



Universität  
Gesamthochschule Paderborn

Fachbereich · 17 Mathematik - Informatik

EINE AUSFÜHRUNGSMASCHINE FÜR GENERIC FUZZY REASONING NETS  
AUF BASIS UNSCHARFER PETRINETZE

**Diplomarbeit**

von

**Melanie Heitbreder**  
**Csilla-von-Boeselager Str. 9**  
**33104 Paderborn**

Vorgelegt bei

**Prof. Dr. Wilhelm Schäfer**

und

**Prof. Dr. Gregor Engels**

im Juli 1998

Diese Arbeit wurde am Lehrstuhl Softwaretechnik der Universität-Gesamthochschule Paderborn erstellt. Mein Dank gilt Herrn Dipl. Inf. Jens H. Jahnke für die interessante Aufgabenstellung, Herrn Prof. Dr. W. Schäfer für die Betreuung und Erstkorrektur, sowie Herrn Prof. Dr. G. Engels für die Zweitkorrektur der Arbeit. Außerdem danke ich Herrn Dipl. Inf. Oliver Heitbreder für die kritische Durchsicht des Manuskripts, allen Mitarbeitern der AG Schäfer, die mir im Verlauf dieser Arbeit zur Seite standen und meinen Eltern, die mir das Studium ermöglicht haben.

Ich versichere, die vorliegende Diplomarbeit selbständig angefertigt, alle benutzten Hilfsmittel vollständig angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer direkt oder mit Abänderungen entnommen wurde.

Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Paderborn, den

Melanie Heitbreder



*meinen Eltern*



# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung und Motivation</b>	<b>1</b>
1.1	Datenbank Reverse Engineering – Ein Überblick . . . . .	1
1.2	Szenario . . . . .	2
1.3	Varlet – Werkzeugunterstützung für Datenbank Reverse Engineering . . . . .	5
1.4	Anforderungen an eine geeignete Inferenzmaschine . . . . .	6
1.5	Zielsetzung . . . . .	7
1.6	Lösungsansatz . . . . .	8
1.7	Struktur der Arbeit . . . . .	8
<b>2</b>	<b>Logisches Schließen unter Unsicherheit</b>	<b>11</b>
2.1	Modale Logik . . . . .	11
2.2	Anfechtbare Logik . . . . .	12
2.3	Probabilistische Logik . . . . .	13
2.4	Sicherheitsfaktoren . . . . .	15
2.5	Fuzzy-Logik . . . . .	16
2.6	Possibilistische Logik . . . . .	18
2.7	Bewertung . . . . .	21
<b>3</b>	<b>Generic Fuzzy Reasoning Nets</b>	<b>25</b>
3.1	Die Semantik einer einfachen GFRN-Regel am Beispiel . . . . .	25
3.2	Axiome zur Klassifikation des initialen Wissens . . . . .	26
3.3	Eine GFRN-Spezifikation für das Eingangsszenario . . . . .	27
3.4	Die formale Syntax und Semantik der GFRNs . . . . .	29
3.4.1	Anwendung des Übersetzungsalgorithmus auf das Eingangsbeispiel . . . . .	34
<b>4</b>	<b>Das formale Modell der Inferenzmaschine: Ein unscharfes Petrinetz</b>	<b>39</b>
4.1	S/T-Netze . . . . .	39
4.2	Das unscharfe Petrinetzmodell von Konar und Mandal . . . . .	42
4.2.1	Das Schaltverhalten der K/M-Netze . . . . .	44
4.2.1.1	Die Berechnung der Fuzzy-Truth-Token . . . . .	44

4.2.1.2	Die Berechnung der Fuzzy-Beliefs	45
4.2.2	Anwendung des Modells von Konar und Mandal	46
4.3	Das unscharfe Petrinetzmodell der Inferenzmaschine - Fuzzy Reasoning Nets	49
4.3.1	Das formale Modell	49
4.3.2	Das Schaltverhalten der FRNs	51
4.3.2.1	Die Berechnung der Fuzzy-Truth-Token	51
4.3.2.2	Die Berechnung der Fuzzy-Beliefs	52
4.3.3	Algorithmus und Stabilitätsanalyse	53
4.3.4	Das Beheben von Begrenzungskreisen	63
<b>5</b>	<b>Überführung eines GFRNs in ein FRN: Die Expansion</b>	<b>69</b>
5.1	Die Abbildung einer GFRN-Regel in einem FRN	69
5.2	Der Inferenzalgorithmus	71
<b>6</b>	<b>Design und Implementierung</b>	<b>75</b>
6.1	Das Design der Ausführungsmaschine	75
6.2	Implementierungsdetails	76
6.3	Das Laufzeitverhalten der Inferenzmaschine	81
6.4	Die Entwicklungsumgebung	81
<b>7</b>	<b>Anwendung der Inferenzmaschine auf das Eingangsszenario</b>	<b>83</b>
<b>8</b>	<b>Zusammenfassung und Ausblick</b>	<b>91</b>
8.1	Zusammenfassung	91
8.2	Ausblick	92
<b>Anhang A</b>	<b>Beweise zu Kapitel 4</b>	<b>93</b>
<b>Anhang B</b>	<b>Klassendefinitionen</b>	<b>105</b>
	<b>Abbildungsverzeichnis</b>	<b>121</b>
	<b>Literatur</b>	<b>123</b>

**Index**

**127**



# 1 Einleitung und Motivation

## 1.1 Datenbank Reverse Engineering – Ein Überblick

Einer der wohl wichtigsten Schritte bei dem Entwurf von Datenbanken ist die Erstellung des konzeptionellen Schemas. Hier werden die in der Anforderungsanalyse gewonnenen Ergebnisse über die statische Semantik einer Datenbankanwendung strukturiert. Nur durch die sorgfältige Strukturierung aller für die Datenbank notwendigen (Daten-)Objekte und die Festlegung ihrer Beziehungen untereinander kann im späteren Datenbankeinsatz eine konsistente und nichtredundante Datenhaltung gewährleistet werden.

Aber nicht nur beim Datenbankentwurf, sondern auch in späteren Lebensphasen einer Datenbank, wie bei der Wartung oder Änderung, ist das konzeptionelle Schema wichtig. Leider werden aber Änderungen und Erweiterungen einer Datenbank oftmals nur am physikalischen und logischen Schema und am Applikationscode durchgeführt, ohne jedoch das konzeptionelle Schema und die Dokumentation ebenfalls zu aktualisieren. Viele relationale Datenbanken sind außerdem bereits sehr alt, und es haben oft mehrere Generationen von Entwicklern an der Entwicklung der Datenbank mitgewirkt. Aufgrund wechselnder Anforderungen und Möglichkeiten mußte die ursprüngliche Implementierung unter Umständen häufig angepaßt, also verändert werden. Dabei ist allerdings nicht davon auszugehen, daß diese Veränderungen, beispielsweise wegen Zeitdrucks während der Projektarbeit, in den Dokumentationen der Datenbank ebenfalls festgehalten worden sind. Auch kann man viele der ehemaligen Datenbank-Entwickler nicht nach dem aktuellen Design, bzw. nach Änderungen des ursprünglichen Designs befragen, wenn sie die Firma, in der die Datenbank eingesetzt wird, bereits verlassen haben.

Dies führt zu den sogenannten *Legacy-Datenbanksystemen*, bei denen möglicherweise die Konzeption, das physikalische Schema und die Dokumentation nicht mehr konsistent zueinander sind. Da aber heutzutage solche relationale Legacy-Datenbanken häufig anzutreffende, komplexe Systeme sind, ist ein Verfahren erforderlich, das zu einer bestehenden relationalen Datenbank ein aktuelles Design ermittelt. Mit diesem Aspekt befaßt sich das *Reverse Engineering von Datenbanken* [PB94] als Teil des *Reengineering von relationalen Datenbanken* [Arn93], wobei sich letzteres allgemein mit der Anpassung solcher Datenbanken an neue Erfordernisse beschäftigt. Dabei soll das Wissen, das in diesen Datenbanken gespeichert ist, d.h. die in den Datenbanken gespeicherten Daten und Geschäftsregeln, erhalten bleiben.

Weitere Anwendungsbereiche, für die ein aktuelles konzeptionelles Schema von Datenbanken notwendig ist – und damit für Reverse-Engineering –, sind neben der oben erwähnten Wartung und Änderung die *Erweiterung und Föderierung von Datenbanken*, die *Integration von relationalen und objektorientierten Datenbanken*, die *Migration von relationalen zu objektorientierten Datenbanken* [JSZ96], etc. Dabei meint die Erweiterung von Datenbanken beispielsweise das Hinzufügen von Datenobjekten oder Attributen zum konzeptionellen Schema. Die Integration soll den Datenaustausch zwischen verschiedenen Datenbanken ermöglichen, wohingegen bei der Föderierung von Datenbanken ein Zugriff auf die einzelnen heterogenen Datenbanken über eine gemeinsame, meist objektorientierte Zugriffsschicht erfolgen soll. Außerdem ist es zur Erfüllung neuer Anforderungen meist unumgänglich, Daten aus unterschiedlichen Anwendungsbereichen, wie z.B. der computerunterstützten Konstruktion (CAD: engl.: computer aided design), der computerunterstützten Fertigung (CIM: engl.: computer integrated manufacturing)

oder der Softwaretechnik (SE: engl.: software engineering) mit Geschäftsdaten zu kombinieren. Zur Modellierung einer einheitlichen Zugriffsschicht auf alle vorhandenen Daten werden die konzeptionellen Schemata aller beteiligten Datenbanken benötigt. Die Datenbankmigration als letztes der oben angeführten Beispiele hat allgemein die Aufgabe, eine Datenbank von einer technischen Plattform in eine andere zu überführen. Bei der Migration einer relationalen Datenbank in eine objektorientierte findet diese Überführung auf Grundlage ihres Designs statt.

Das aktuelle Design ist demzufolge für viele Anwendungsbereiche von relationalen Datenbanken von großem Interesse. Leider kann man dieses aus den oben genannten Gründen in vielen Fällen jedoch nicht aus den bereits vorhandenen Dokumentationen übernehmen. Deshalb bleibt zum Erlangen der aktuellen semantischen Informationen im Rahmen von Datenbank Reverse Engineering nur die Möglichkeit, diese direkt aus der entsprechenden relationalen Datenbank, d.h. aus dem Schema, den Daten und dem Applikationscode abzuleiten. Aber auch hierbei sind einige Probleme zu bewältigen. So bieten alte Datenbankmanagementsysteme (DBMS) nur eine eingeschränkte Funktionalität. Erst seit dem Datenbanksprachstandard SQL'92 ist es beispielsweise möglich, die Fremdschlüsselbeziehungen zwischen Relationen explizit anzugeben [KE96]. Bei einer Umstellung der Datenbanken auf neuere Versionen kann nicht sicher damit gerechnet werden, daß die Datenbank durch die neu hinzugekommene Funktionalität „modernisiert“ wurde, da der Datenbankadministrator z.B. keine Gelegenheit dazu hatte oder ihm die neue Funktionalität nicht bekannt war. Dieser Aspekt ist auch als „Problem der Abwärtskompatibilität“ bekannt. Außerdem enthalten viele Datenbankanwendungen zur Erhöhung der Effizienz Optimierungsstrukturen, die mit einer Denormalisierung des konzeptionellen Schemas einhergehen und beispielsweise Redundanzen verursachen. Auch solche Maßnahmen erschweren die Herleitung des Designs der Datenbank.

