const_c(.)$	Konstellationsfunktion einer Klasse *c*; filtert aus der Menge aller Parameter, die für die Existenz eines Objektes signifikanten Merkmale heraus	20
C_{out}	Klassen der Ausgabeschnittstelle	15
$C_{primitive}$	Klassen der Ausgabeschnittstelle	15
Def(.)	Definitionsbereich	
$environ_c$	Funktion, die alle Objekte in geometrischer Umgebung um ein Startobjekt liefert; abstrakte Funktion der Metaklasse **Umgebung**, die in den konkreten Klassen spezifiziert werden muß	25
$environNext_c$	Abstrakte Funktion der Metaklassen **Iteration1** und **IterationN²**	29
$environPrev_c$	Abstrakte Funktion der Metaklassen **Iteration1** und **IterationN²**	29
f(.)	akkumulierte Evidenz für das Fehlen des Objektes in einer bestimmten Interpretation	55
$G=(V, E)$	Graph *G* mit der Knotenmenge *V* und den Kanten *E*	56
$g_c(.)$	Objektgenerator; Funktion, die die Merkmale p_o der Objekte *o* der Klasse $c = c_o$ berechnet. Benutzt werden dafür die Parameter der Bestandteile von *o*. Diese Bestandteile sind auch wieder Objekte.	15
GO(.)	Globales Optimalitätskriterium; Funktion einer Interpretation	55
$GO_{norm}(.)$	normierendes Globales Optimalitätskriterium	72
$GO_{opt}(.)$	im stochastischen Sinne beste Bewertung einer Interpretation	62
$GO_{simple}(.)$	einfaches Globales Optimalitätskriterium	61
I	Menge aller Interpretationen	53
i	Interpretation einer Zeichnung; konfliktfreie, nicht erweiterungsfähige Menge von Objekten	53

i_{opt}	optimale Interpretation	55
line	Parametertyp, der zwei Koordinatenpaare zur Festlegung einer Strecke in einer Zeichnung enthält	
M	Modell einer Domäne; besteht aus statischem und dynamischen Modellanteilen	39
m_c	Anzahl der Komponentobjekte eines Objektes der Klasse c	15
$metaklasse_c$	Zusammenfassungsalgorithmus, welcher der Klasse c zugeordnet wird (z. B. Kartesisches Produkt, Umgebung, Iteration1...)	34
Metaklassen	Menge aller *metaklassen*	34
M_{stat}	statisches Modell einer Sorte technischer Zeichnungen	22
N	Menge der natürlichen Zahlen	
n_c	Anzahl der Parameter des Objektgenerators der Klasse c	12
NP	Komplexitätsklasse; „leichte und schwere Probleme“	56
NP-hart	Komplexitätsklasse; „schwere und sehr schwere Probleme“	57
NP-vollständig	Komplexitätsklasse; „schwere Probleme“	56
o	Objekt; Teil einer technischen Zeichnung; Element einer Klasse c_o	13
O	Menge aller Objekte einer Zeichnung	13
O_{out}	Menge der Objekte der Ausgabeschnittstelle	15
$O_{primitive}$	Menge der Objekte der Eingabeschnittstelle	15
P	Komplexitätsklasse; „leichte Probleme“	56
$P^{threshold}_{twoObjects}$	Wahrscheinlichkeit, daß in einer mehrdeutigen Situation mehr als ein Objekt erzeugt wird	38
$P^{threshold}_{errors}$	Wahrscheinlichkeit, daß ein zur Interpretation gehörendes Objekt nicht erzeugt wird	37
$Para_c$	Raum der Parametertypen einer Klasse c	12
part_of	Teil/Ganzes-Beziehung; binäre Relation über Objekten; Teilmenge von *causes* über Objekten	41
$parts_c$	boolescher Vektor, der die exklusiven Bestandteile einer Klasse markiert;	18
path	Relation, welche die (indirekte) Notwendigkeit eines Objektes für ein anderes Objekt beschreibt	43
P_c	bedingte Wahrscheinlichkeit der Existenz der Objekte der Klasse c; Funktion der signifikanten Merkmale der Klasse	20
p_o	Parametervektor eines Objektes; Element von $Para_{c_o}$	13
P_o	bedingte Wahrscheinlichkeit des Objektes o; rationale Zahl	20
Primitive(o)	alle primitiven, (indirekten) echten Bestandteile eines Objektes o	71
Q	Menge der rationalen Zahlen	
R	Menge der reellen Zahlen	
t(.)	Akkumulierte Evidenz für die Existenz des Objektes in einer bestimmten Interpretation	55
Unwahrscheinlich(o)	Funktion, die lokale Plausibilität eines Objektes o prüft; implementiert zusammen mit Metaklasse den Schwellwertklassifikator	38
$\mathcal{I}$(.)	Anzahl der unabhängigen Knotenmengen; Funktion eines Graphen	80
$\boldsymbol{O}$(.)	Ordnung der mittleren Zeitkomplexität als Funktion der Problemgröße	