Heuristiken spielen bei der Ermittlung semantischer Informationen eine große Rolle. Abschnitt 1.2 zeigt einen Ausschnitt aus einer relationalen Datenbankanwendung (RDBA) und die „manuelle“ Herleitung einiger semantischer Fakten aus dieser Beispiel-Datenbank. Hierzu wird die Datenbank nach Indikatoren durchsucht, aus denen mit Hilfe von Heuristiken semantische Informationen und Integritätsbedingungen extrahiert werden.

## 1.2 Szenario

Abbildung 1 zeigt einen Ausschnitt aus einer relationalen Datenbankanwendung, bestehend aus seinem Schema, seinen Daten und seinem Applikationscode. An diesem sollen exemplarisch einige semantische Informationen aus den Hinweisen abgeleitet werden, die sich in einer solchen Datenbank finden lassen. Das Beispiel enthält als Mieter-Vermieter-Szenario eine Tabellendefinition für die Relation **Mieter** mit den Attributen *Name*, *Haus*, *Ap*, *Miete* und *HMName* sowie eine Definition der Tabelle **ApHaus** mit den Attributen *HausID*, *Wohnungen*, *Straße* und *Stadt*. Der dargestellte Ausschnitt aus dem Applikationscode zeigt im oberen Teil eine SQL-Anfrage, in der Personen aus der Tabelle **Mieter** gesucht werden, die in demselben Haus wohnen, aber unterschiedliche Namen haben. Die zweite Anfrage bestimmt den Wert des Attributs *Haus* aus der Tabelle **Mieter** für einen Bewohner mit einem bestimmten Namen. Die Daten in Abbildung 1 zeigen eine kleine Ausprägung für die Tabellen **Mieter** und **ApHaus**.

Relationale Datenbankanwendungen enthalten eine Vielzahl von Hinweisen über ihre Semantik und ihre Integritätsbedingungen. Leider konnten manche semantische Informationen, wie bereits erwähnt, gerade in alten relationalen Datenbanksystemen wegen ihrer eingeschränkten

Funktionalität nicht explizit angegeben werden. Dazu zählen z.B. Fremdschlüsselbeziehungen zwischen Tabellen oder die Schlüsseleigenschaft bestimmter Attribute. Durch Anwendung generischen Expertenwissens in Form von Heuristiken als „Daumenregeln“ können Experten aus kleineren Datenbankanwendungen jedoch solche Informationen ableiten. Auch in einem entsprechenden wissensbasierten Analysewerkzeug sollen diese Heuristiken spezifizierbar und für die Analyse relationaler Datenbanken anwendbar sein.

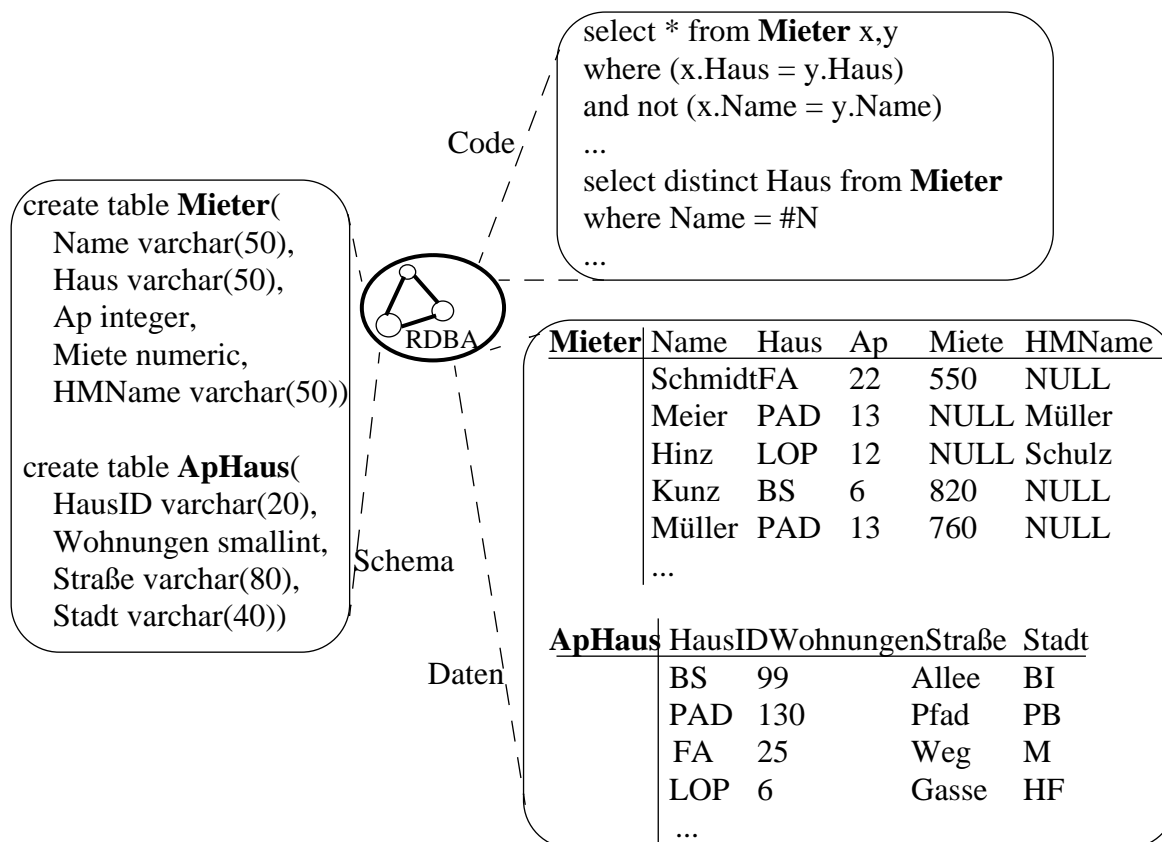


Abb. 1: Ausschnitt aus einer relationalen Datenbankanwendung (RDBA)

Im folgenden wird nun gezeigt, wie die Ableitung von Beziehungen zwischen Tabellen mit Hilfe von Heuristiken vor sich gehen kann. Hinweise auf Fremdschlüsselbeziehungen zwischen Tabellen lassen sich u.a. im Schemakatalog einer Datenbank finden. So liefern ähnliche Namen von Attributen derselben oder verschiedener Relationen solche Anhaltspunkte. Die Bezeichnungen der Attribute *Name* und *HMName* der Tabelle **Mieter** beispielsweise, sowie die Attribute *Haus* der Tabelle **Mieter** und *HausID* der Tabelle **ApHaus** ähneln sich in hohem Maße.

Eine weitere wichtige Information eines konzeptionellen Schemas einer relationalen Datenbank stellt die Frage nach den Schlüsseln von Tabellen dar. Auch Schlüsselkandidaten können in veralteten relationalen Datenbanksystemen mit einem Sprachstandard älter als SQL'92 [KE96] nicht explizit angegeben werden.

Indikatoren für die Schlüsseleigenschaft von Attributen können bestimmte Anfragemuster im Applikationscode liefern. Diese Muster haben eine exakt definierte Form und werden auch *Clichés* genannt [JSZ97, And94].

Beispiele für solche Clichés sind die beiden SQL-Anfragen aus Abbildung 1. Das erste Cliché wird *cyclic-join* Cliché genannt, da es einen Verbund über Attribute derselben Tabelle durchführt. Ein solcher Verbund liefert einen starken Hinweis auf eine Schlüsseleigenschaft des Attributs *Name*, da die Anfrage zwei Mieter aus der Relation **Mieter** sucht, die in demselben Haus wohnen, sich jedoch in ihrem Namen unterscheiden. Diese Tatsache weist möglicherweise darauf hin, daß sich die Tupel aus der Relation **Mieter** generell in dem Attribut *Name* unterscheiden und *Name* somit der Schlüssel dieser Relation ist. Der Aufbau sowie das Finden von Clichés werden hier nicht weiter erläutert, sondern sind Thema einer weiteren Diplomarbeit [BB98].

Eine Datenbank kann auch widersprüchliche Indikatoren über ihre semantischen Eigenschaften enthalten. Als Indikator gegen die Schlüsseleigenschaft des Attributs *Name* der Relation **Mieter** kann nämlich die zweite in Abbildung 1 dargestellte SQL-Anfrage herangezogen werden. In diesem sogenannten *select-distinct* Cliché weist das Schlüsselwort *distinct* darauf hin, daß das Attribut *Name* der Relation **Mieter** kein Schlüssel ist. Solche Widersprüche müssen selbstverständlich bei der Analyse erkannt und aufgelöst werden, bevor ein korrektes konzeptionelles Schema resultieren kann.

Höhere und insbesondere objektorientierte Modellierungskonstrukte, wie die Vererbung, sind zwar oft in einer RDBA enthalten, werden aber von relationalen Datenbanksystemen nicht unterstützt. Einen Hinweis auf eine solche versteckte Vererbungsbeziehung läßt die Beispielausprägung der Relation **Mieter** in Abbildung 1 erkennen. Hier tragen entweder die Attribute *Miete* oder *HMName* einen NULL-Wert. Diese Tatsache weist auf Varianten der Relation **Mieter** und damit auf eine versteckte Vererbungsbeziehung hin.

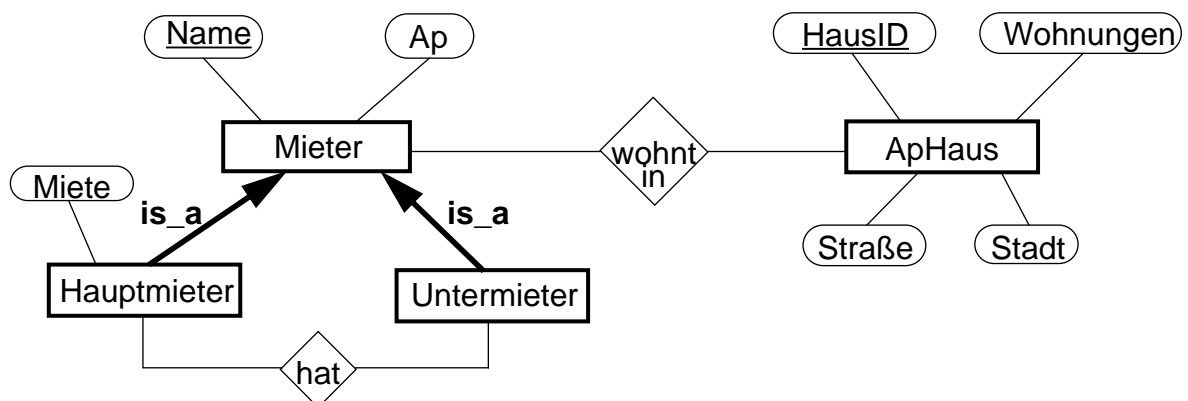


Abb. 2: Konzeptionelles Schema in EER-Notation

Abbildung 2 zeigt das konzeptionelle Schema des Szenarios in EER-Notation [BCN92]. Die Schlüsselkandidaten *Name* der Tabelle **Mieter** und *HausID* der Tabelle **ApHaus** sind unterstrichen dargestellt. Die Varianten der Relation **Mieter** sind durch die dicker gezeichneten **is\_a** Kanten kenntlich gemacht. Der **Mieter** läßt sich demnach in eine Variante **Hauptmieter** mit den Attributen *Name*, *Haus*, *Ap* und *Miete* und in eine Variante **Untermieter** mit den Attributen *Name*, *Haus*, *Ap* und *HMName* unterteilen. Die Fremdschlüsselbeziehung der Relation **Mieter** über die Attribute *Name* und *HMName* ist im EER-Diagramm durch die hat-Beziehung dargestellt, die Fremdschlüsselbeziehung zwischen den Tabellen **Mieter** und **ApHaus** über die Attribute *Haus* und *HausID* durch die *wohnt\_in*-Beziehung.

### 1.3 *Varlet* – Werkzeugunterstützung für Datenbank Reverse Engineering

Eine „manuelle“ Vorgehensweise zur Ableitung des konzeptionellen Schemas ist bei realen Datenbank Anwendungen wegen ihrer Komplexität nicht durchführbar, da solche Datenbanken häufig mehrere 100 000 Zeilen Applikationscode und unüberschaubar große Datenmengen umfassen. Die Unterstützung der Datenbankanalyse durch ein geeignetes Werkzeug, wie zum Beispiel der Reengineering-Umgebung *Varlet* (*Verified Analysis and Reengineering of Legacy Database Systems Using Equivalence Transformations*), ist somit unerlässlich.

Die Reengineering-Umgebung *Varlet* stellt die erforderlichen Werkzeuge zur Migration einer relationalen zu einer objektorientierten Datenbank und damit auch für das Datenbank Reverse Engineering zur Verfügung. *Varlet* verfolgt dazu einen wissensbasierten Ansatz zur Datenbankanalyse. Insbesondere ist es möglich, Reengineering-Wissen in diesem Werkzeug explizit zu spezifizieren, zu modifizieren und auszuwerten. Die Spezifikation generischen Reengineering-Wissens erfolgt in graphischer und abstrakter Weise mit Hilfe der Generic Fuzzy Reasoning Nets (GFRNs) [JSZ97], die in Kapitel 3 näher erläutert werden.

Eine wichtige Eigenschaft der GFRNs ist, daß die Spezifikation des Wissens abhängig von technischen Parametern, wie dem verwendeten Datenbanksystem oder der verwendeten Sprache, und nichttechnischen Parametern, wie dem persönlichen Programmierstil des jeweiligen Datenbankentwicklunglers, erfolgen kann. Ein Beispiel dafür, wie der Programmierstil die Analyseergebnisse beeinflussen kann, zeigen die beiden Clichés in Abbildung 1. Während das *cyclic-join* Cliché auf die Schlüsseleigenschaft des Attributs *Name* hinweist, widerspricht dieser Annahme das *select-distinct* Cliché. Hätte ein gewissenhafter Entwickler, der das Schlüsselwort *distinct* nur dann verwendet, wenn wirklich ein mehrfaches Vorkommen derselben Tupel in der Ergebnisrelation der Anfrage erwartet wird, den Applikationscode erstellt, wäre ein solcher Widerspruch vermeidbar gewesen. Das Analysewerkzeug soll sich also flexibel an diese jeweiligen Gegebenheiten anpassen lassen, beispielsweise durch eine unterschiedliche Gewichtung speziell der widersprüchlichen Indikatoren. Findet das Analysewerkzeug etwa ein *cyclic-join* Cliché als Indikator für die Schlüsseleigenschaft des Attributs *Name*, sollte dieser Indikator höher gewichtet werden, als ein Indikator gegen diese Eigenschaft, der durch ein *select-distinct* Cliché gefunden wird.

Während bei existierenden Ansätzen [FV95, And94, PB94, PKBT94, JK90, SK90, DA87] das Analysewissen im Analysewerkzeug fest kodiert ist, soll das Wissen in *Varlet* explizit spezifizierbar und flexibel an veränderte Rahmenbedingungen anpaßbar sein, wie beispielsweise das Hinzukommen neuen Wissens.

Eine direkte Analyse einer relationalen Datenbank mit Hilfe der GFRNs führt wegen der oben erwähnten Abstraktheit des Wissens zu einem möglicherweise sehr komplexen Verfahren. Des-

halb ist es ratsam, eine GFRN-Spezifikation vor der Analyse in ein Modell zu überführen, das effizienter ausführbar ist. Dieses ausführbare Modell wird im folgenden *Inferenzmaschine* genannt und ist Gegenstand dieser Diplomarbeit.

Die GFRN-Spezifikation des generischen Datenbankwissens wird in *Varlet* mit einer speziellen Legacy-Datenbank parametrisiert und unter Verwendung des Datenbankschemas, der Daten und des Applikationscodes in ein ausführbares Modell überführt. Zur Ausführung einer GFRN-Spezifikation wurde ein unscharfes Petrinetz (FPN: engl.: fuzzy petri net) Modell ausgewählt, das für diesen Zweck besonders geeignet ist. Nach der Ausführung des FPNs resultieren als Zwischenergebnis des Inferenzprozesses semantische Informationen über eine relationale Datenbank, die eventuell noch unvollständig oder widersprüchlich sind. Durch Hinzufügen weiterer Annahmen und einer erneuten Auswertung der Inferenzmaschine hat der Reengineer die Möglichkeit, die Analyseergebnisse zu beeinflussen. Abbildung 3 stellt diesen Prozeß noch einmal graphisch dar.

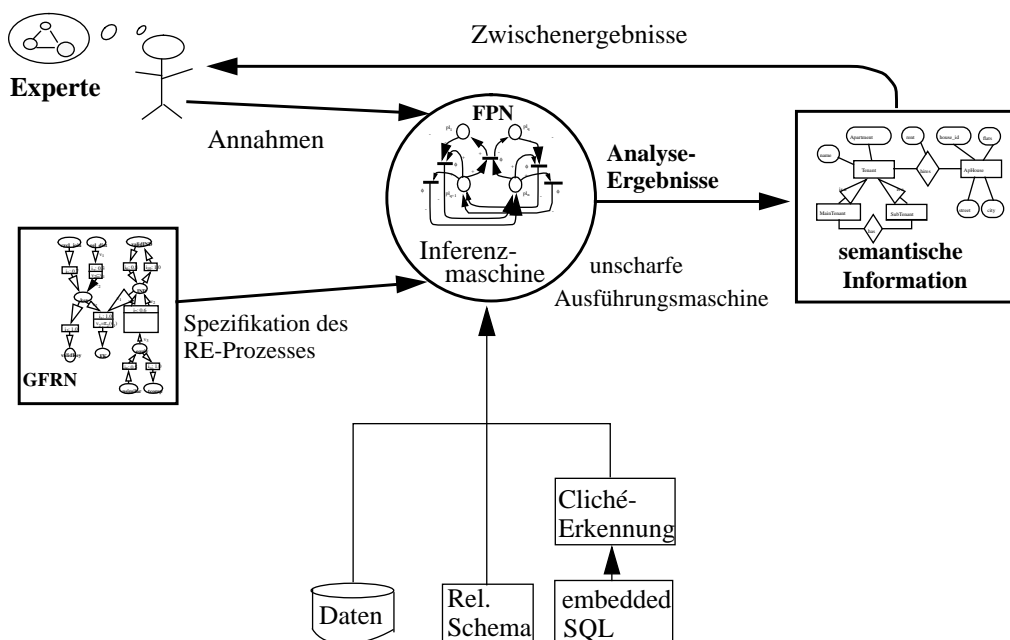


Abb. 3: Datenbankanalyse in *Varlet*

#### 1.4 Anforderungen an eine geeignete Inferenzmaschine

Die bisherigen Erläuterungen zur Ableitung eines konzeptionellen Schemas einer Datenbank und der dabei möglicherweise entstehenden Probleme werfen einige Anforderungen auf. Eine geeignete Inferenzmaschine sollte diese erfüllen, um möglichst aussagekräftige Analyseergebnisse über den semantischen Inhalt einer Datenbank liefern zu können.

### **Anforderung 1: Interaktion**

Ein vollautomatischer Prozeß zur Ableitung semantischer Informationen ist nicht sinnvoll, da der Reengineer zu Beginn einer Analyse eventuell nur eine sehr unvollständige oder widersprüchliche Menge an Wissen einbringen kann. Daraus resultierende Unsicherheiten und Widersprüche müssen aufgelöst werden. Während der Inferenz neu gewonnene oder von außen herangetragene Informationen oder die Auflösung von Widersprüchen können es notwendig machen, den weiteren Verlauf der Inferenz durch Revision von Ergebnissen oder Hinzufügen von Wissen zu beeinflussen.

### **Anforderung 2: Handhabung von unsicherem Wissen und Widersprüchen**

Wie das Szenario aus Abschnitt 1.2 gezeigt hat, spielen Heuristiken bei der Ableitung semantischer Informationen aus einer RDBA eine wichtige Rolle. Aufgrund der Komplexität gerade alter relationaler Datenbanken sind manche Vermutungen eines Reengineers nicht mehr „durch scharfes Hinsehen“ überprüfbar. Mit Hilfe von Heuristiken als „Daumenregeln“, die im Werkzeug repräsentiert werden, lassen sich geeignete Hinweise in einer RDBA finden, die bei der Ableitung aussagekräftiger Analyseergebnisse helfen. Solche Heuristiken können jedoch zu Widersprüchen führen, wie die Heuristiken über Schlüssel-eigenschaften aus *cyclic-join* und *select-distinct* Clichés gezeigt haben. Das Wissen über eine RDBA kann somit unsicher und widersprüchlich sein. Trotz allem wird gefordert, daß das Analysewerkzeug dieses unsichere Wissen verarbeiten und Widersprüche aufzeigen kann.

### **Anforderung 3: Effizienz**

Mit der Inferenzmaschine soll es möglich sein, eine gegebene RDBA auf Grundlage des Wissens zu analysieren, das in einer abstrakten GFRN-Spezifikation beschrieben worden ist. Die dazu notwendigen Analyseoperationen sollen nach ihrem Laufzeitverhalten klassifizierbar sein, so daß besonders aufwendige Analyseoperationen nur bei Bedarf ausgeführt werden können.

### **Anforderung 4: Nichtmonotonie**

Die verspätete Ausführung teurer Analyseoperationen bei Bedarf kann dazu führen, daß nach teilweise erfolgter Ableitung von Wissen noch neue Indikatoren über die Semantik der zu analysierenden Datenbank gefunden werden. Diese beeinflussen unter Umständen die bisher gefundenen Analyseergebnisse und können diese sogar falsifizieren. Deshalb ist ein nichtmonotoner Analyseprozeß notwendig, der bereits gefundene Ergebnisse bei Hinzukommen neuer Fakten überprüft und ggf. korrigiert.

## **1.5 Zielsetzung**

Das Ziel dieser Diplomarbeit ist es, einen Ausführungsmechanismus für die oben erwähnten GFRNs zur Wissensspezifikation als Teil des Analysewerkzeugs der *Varlet*-Umgebung zu implementieren. Dabei sollen die Anforderungen 1 bis 4 dieses Abschnitts erfüllt werden, also die Forderungen nach der Möglichkeit zur Interaktion, die Forderung nach Handhabung von unsicherem und widersprüchlichem Wissen und die Forderung nach der Übersetzung einer gegebene-

nen GFRN-Spezifikation in ein adäquates und effizient ausführbares Modell, das die Option zur verzögerten Ausführung teurer Analyseoperationen bietet.

## 1.6 Lösungsansatz

Die Maßnahmen, die zur Ausführung einer gegebenen GFRN-Spezifikation notwendig sind, lassen sich grob in folgende Komponenten aufteilen:

- 1) Eine Ausführungsmaschine zur Inferenz von Aussagen über den semantischen Inhalt einer relationalen Datenbankanwendung auf Grundlage einer gegebenen GFRN-Spezifikation und Fakten über eine konkrete Legacy-Datenbank. Die Spezifikation und Implementierung der Ausführungsmaschine bilden den Schwerpunkt dieser Diplomarbeit.
- 2) Ein Übersetzungsmechanismus zur Überführung einer gegebenen GFRN-Spezifikation in ein ausführbares Modell, auf dem die Auswertungsmaschine ihre Inferenz startet.

Für die Erstellung dieser Arbeit steht bereits eine formale Definition für die *Generic Fuzzy Reasoning Nets* zur Verfügung, deren Erläuterung Bestandteil von Kapitel 3 ist.

Die Ausführungsmaschine für eine GFRN-Spezifikation wird auf Basis eines unscharfen Petrietz (FPN) Modells als Erweiterung eines Modells von Konar und Mandal [KM96] implementiert. Ein solches FPN stellt sich zu diesem Zweck aus mehreren Gründen als besonders geeignet dar. Zum einen erlaubt es eine natürliche Darstellung der Inferenzregeln einer GFRN-Spezifikation, weshalb eine einfache Übersetzung dieser Spezifikation in eine FPN-Darstellung möglich ist. Zum anderen läßt sich ein FPN wegen seiner besonderen Struktur und modellinhärenten Nebenläufigkeit einfach und parallel ausführen.

Als Theorie, die der Simulation dieses FPN-Modells zugrunde liegt, wurde die *Possibilistische Logik* [DLP94] ausgewählt. Diese basiert auf der *Fuzzy-Logik* [Zad78, Zad75].

Einige der in dieser Diplomarbeit gewonnenen Ergebnisse wurden bereits in [JH98] veröffentlicht.

## 1.7 Struktur der Arbeit

Kapitel 2 stellt zunächst einige Ansätze zu unsicherer Logik vor, die in der Literatur zu finden sind. Diese Ansätze werden kurz verglichen, und es wird ein für die Inferenzmaschine besonders geeigneter ausgewählt.

Kapitel 3 stellt die Sprache zur Spezifikation des Expertenwissens dar, die *Generic Fuzzy Reasoning Nets* (GFRNs).

Anschließend stellt Kapitel 4 zunächst den Aufbau eines einfachen Petrietzes vor, bevor es anschließend das Grundmodell von Konar und Mandal und dann die für diese Diplomarbeit notwendigen Erweiterungen darlegt.

Eine Skizzierung des Expansionsalgorithmus erfolgt in Kapitel 5.

Kapitel 6 beschreibt das Design der implementierten Module und die Entwicklungsumgebung, in der sie erstellt wurden, sowie die Idee einiger interessanter Implementierungsdetails.

Kapitel 7 zeigt noch einmal im Zusammenhang die GFRN-Spezifikation für das Eingangsszenario, ihre Übersetzung in das ausführbare Modell und dessen Ausführung.

Als letztes faßt Kapitel 8 die Ergebnisse dieser Arbeit zusammen und gibt einen Ausblick auf Erweiterungen und zusätzliche Anwendungsmöglichkeiten.



## 2 Logisches Schließen unter Unsicherheit

Kapitel 1 erwähnte bereits die Qualität des Reengineering-Wissens, mit dessen Hilfe das konzeptionelle Schema einer relationalen Datenbank abgeleitet werden soll. Da das Wissen vor der Analyse einer Datenbank meist noch sehr unvollständig und vage sein kann, muß der Reengineer die Möglichkeit haben, früher getroffene Annahmen zu korrigieren oder ganz aus der Analyse zu entfernen. Außerdem sollen völlig neue Erkenntnisse während der Inferenz hinzugefügt werden können. Wegen der Ableitung des Wissens auf Grundlage von Heuristiken können auch Widersprüche enthalten sein, deren Existenz die Analyse nicht zum Stillstand bringen darf, sondern ebenfalls Interaktionen hervorrufen muß. Einmal gewonnene Analyseergebnisse sollen nach einer Interaktion des Reengineers, d.h. nach dem Verändern, Hinzufügen oder Löschen von Ableitungsregeln also revidiert werden können, was einen nichtmonotonen Schlußfolgerungsprozeß erfordert.

In der Literatur werden mehrere Ansätze zum logischen Schließen unter Unsicherheit diskutiert, die diese Problematik auf unterschiedliche Weise berücksichtigen. Es werden grob zwei Arten von nichtmonotoner Logik unterschieden, nämlich die *symbolischen* und die *numerischen* Ansätze [Gin94]. Zu den symbolischen Ansätzen gehören beispielsweise die *Modale Logik* und die *Anfechtbare Logik*. Zu den numerischen Ansätzen zählen die *Probabilistische Logik*, das Konzept der *Sicherheitsfaktoren* bzw. der *Konfidenzen*, die *Fuzzy-Logik* und die *Possibilistische Logik*.

Die eben erwähnten symbolischen und numerischen Logikkalküle werden in den folgenden Abschnitten dieses Kapitels in ihren Grundzügen kurz vorgestellt und anschließend verglichen.

### 2.1 Modale Logik

Im Gegensatz zu der klassischen Logik der Mathematik wird einer Aussage  $p$  in der Modalen Logik [Haj94] nicht mehr im absoluten Sinne der Wert *wahr* oder *falsch* zugeordnet, sondern es werden die jeweiligen Umstände, unter denen  $p$  gilt, mitberücksichtigt. Entsprechend diesem Interpretationswandel werden auch nicht mehr die Ausdrücke *wahr* und *falsch* angewendet, sondern man spricht von der *Notwendigkeit* und *Möglichkeit* einer Aussage. Dabei bezeichnet

$\diamond p$  : „Es ist *möglich*, daß  $p$  wahr ist.“

$\square p$  : „Es ist *notwendig*, daß  $p$  wahr ist.“

Zusammen mit den Ausdrücken der Aussagenlogik [Schö92] und Konstanten für Prädikate und Funktionen wird die Modale Logik aufgebaut. Die Interpretation der Formeln ist jedoch schwieriger als in der herkömmlichen Aussagenlogik, da die Umstände, unter denen eine Aussage notwendig oder möglich ist, mitzubersichtigen sind.

Die Modale Logik wurde erst durch Einführung einer adäquaten formalen Semantik, der sogenannten Kripke-Modelle [Haj94], ein Teil der mathematischen Logik. Im allgemeinen Fall ist ein Kripke-Modell ein Tripel  $K = \langle W, \Vdash, S \rangle$ , bei dem  $W$  eine nichtleere Menge möglicher Zustände,  $\Vdash$  eine Funktion, die jeder atomaren Formel  $P$  und jedem Zustand  $w \in W$  einen Wert 0 oder 1 zuweist, und  $S$  eine Struktur auf  $W$  ist. In den klassischen Systemen ist  $S$  eine binäre Relation  $\mathfrak{R}$  auf  $W$ , d.h.  $\mathfrak{R} \subseteq W \times W$ . Dann ist  $\Vdash$  für alle Formeln folgendermaßen defi-

niert:

$w \Vdash A \wedge B$  gdw.  $w \Vdash A$  und  $w \Vdash B$   
 $w \Vdash \neg A$  gdw. *nicht*  $w \Vdash A$   
 $w \Vdash \Box A$  gdw.  $\forall v$ , für die  $w \mathfrak{R} v$ :  $v \Vdash A$  und  $w \Vdash B$ ,  
 d.h.  $A$  ist genau dann notwendigerweise in  $w$  wahr, wenn  $A$  in allen erreichbaren Zuständen wahr ist.

Die Vorteile der Modalen Logik liegen darin, daß bei Wahl einer Relation  $\mathfrak{R}$  mit einfachen Eigenschaften auch ein einfaches Axiomensystem resultiert. In [Haj94] werden dazu drei einfache Beispiele gezeigt, von denen an dieser Stelle nur das erste vorgestellt werden soll:

Ist  $\mathfrak{R}$  eine Äquivalenzrelation (reflexiv, symmetrisch, transitiv), ergibt sich das folgende Axiomensystem:

$\Box(A \rightarrow B) \rightarrow \Box A \rightarrow \Box B$ ,  
 $\Box A \rightarrow \Box \Box A$ ,  
 $\Box A \rightarrow A$  und  
 $\Diamond A \rightarrow \Box \Diamond A$ .

Die Deduktionsregeln sind der Modus Ponens und die „Necessitation“-Regel mit der Form  $A \rightarrow \Box A$ .

## 2.2 Anfechtbare Logik

Die Anfechtbare Logik (engl.: defeasible logic) [Nute94, PAA94] verfolgt die Idee, Widersprüche in den vorliegenden Ableitungsregeln und Fakten durch eine Definition von Prioritäten auf den Regeln aufzuheben. Dazu werden die Regeln der Anfechtbaren Logik unterteilt in:

**Feststehende Regeln:** Falls  $A$ , dann  $B$  mit der Darstellung  $B \leftarrow A$ .

**Feststehende Fakten:**  $A$  ist wahr, Darstellung:  $A$ , bzw.  $A$  ist falsch, Darstellung:  $\neg A$ .

**Anfechtbare Regeln:** Falls  $A$ , dann gilt normalerweise  $B$ , Darstellung:  $B \leftarrow A, \sim ab$ . Hierbei ist  $\sim ab$  ein neues Prädikatsymbol. Als Beispiel betrachte man die Regel „Vögel fliegen normalerweise“ mit der Präsentation  $fliegt(X) \leftarrow Vogel(X), \sim ab$ .

**Bekannte Ausnahmen zu den Anfechtbaren Regeln:** Unter bestimmten Bedingungen COND gibt es bekannte Ausnahmen zu einer anfechtbaren Regel  $H_1 \leftarrow B_1, \sim ab_1$ :

$$ab_1 \leftarrow COND$$

Ein Beispiel für eine solche bekannte Ausnahme ist: „Pinguine sind eine Ausnahme zu der Regel *Vögel fliegen normalerweise*“. Die Darstellung dieser Ausnahme sieht dann folgendermaßen aus:  $ab(X) \leftarrow Pinguin(X)$ .

**Mögliche Ausnahme zu den Anfechtbaren Regeln:** Unter bestimmten Bedingungen *COND* gibt es keine mögliche Ausnahme zu einer anfechtbaren Regel  $H_1 \leftarrow B_1, \sim ab$ . Die Darstellung dieser Aussage hat die Form:

$$ab_1(X) \leftarrow \sim \neg COND(X) .$$

Als Beispiel für diesen Typ der Ausnahmeregel betrachte man die Aussage „Tiere, von denen man nicht weiß, daß sie kein Pinguin sind, sind eine Ausnahme zu der Regel *Vögel fliegen normalerweise* ( $f(X) \leftarrow V(X), \sim ab$ )“:  $ab(X) \leftarrow \sim \neg Pinguin(X)$ .

**Präferenz-Regeln:** Unter der Bedingung *COND* ziehe die Anwendung der Regel  $H_1 \leftarrow B_1, \sim ab_1$  der Anwendung der Regel  $H_2 \leftarrow B_2, \sim ab_2$  vor. Die Repräsentation ist:

$$ab_1 \leftarrow COND, \sim ab_2$$

Ein Beispiel hierfür ist: „Wenn für Pinguine die Regel *Pinguine können normalerweise nicht fliegen* anwendbar ist, vermeide die Anwendung der Regel *Vögel können normalerweise fliegen*“:  $ab_{Vogel}(X) \leftarrow Pinguin(X), \sim ab_{Pinguin}(X)$  .

Mit Hilfe der Anfechtbaren Logik ist es also möglich, eine Priorität für die Anwendung von Inferenzregeln anzugeben. Dabei sind die Ausnahmeregeln denjenigen Regeln vorzuziehen, zu denen sie die Ausnahme bilden. Ausnahmen können zu Prädikaten, zu Regeln und zu Ausnahmeregeln definiert werden.

Die bisher beschriebenen Ansätze gehören zur Klasse der symbolischen Logiken. Der nun folgende Teil wendet sich vier numerischen Logiken zu.

### 2.3 Probabilistische Logik

Die Probabilistische Logik [Paa88, Rol89] basiert auf der Wahrscheinlichkeitstheorie und verfügt damit über eine präzise mathematische Grundlage. Sie ist dort einsetzbar, wo es nötig ist, Unsicherheit über das Eintreten bestimmter Ereignisse zu quantifizieren. Jedem möglichen Ereignis  $E$  aus einem Ereignisraum  $S$  wird eine Wahrscheinlichkeit  $p(E)$  zwischen 0 und 1 zur Quantifizierung der Eintrittswahrscheinlichkeit des Ereignisses zugeordnet. Dieser Wert wird beispielsweise durch eine Wahrscheinlichkeitsfunktion vorgegeben oder durch Schätzungen/Intuition, direkte Messungen oder durch statistische Erhebungen der relativen Eintrittshäufigkeit des Ereignisses  $E$  ermittelt. Ein Wahrscheinlichkeitswert für  $p(E)$  von 1 bedeutet sicheres Wissen, daß  $E$  eintreten wird, ein Wahrscheinlichkeitswert für  $p(E)$  von 0 hingegen steht für das sichere Wissen, daß  $E$  nicht eintreten wird.

In Expertensystemen (ES) wird die Probabilistische Logik eingesetzt, wenn das in der Datenbank (DB) repräsentierte Wissen, d.h. die in der DB abgelegten Regeln und Aussagen, unsicher ist. Ziel der Probabilistischen Logik ist damit eine Definition und Auswertung eines Wahrscheinlichkeitsmaßes für logische Aussagen. Wahrscheinlichkeitswerte für logische Aussagen haben dabei eine etwas andere Semantik als für herkömmliche Ereignisse von Zufallsexperi-

menten in der Wahrscheinlichkeitstheorie. Während in der Wahrscheinlichkeitstheorie Wahrscheinlichkeiten für das Eintreten von Ereignissen durch wiederholtes Durchführen desselben Zufallsexperimentes gewonnen werden können – diese Wahrscheinlichkeiten werden *objektive Wahrscheinlichkeiten* [Rol89] genannt und werden durch die relative Häufigkeit approximiert –, ist es für logische Aussagen in der Regel nicht möglich, die Wahrscheinlichkeiten über Wiederholungen oder Statistiken zu ermitteln. Solche geschätzten Wahrscheinlichkeiten – auch *subjektive Wahrscheinlichkeiten* [Rol89] genannt – drücken eher eine persönliche Überzeugung des schätzenden Experten für die Chance aus, daß die Aussage oder Regel wahr sein könnte, als eine statistisch nachweisbare Tatsache.

Jeder Aussage und Regel der DB wird so eine solche subjektive Wahrscheinlichkeit zugeordnet. Um in einem Inferenzprozeß zu aussagekräftigen Wahrscheinlichkeiten für abzuleitende Konsequenzen zu gelangen, ist es notwendig, alle diese Wahrscheinlichkeiten über ein einheitliches Wahrscheinlichkeitsmaß zu modellieren. Ein solches Wahrscheinlichkeitsmaß erfüllt die folgenden Gesetze [DP88b](*Axiomensystem von Kolmogoroff*):

- (1)  $\forall(A \subseteq S): 0 \leq P(A) \leq 1$
- (2)  $P(S) = 1$
- (3)  $P(\emptyset) = 0$
- (4)  $P(A \cup B) = P(A) + P(B) - P(A \cap B), A, B \subseteq S,$

hierbei stellt  $S$  die Grundmenge der möglichen Aussagen dar.

In [Paa88] wird ein Verfahren zur Konstruktion eines solchen Wahrscheinlichkeitsmaßes beschrieben, das die obigen Axiome erfüllt und sich auf die sogenannten *Axiome rationalen Verhaltens* stützt. Aufgrund einiger vereinfachender Annahmen, wie beispielsweise die Exklusivität elementarer Aussagen, vereinfacht sich Axiom (4) zu  $P(A \cup B) = P(A) + P(B)$ .

Zur logischen Inferenz innerhalb der Probabilistischen Logik werden Implikationen der Form  $A \rightarrow B$  als bedingte Wahrscheinlichkeiten  $P_A(B)$  interpretiert, was bedeutet, daß der Konsequenz  $B$  eine Eintrittswahrscheinlichkeit in Abhängigkeit von der Eintrittswahrscheinlichkeit  $A$  zugeordnet wird. In komplizierteren Fällen findet der *Satz von Bayes* Anwendung. Auch diese Formel verwendet bedingte Wahrscheinlichkeiten.

Angenommen, die DB enthält eine Regel der Form  $F_1 = „A“$  gilt mit Wahrscheinlichkeit  $\pi_1$  und eine Regel der Form  $F_2 = „A \rightarrow B“$  gilt mit Wahrscheinlichkeit  $\pi_2$ . In der klassischen zweiwertigen Logik wäre diese Implikation wahr, wenn  $(A \wedge B) \vee \neg A$  gelten würde. Wegen der Mehrwertigkeit der Probabilistischen Logik kann diese eindeutige Zuordnung von „wahr“ oder „nicht wahr“ zu einer Aussage oder Regel nicht erfolgen. Die subjektiven Wahrscheinlichkeitsgrade, zu denen eine Aussage als wahr bzw. eine Regel als anwendbar angesehen werden kann, sind also aus dem Intervall  $[0, 1]$ . Damit berechnet sich die Wahrscheinlichkeit  $\pi_2$ , daß  $B$  unter Voraussetzung von  $A$  als wahr angesehen werden kann, nach der folgenden Formel für bedingte Wahrscheinlichkeiten:

$$\pi_2 = P_A(B) = \frac{P(A \wedge B)}{P(A)}.$$

Die Wahrscheinlichkeiten  $P(A)$  und  $P(A \wedge B)$  sind wegen des zuvor bestimmten Wahrscheinlichkeitsmaßes bekannt bzw. können daraus berechnet werden.

## 2.4 Sicherheitsfaktoren

Das Konzept der Sicherheitsfaktoren (CF) [KL97] stellt eine relativ informelle Möglichkeit zur Verfügung, mit Hilfe von Heuristiken logisch zu schließen. Das Verwenden von Heuristiken ist immer dann sinnvoll, wenn Expertenwissen eingesetzt werden muß, das auf Erfahrung und Schätzungen basiert und nicht anhand von Tests überprüfbar ist.

Sicherheitsfaktoren werden in vielen Fällen den bedingten Wahrscheinlichkeiten der Probabilistischen Logik vorgezogen, da diese Wahrscheinlichkeiten a priori schwer abschätzbar sind. Das Expertenwissen wird anhand von Regeln der Form *Hinweis*  $\rightarrow$  *Ergebnis* modelliert.

Regeln und Aussagen werden zwei Werte, nämlich ein *Glaubwürdigkeitsmaß*  $\mu_B$  (Measure of Belief) und ein *Fragwürdigkeitsmaß*  $\mu_D$  (Measure of Disbelief) zugeordnet, aus denen sich der sogenannte Sicherheitsfaktor bzw. Konfidenz zusammensetzt. Das Glaubwürdigkeitsmaß  $\mu_B$  gibt an, wie stark ein Hinweis H auf ein Ergebnis E hinweist, während das Fragwürdigkeitsmaß  $\mu_D$  darüber Aufschluß gibt, inwieweit ein Hinweis H gegen das Eintreten des Ereignisses E spricht. Der Sicherheitsfaktor  $CF(E|H)$  für eine Regel  $H \rightarrow E$  setzt sich folgendermaßen zusammen:

$$CF(H|E) = \mu_B(E|H) - \mu_D(E|H)$$

und wird als Maß für die Gültigkeit einer Regel der Form

IF H THEN E

angesehen.

Die möglichen Werte des Sicherheitsfaktors liegen im Intervall  $[-1,1]$ , wobei ein Sicherheitsfaktor mit einem Wert größer als Null als Zustimmung, ein Wert von Null als Unwissen und ein Wert kleiner als Null als Ablehnung interpretiert wird.

Ist H aus mehreren Hinweisen  $H_i$  zusammengesetzt, werden die Konfidenzen der einzelnen konjunktiv oder disjunktiv verknüpften Hinweise  $H_i$  mit Hilfe der Minimum- bzw. der Maximum-Operation wie folgt verrechnet:

$$CF(H_1 \wedge H_2) = \min(CF(H_1), CF(H_2))$$

bzw.

$$CF(H_1 \vee H_2) = \max(CF(H_1), CF(H_2)).$$

Führen mehrere Hinweise zum selben Ergebnis, wie zum Beispiel die Regeln IF  $H_1$  THEN E und IF  $H_2$  THEN E, werden die Sicherheitsfaktoren  $CF(E|H_1)$  und  $CF(E|H_2)$  zu einem Sicherheitsfaktor  $CF(E|H_1 \wedge H_2)$  kombiniert:

$$CF(E|H_1 \wedge H_2) = \begin{cases} CF_1 + CF_2 - (CF_1 \cdot CF_2), & \text{falls } CF_1 > CF_2 \\ CF_1 + CF_2 + (CF_1 \cdot CF_2), & \text{falls } CF_2 > CF_1 \\ \frac{CF_1 + CF_2}{1 - \min(|CF_1|, |CF_2|)}, & \text{sonst} \end{cases}$$

Hierbei steht  $CF_1$  für  $CF(E|H_1)$  und  $CF_2$  für  $CF(E|H_2)$ .

Der Term  $CF_2 + (CF_1 \cdot CF_2)$  drückt dabei den Beitrag eines neuen Hinweises  $H_2$  zum Sicherheitsfaktor des Gesamtergebnisses aus.

Ein weiterer Ansatz, der in der Literatur zum Konzept der Sicherheitsfaktoren zu finden ist, interpretiert Konfidenzen etwas anders. In [Rol89] gilt das Prinzip, daß ein Hinweis  $H$  für ein Ergebnis  $E$  entweder nur Zustimmung oder nur Ablehnung ausdrückt. Das bedeutet, daß für eine Regel IF  $H$  THEN  $E$  entweder nur das Glaubwürdigkeitsmaß  $\mu_B$  oder nur das Fragwürdigkeitsmaß  $\mu_D$  größer als Null sein darf, der andere Wert muß dann gleich Null sein. Dadurch ändert sich auch die Interpretation des Sicherheitsfaktors. Hier wird ein Sicherheitsfaktor von Eins als Zustimmung und ein Sicherheitsfaktor von Null als Ablehnung interpretiert. Die Berechnung des Sicherheitsfaktors einer Regel als Differenz des Glaubwürdigkeitsmaßes und des Fragwürdigkeitsmaßes bleibt erhalten, lediglich die Gleichung zur Berechnung des Sicherheitsfaktors bei Regeln mit unterschiedlichen Hinweisen aber gleichem Ergebnis ist anzupassen.

Die Probleme dieses Ansatzes, die weiter unten im Vergleich zu anderen Ansätzen zum logischen Schließen unter Unsicherheit aufgegriffen werden, sind jedoch mit den Problemen des oben beschriebenen Konzepts der Sicherheitsfaktoren vergleichbar.

## 2.5 Fuzzy-Logik

Die Fuzzy-Logik [Zad75, KL97, Gr95, DP88] baut auf das in [Zad75] erstmals vorgestellte Konzept der Fuzzy-Mengen auf. Sie wurde entwickelt, um die Verarbeitung von unsicherem, unpräzisen Wissen zu ermöglichen, wie es von Menschen zum Schlußfolgern benutzt wird. Ein wichtiges Problemfeld, das mit der Fuzzy-Logik erfaßt werden soll, ist der Zielkonflikt zwischen hoher Präzision und hoher Komplexität. Sie kann überall dort angewendet werden, wo eine Problemlösung weniger genaue aber dennoch ausreichende Ergebnisse zur Minderung der Komplexität des Problemlöseverfahrens hinnehmen kann [KL97]. In Zadehs Papier wird eine Fuzzy-Menge als eine Klasse von Elementen mit kontinuierlichen Zugehörigkeitsgraden angesehen.

Dinge werden vom Menschen naturgemäß in Klassen eingeteilt, deren Bedeutung gut verstanden wird, deren Grenzen jedoch nicht exakt festlegbar sind. Jedem Element einer Klasse ist ein Zugehörigkeitsgrad zwischen 0 und 1 zugeordnet. Damit wird die strenge Zweiwertigkeit der klassischen Mengentheorie aufgehoben. Die Zugehörigkeitsgrade der Elemente einer solchen Fuzzy Menge drücken jedoch keine Wahrscheinlichkeiten, sondern subjektive Empfindungen des jeweiligen Menschen aus. Beispielsweise wird ein „großer“ Mensch von 1,90 m einen anderen mit einer Größe von 1,70 m eher als „klein“ empfinden, als dies ein Mensch von 1,60 m

tun würde. Die Klasseneinteilung ist also in höchstem Maße subjektiv. Gerade diese Eigenschaft macht es sehr schwer, exakte Mengendefinitionen für von Menschen verwendete Konzepte und Klassifikationen aufzustellen – sie sind unscharf.

Die Zugehörigkeitsgrade werden den Elementen einer Fuzzy-Menge  $A$  entweder explizit in Form einer Wertetabelle oder mit Hilfe einer *Zugehörigkeitsfunktion*  $\mu_A$  zugeordnet. Diese kann je nach Anwendungsfall die unterschiedlichsten Formen haben [Gr95].

Eng verknüpft mit der Fuzzy-Theorie ist das Konzept der sogenannten *linguistischen Variablen*, wie beispielsweise „sehr“, „ziemlich“, „wenig“ etc. zur Modifikation von Fuzzy-Werten. Diese Variablen können als Operatoren zur Beeinflussung von Wahrheitswerten angewendet werden.

Zur Verknüpfung von Fuzzy-Mengen müssen die herkömmlichen mengentheoretischen Gesetze an das Konzept der Fuzzy-Mengen angepaßt werden. Verknüpfungen sind auf Fuzzy-Mengen demnach folgendermaßen definiert [KL97, DP88]:

$$\text{Vereinigung: } C = A \cup B \quad \mu_C(x) = \max(\mu_A(x), \mu_B(x))$$

$$\text{Schnittbildung: } C = A \cap B \quad \mu_C(x) = \min(\mu_A(x), \mu_B(x))$$

$$\text{Komplement: } \bar{A}: \quad \mu_{\bar{A}}(x) = 1 - \mu_A(x).$$

Für Fuzzy-Mengen gelten viele einfache Rechenregeln, wie die Kommutativität, Assoziativität oder beispielsweise die Distributivität. Auch das Gesetz von DeMorgan ist auf Fuzzy-Mengen anwendbar [KL97]. Es fehlen jedoch andere wichtige Eigenschaften der klassischen Mengenlehre. Zum Beispiel können für das Komplement  $\neg A$  von  $A$  gelten [KL97]:

$$A \cup \neg A \neq X, \text{ da } \mu_{A \cup \neg A}(x) = \max(\mu_A(x), 1 - \mu_A(x))$$

und

$$A \cap \neg A \neq \emptyset, \text{ da } \mu_{A \cap \neg A}(x) = \min(\mu_A(x), 1 - \mu_A(x)).$$

Eine Fuzzy-Inferenz wird üblicherweise auf Grundlage eines erweiterten *Modus Ponens* auf numerischen Wahrheitswerten der Aussagen aus dem Intervall  $[0,1]$  durchgeführt [KL97, DLP94, DP88]. Das Schema zur Ermittlung einer Konsequenz aus gegebenen Prämissen nach dem *Modus Ponens* hat das folgende Aussehen:

$$A' \wedge (A \rightarrow B) \quad \rightarrow B''.$$

Hierbei sind  $A \rightarrow B$  eine unscharfe Implikation,  $A'$  die Prämisse und  $B''$  die Konsequenz, denen je ein Fuzzy-Wert zugeordnet ist, der als Wahrheitswert zu interpretieren ist. Der Wahrheitswert einer Aussage gibt den Zugehörigkeitsgrad dieser Aussage zur Menge der *wahren Aussagen* an, der Wahrheitswert einer Implikation hingegen, zu welchem Grad aus der Wahrheit einer Prämisse auf die Konsequenz geschlossen werden kann.

Das Problem des approximativen Schließens auf Grundlage dieser Regel besteht nun darin, aus den Wahrheitswerten für die Prämisse und die Implikation einen Fuzzy-Wert für die Konse-

quenz zu ermitteln. Dazu sind die Begriffe *Fuzzy-Relation* und *Fuzzy-Verkettung* hilfreich. Eine binäre *Fuzzy-Relation*  $R$  über den Grundmengen  $X$  und  $Y$  ist nach Zadeh eine Fuzzy-Menge im Produktraum

$$X \times Y: R \subseteq \{(x,y), \mu_R(x,y) | x \in X, y \in Y\}.$$

Die Zugehörigkeitsfunktion  $\mu_R$  bzgl. der Fuzzy-Mengen  $A$  und  $B$  kann als Abbildung  $\mu_R: X \times Y \rightarrow [0,1]$  wie folgt definiert werden:

$$\mu_R(x,y) = \min(\mu_A(x), \mu_B(y)).$$

Die *Verkettung*  $A \circ R$  einer Fuzzy-Menge  $A$  mit einer binären Relation  $R$  ergibt eine neue einstellige Relation  $B = A \circ R = \{b | a \in A, (a,b) \in R\}$  mit

$$\mu_B(b) = \sup_{\forall a \in A} \{\min(\mu_A(a), \mu_R(a,b))\}.$$

Diese Verkettung hilft bei der Beschreibung der logischen Inferenz auf Grundlage unscharfer Implikationen der Form  $A \rightarrow B$ , da zu einer solchen Implikation die Fuzzy-Relation  $R: A \rightarrow B$  gebildet werden kann. Eine Fuzzy-Regel kann also durch eine Fuzzy-Relation dargestellt werden.

Eine Schlußfolgerung nach dem Modus Ponens wird nun als Verkettung zweier Relationen betrachtet, nämlich der Relationen  $B'' = A' \circ R$  mit  $R: A \rightarrow B$ . Die Berechnung eines Fuzzy-Wertes für  $B''$  erfolgt somit nach der folgenden Gleichung:

$$\mu_{B''}(x) = \sup_{\forall a \in A} (\mu_{A'}(x) \wedge \mu_R(x,y)).$$

In der Literatur [Gr95] sind noch weitere mögliche Regeln zur Schlußfolgerung mit Fuzzy-Wissen zu finden, die von unterschiedlichen Definitionen der Relation  $R_{A \rightarrow B}$  herrühren. Diese werden an dieser Stelle nicht weiter verfolgt.

## 2.6 Possibilistische Logik

Die Possibilistische Logik [DLP94, She92, FP92] ist ein weiteres Beispiel, wie mit unsicherem und unvollständigem Wissen logisch geschlossen werden kann. Die Possibilistische Logik arbeitet mit Formeln und Aussagen der Logik erster Ordnung [DRSW86], denen zusätzlich sogenannte *Möglichkeitsmaße* oder *Notwendigkeitsmaße* als numerische Werte aus dem Intervall  $[0,1]$  zugeordnet werden. In [DLP94] wird das Möglichkeitsmaß  $\Pi(A)$  einer Teilmenge  $A$  einer Grundmenge  $U$  als der Grad interpretiert, zu dem *ein* Wert  $u$  aus  $A$  existiert, der als Wert einer Variable  $X$  in Frage kommt, während das Notwendigkeitsmaß  $N(A)$  angibt, zu welchem Grad *alle* möglichen Werte der Variablen  $X$  zu  $A$  gehören.

Notwendigkeits- und Möglichkeitsmaße leiten sich aus *Möglichkeitsverteilungen* ab, die eng mit dem Konzept der Fuzzy-Mengen verwandt sind. In [Zad78] wird eine Herleitung der Pos-

sibilistischen Theorie auf Grundlage von Fuzzy-Mengen vorgestellt. Möglichkeitsverteilungen sind demnach Zugehörigkeitsfunktionen von Fuzzy-Mengen, die die möglichen Werte einer Fuzzy-Variablen beschreiben. In diesem Sinne fungieren nach Zadeh Fuzzy-Mengen als Beschränkungen für die möglichen Werte, die eine Variable  $X$  annehmen kann, und die Zugehörigkeitsfunktion dieser Fuzzy-Menge heißt *Möglichkeitsverteilungsfunktion*.

Aus dieser neuen Sichtweise der Fuzzy-Mengen resultiert eine etwas andere Interpretation der *Möglichkeitsverteilungsfunktion* als Zugehörigkeitsfunktion: wurde im Sinne der klassischen Zugehörigkeitsfunktion ein Zugehörigkeitsgrad des Wertes 28 zu einer Fuzzy-Menge *jung* über dem Intervall  $[0,1]$  mit 0.7 bewertet, bedeutet dieser Sachverhalt, daß das Alter 28 mit dem Konzept *jung* zum Grad 0.7 *kompatibel* ist. Wird dagegen eine Aussage der Form „ $X$  ist  $F$ “ betrachtet, z.B. die Aussage „Frieda ist jung“, wird der Wert 0.7 nicht mehr als Kompatibilitätsgrad angesehen, sondern konvertiert zu einem Möglichkeitsgrad (degree of possibility), daß Frieda 28 Jahre alt ist, wenn die Aussage „Frieda ist jung“ gegeben ist. Die Zugehörigkeitsfunktion  $\mu_F$  einer Fuzzy-Menge  $F$  und die Möglichkeitsverteilung  $\pi_X$  einer Aussage „ $X$  ist  $F$ “, in der die Fuzzy-Menge als Werte-Beschränkung für  $X$  fungiert, werden somit als numerisch gleich angesehen, während die Interpretation wechselt. Die Möglichkeitsverteilungsfunktion über eine Variable  $X$  wird deshalb mit einem anderen Symbol als die Zugehörigkeitsfunktion einer Fuzzy-Menge dargestellt, nämlich mit  $\pi_X$ .

Für Möglichkeitsverteilungen wurden die folgenden Konventionen festgelegt:

- $\pi_X(u) = 1$  für genau ein  $u$  aus  $U$  (Normalisierung)
- $\pi_X(u) = 0$  bedeutet, daß  $x = u$  unmöglich ist,
- $\pi_X(u) = 1$  bedeutet, daß  $x = u$  vollständig erlaubt ist, und
- $\pi_X(u) > \pi_X(u')$  bedeutet, daß  $x = u$  der Belegung  $x = u'$  vorzuziehen ist.

Eine vage Information definiert somit eine implizite Ordnung auf den möglichen Fakten, auf die sie sich bezieht. Diese Ordnung wird mit Hilfe einer Zugehörigkeitsfunktion kodiert.

Aus der Möglichkeitsverteilung  $\pi_X$  sind zwei Maße ableitbar, die jeder Teilmenge  $A$  einer Grundmenge  $U$  Werte aus dem Intervall  $[0,1]$  zuweisen: das *Möglichkeitsmaß*  $\Pi(A)$  und das *Notwendigkeitsmaß*  $N(A)$ . Diese Maße sind wie folgt definiert:

$$\Pi(A) = \sup_{u \in A} \pi_X(u) \text{ und}$$

$$N(A) = 1 - \Pi(\bar{A}) = \inf_{u \in \bar{A}} 1 - \pi_X(u), \text{ wobei } \bar{A} \text{ das Komplement von } A \text{ ist.}$$

Mit Hilfe des Möglichkeitsmaßes  $\Pi(A)$  können Aussagen der Form „ $X$  ist  $A$  ist möglich“ ausgedrückt werden, insbesondere aber lassen sich mit dem Notwendigkeitsmaß  $N(A)$  Aussagen der Form „ $X$  ist  $A$  ist —-sicher“ formulieren. Das Notwendigkeitsmaß  $N(A)$  kann somit als untere Schranke für die Sicherheit der Aussage „ $X$  ist  $A$ “ angesehen werden.

Für Möglichkeitsmaße gelten die folgenden Basisaxiome:

- (1)  $\Pi(\emptyset) = 0$ ;  $\Pi(U) = 1$
- (2)  $\Pi(A_1 \cap A_2) = \min(\Pi(A_1), \Pi(A_2))$

Möglichkeits- und Notwendigkeitsmaße haben zusätzlich die folgenden Eigenschaften:

- (1)  $N(A \cup B) = \min(N(A), N(B)); \quad N(A \cap B) \geq \max(N(A), N(B));$
- (2)  $\Pi(A \cup B) = \max(\Pi(A), \Pi(B)); \quad \Pi(A \cap B) \leq \min(\Pi(A), \Pi(B)),$
- (3)  $\min(N(A), N(\bar{A})) = 0,$
- (4)  $\max(\Pi(A), \Pi(\bar{A})) = 1,$

In [DLP94] wird die Sprache PL1 für notwendigkeitsbewertete Formeln erster Ordnung angegeben. Die Autoren beschränken sich hierbei auf Formeln der Form  $(\varphi \alpha)$ . Diese sind mit Notwendigkeitsmaßen bewertet, da es mit diesen Maßen möglich ist, eine Präferenz-Ordnung auf Formeln zu definieren. Mit  $(\varphi \alpha)$  wird ausgedrückt, daß eine Formel  $\varphi$  in der Logik erster Ordnung mindestens zum Grad  $\alpha$  sicher ist, d.h.  $N(A) \geq \alpha$ . Je größer die Bewertung  $\alpha$  einer Formel  $\varphi$  also ist, desto sicherer ist sie.

Sei nun  $F$  die Menge aller Possibilistischen (notwendigkeitsbewerteten) Formeln der Form  $(\varphi \alpha)$  und  $\Omega$  die Menge aller möglichen Interpretationen von  $F$ , dann ist die Semantik einer Menge von Formeln durch die Teilmenge der Interpretationen aus  $\Omega$  bestimmt, die  $F$  erfüllen. Jede dieser Interpretationen wird *Modell* genannt.

Im Fall der Possibilistischen notwendigkeitsbewerteten Logik wird eine Möglichkeitsverteilungsfunktion über  $\Omega$  betrachtet, die die Fuzzy-Menge der Modelle von  $F$  darstellt. Das bedeutet, daß  $F$  eine Präferenz-Ordnung über  $\Omega$  beschreibt, die durch eine Möglichkeitsverteilung kodiert ist. In diesem Sinne ist das Möglichkeitsmaß  $\Pi(\varphi)$  definiert als

$$\Pi(\varphi) = \sup\{\pi(\omega), \omega \models \varphi\},$$

wobei  $\omega \models \varphi$  bedeutet, daß  $\omega$  ein Modell von  $\varphi$  ist. Das duale Notwendigkeitsmaß  $N$  berechnet sich dann zu:

$$N(\varphi) = 1 - \Pi(\neg\varphi) = \inf\{1 - \pi(\omega), \omega \models \varphi\}.$$

Wird die *Normalisierungsbedingung*  $\sup\{\pi(\omega), \omega \models \varphi\} = 1$  aufgehoben, zeigt das Notwendigkeitsmaß ein etwas modifiziertes Verhalten zur herkömmlichen Possibilistischen Theorie, denn wenn  $1 - \alpha_\pi = \sup\{\pi(\omega), \omega \models \varphi\} < 1$  erlaubt ist, gilt die Regel

$$\min(N(\varphi), N(\neg\varphi)) = \alpha_\pi > 0.$$

Diese Regel definiert ein wichtiges Inkonsistenzmaß *incons*, auf das in Kapitel 4.3 noch einmal eingegangen wird.

Zur Beschreibung der Deduktion mit Hilfe der Possibilistischen Logik sind die folgenden Definitionen hilfreich:

Eine Möglichkeitsverteilungsfunktion  $\pi$  über  $\Omega$  *erfüllt* eine notwendigkeitsbewertete Formel  $(\varphi \alpha)$ , gdw.  $N(\varphi) \geq \alpha$ , wobei  $N$  das über  $\pi$  definierte Notwendigkeitsmaß ist.

Eine Möglichkeitsverteilungsfunktion  $\pi$  über  $\Omega$  *erfüllt* eine Menge Possibilistischer For-

meln  $F = \{(\varphi_i \alpha_i), i = 1, \dots, n\}$ , gdw.  $\pi$  jede einzelne possibilistische Formel  $(\varphi_i \alpha_i)$  aus  $F$  erfüllt.

Eine possibilistische Formel  $\Phi$  heißt *logische Konsequenz* der Menge der possibilistischen Formeln  $F$ , gdw. jede Möglichkeitsverteilung, die  $F$  erfüllt, auch  $\Phi$  erfüllt.

Das Deduktions-Problem stellt sich in der Possibilistischen Logik dann folgendermaßen dar: soll eine Formel  $\varphi$  aus einer gegebenen Menge von Formeln  $F$  geschlossen werden, wird die beste Bewertung  $\alpha$ , also die beste bekannte untere Schranke des Notwendigkeitsgrades so berechnet, daß  $\varphi$  die logische Konsequenz von  $F$  ist.

Formal läßt sich diese Deduktion als

$$\text{Val}(\varphi, F) = \sup\{\alpha \mid [0,1], F \models (\varphi \alpha)\}$$

schreiben.

## 2.7 Bewertung

Dieses Kapitel zeigt einige der vielen Ansätze zu logischem Schließen unter Unsicherheit, die in der Literatur zu finden sind. Doch nicht alle der hier vorgestellten Konzepte sind für die Logik der Inferenzmaschine geeignet. Wie das Szenario aus Kapitel 1 bereits gezeigt hat, ist es zur Vermeidung von Widersprüchen in der Semantik einer zu analysierenden relationalen Datenbank notwendig, Indikatoren zur Herleitung der semantischen Informationen unterschiedlich zu gewichten, wie es beispielsweise bei der Herleitung der Schlüsseleigenschaft oder Nicht-Schlüsseleigenschaft von Attributen geschehen ist. Das Szenario aus Kapitel 1.2 führt dazu zwei Heuristiken an: Implikation  $i_1$  drückt dort mit einer Konfidenz von 0.7 aus, daß ein *cyclic-join* Cliché ein Indikator für eine Schlüsseleigenschaft der Attributmengemenge  $v$  ist, Implikation  $i_2$  hingegen spezifiziert mit Konfidenz 0.3, daß ein *select-distinct* Cliché der Schlüsseleigenschaft einer Attributmengemenge  $v$  widerspricht. Für den Fall, daß bei der Analyse einer Datenbank die beiden Attributmengemengen aus Implikation  $i_1$  und  $i_2$  gleiche Attribute enthalten, läge ein Widerspruch bezüglich der Schlüsseleigenschaft der Attribute in  $v$  vor. Durch die unterschiedliche Gewichtung der Implikationen im GFRN kann dieser Widerspruch abgeschwächt werden. Das *select-distinct* Cliché ist hier nur mit einer Konfidenz von 0.3 gewichtet. Dieser Wert drückt eine relativ geringe subjektive Annahme des Reengineers aus, daß ein *select-distinct* Cliché tatsächlich gegen die Schlüsseleigenschaft der Attribute in  $v$  spricht, da die Aussagekraft dieser Heuristik stark vom Programmierstil des Anwendungsentwicklers abhängt, der das *select-distinct* Cliché programmiert hat<sup>1</sup>.

Durch die unterschiedliche Gewichtung kann also eine sinnvolle Bewertung der gefundenen Indikatoren vorgenommen werden. Um dieses zu ermöglichen, ist jedoch eine mehrwertige Logik notwendig, bei der der „Grad der Unsicherheit“ einer Heuristik quantifiziert werden kann. Deshalb sind die symbolischen Ansätze, wie die oben beschriebene Modale Logik oder auch die Anfechtbare Logik für die Inferenzmaschine nicht geeignet.

Aber auch einige der numerischen Ansätze, die oben beschrieben worden sind, weisen Eigen-

---

1. vgl. Kapitel 3.3.

schaften auf, die gegen eine Anwendung für die Inferenzmaschine sprechen. So ist die Probabilistische Logik zwar ein weitverbreiteter Ansatz zum logischen Schließen unter Unsicherheit, die Wahrscheinlichkeiten für das Zutreffen von Aussagen und Regeln müssen allerdings von Experten festgelegt werden. Dies geschieht typischerweise auf Grundlage einer Theorie, durch Schätzungen oder Erfahrungen. Diese Wahrscheinlichkeiten sind meist fehlerbehaftet [Paa88], und diese Fehler können sich bei additiver Verrechnung verstärken [DP88]. Soll die Probabilistische Logik trotzdem eingesetzt werden, muß der Benutzer die Reliabilität der Expertenschätzungen bewerten und eine Fehlerabschätzung durchführen [Paa88]. Die Inferenz in der Probabilistischen Logik wird mit Hilfe des „Satzes von Bayes“ durchgeführt, der bedingte Wahrscheinlichkeiten verwendet. Auch diese bedingten Wahrscheinlichkeiten sind schwer zu schätzen, so daß die Inferenz u.U. zu wenig aussagekräftigen Ergebnissen führt [Rol89]. Außerdem existieren in der Probabilistischen Logik keine Konventionen, wie die Unwissenheit über die Wahrheit einer Aussage modelliert werden soll. Man hat lediglich die Möglichkeit, obere und untere Schranken anzugeben, wie das auch in Possibilistischen Logik getan wird [DP88]. Aber noch ein weiterer wichtiger Punkt spricht gegen eine Anwendung der Probabilistischen Logik für die Inferenzmaschine: in der Probabilistischen Logik kann die Unsicherheit über Aussagen mit Hilfe von Wahrscheinlichkeitsgraden nicht adäquat ausgedrückt werden [DP88]. Unwissenheit wird vielmehr als Zufälligkeit interpretiert, bei der alle Ereignisse als gleich wahrscheinlich angesehen werden.

Auch bei dem Konzept der Sicherheitsfaktoren treten Probleme auf, die den Einsatz für die Inferenzmaschine nicht ratsam erscheinen lassen. Diese Probleme stehen in Zusammenhang mit der Unabhängigkeit von Aussagen und der Rekursion in Ableitungsregeln [KL97]. Wird beispielsweise eine Aussage aus zwei scheinbar unabhängigen Regeln mit gleicher Prämisse hergeleitet, führt dies zu einer unerwünschten Erhöhung des Sicherheitsfaktors. Auch Rekursionen der Form  $A \rightarrow B$  und  $B \rightarrow A$  liefern Probleme. Diese führen bei der Inferenz zu einer stetigen Erhöhung des Sicherheitsfaktors, bis dieser den Wert 1 erreicht. Das Ergebnis ist somit nicht aussagekräftig. Außerdem führen Rekursionen in Regeln zu Zyklen in dem für die Inferenzmaschine gewählten unscharfen Petrinetzmodell. Dies hat zur Folge, daß ein zur Analyse verwendetes unscharfes Petrinetz keine Zyklen enthalten dürfte, die aber wiederum für nichtmonotones Schließen notwendig sind.

Eine Logik, die sich an die menschlichen Denkweisen anlehnt, ist die Fuzzy-Logik. Sie findet dort Anwendung, wo bezüglich der Präzision eines Ergebnisses Abzüge hingenommen werden können, um dadurch die Komplexität eines Problemlöseverfahrens zu vermindern. Die Fuzzy-Logik spricht damit einen anderen Problemkreis als die Probabilistische und auch die Possibilistische Logik an: hier geht es nicht so sehr um die Unwissenheit über reale Fakten, sondern eher um die Unschärfe von Randbedingungen, die eine Schlußfolgerung steuern [DP88]. Die Regeln zur Ableitung von Informationen sind hierbei scharf, die Prädikate jedoch unscharf. Das generische Reengineering-Wissen, das für die Analyse einer RDBA in einem GFRN spezifiziert wird<sup>1</sup>, hat jedoch eine andere Qualität. Die GFRN-Implikationen stellen nämlich größtenteils Heuristiken dar und auch die Fakten, die während der Inferenz abgeleitet werden, können bis zum Ende der Analyse unsicher sein. Aus diesen Gründen erweist sich auch die Fuzzy-Logik für die Logik der Inferenzmaschine als nicht geeignet.

---

1. vgl. Kapitel 3.

Die Possibilistische Logik zeigt die oben aufgezählten Nachteile der Probabilistischen Logik und der Sicherheitsfaktoren nicht. Selbst Fehler, die sich in der Probabilistischen Logik durch die additive Verknüpfung verstärken, bleiben in der Possibilistischen Logik konstant [DP88]. Außerdem stellt sich die Inferenz in der Possibilistischen Logik als einfach dar, denn sie wird nur über die Minimum- und Maximumoperation durchgeführt. Diese einfachen Berechnungsvorschriften erlauben beim Einsatz in der Inferenzmaschine eine zeitsparende Ausführung und unterstützen damit die Forderung nach der Effizienz<sup>1</sup> der Inferenzmaschine. Auch die Interpretation der Notwendigkeit, die einer Aussage in diesem Logikkalkül zugewiesen wird, ist leicht verständlich: die Notwendigkeit  $N(p)$  einer Aussage  $p$  gibt eine untere Schranke für die Sicherheit dieser Aussage an. Über abgestufte Notwendigkeiten aus dem Intervall  $[0,1]$  ist es zudem möglich, Unsicherheiten im Wissen, das zur Inferenz verwendet wird, zu berücksichtigen. Diese Tatsachen sind der Grund dafür, daß die Possibilistische Logik als Logik für die Inferenzmaschine angewendet wird.

---

1. vgl. Kapitel 1.4, Anforderung 3.



### 3 Generic Fuzzy Reasoning Nets

Die Zielsetzung dieser Diplomarbeit ist es, einen Ausführungsmechanismus für generische Wissensspezifikationen über relationale Datenbanken zu implementieren<sup>1</sup>. Diese Wissensspezifikation erfolgt mit Hilfe der bereits in Kapitel 1 kurz erwähnten *Generic Fuzzy Reasoning Nets (GFRN)*. Aus den Eigenschaften des Wissens, das in die Analyse einer relationalen Datenbank eingeht, sowie aus den Eigenschaften der GFRNs, resultieren Anforderungen an die Inferenzmaschine, die schon in Kapitel 1.4 aufgeführt sind. Um diese Anforderungen für die Auswahl eines geeigneten Modells für die Ausführungsmaschine konkretisieren zu können, beschreibt das nun folgende Kapitel die Syntax und die Semantik der GFRNs, die erstmals in [JSZ97] vorgestellt wurden.

#### 3.1 Die Semantik einer einfachen GFRN-Regel am Beispiel

Abbildung 4 zeigt eine einfache Regel in GFRN-Notation. Diese Regel wurde schon im Eingangsbeispiel erwähnt und drückt den folgenden Sachverhalt aus:

*Falls* der Applikationscode ein `select_distinct` Statement über einer Menge von Attributen  $v_1$  enthält, *dann* stellen alle Teilmengen  $v_2$  von  $v_1$  mit Notwendigkeit  $\phi$  keine Schlüsselkandidaten dar.

Eine solche Regel besteht aus mehreren Komponenten. Die *Prädikate*, hier das `sel_dist` und das `key` Prädikat, sind als Ovale, *Implikationen* als rechteckige Boxen dargestellt. Die Implikationen können mit einer *Konfidenz*  $\phi$  gewichtet werden. Diese gibt an, zu welchem Grad die Regel als zutreffend angesehen wird und nimmt einen Wert zwischen 0 und 1 an. Prädikate und Implikationen sind in der GFRN-Notation durch *gerichtete Kanten* miteinander verbunden. Eine ausgefüllte Pfeilspitze meint hierbei die Negation einer Aussage. Diese Kanten können mit *formalen Parametern*, hier  $v_1$  und  $v_2$ , versehen werden, über die in den Implikationen *Bedingungen* – wie im Beispiel  $v_2 \subseteq v_1$  – formulierbar sind.

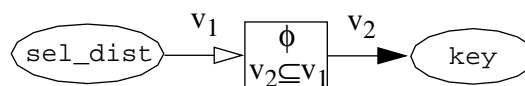


Abb. 4: Einfache Regel in GFRN-Notation

Formal entspricht die Semantik einer solchen Regel in GFRN-Notation gewichteten Formeln in der Prädikatenlogik erster Ordnung  $(F, \phi)$  [DRSW86]. Die Prädikatenlogik erster Ordnung ist eine Erweiterung der reinen Aussagenlogik, in der einfache Aussagen mit Hilfe der logischen Operatoren *NOT*, *AND* und *OR* zu logischen Formeln verknüpft werden. Zusätzlich stellt die Prädikatenlogik zum Aufbau von Formeln Quantoren, Prädikats- und Funktionssymbole zur Verfügung.

Jedes Prädikat in der GFRN-Notation entspricht einem Prädikatssymbol und jede Implikation der GFRN-Regel einem Paar  $F^* = (F, \phi)$  in der Prädikatenlogik, wobei die Gewichtung  $\phi$  der Formel der Konfidenz der GFRN-Regel entspricht. Alle formalen Parameter und Implikationen

1. vgl. Abschnitt 1.5.

in einer GFRN-Regel sind implizit allquantifiziert. Die in Abbildung 4 gezeigte GFRN-Regel läßt sich algorithmisch in die folgende Formel der Logik erster Ordnung übersetzen:

$$F:(\forall v_1)(\forall v_2 \subseteq v_1)(\text{sel\_dist}(v_1) \Rightarrow \neg \text{key}(v_2)).$$

Abschnitt 3.4 schlägt dazu einen Algorithmus zur Übersetzung einer GFRN-Spezifikation in eine Formel der Prädikatenlogik vor.

Ein wesentlicher Unterschied einer Wissensspezifikation mit GFRNs zu herkömmlichen regelbasierten Ansätzen ist, daß der GFRN-Ansatz die *Kontraposition* einer Implikation mitberücksichtigt. D.h. zu einer Regel *falls A dann B* wird auch die Regel *falls NOT B dann NOT A* angewendet. Diese Tatsache ermöglicht das in Abschnitt 1.4 geforderte nichtmonotone Schließen.

### 3.2 Axiome zur Klassifikation des initialen Wissens

Die Interpretation eines GFRNs erfolgt nach unterschiedlichen Gesichtspunkten. Hierbei sind der Zeitpunkt, zu dem Wissen abgeleitet werden kann, sowie die Qualität des initialen Wissens, das der Reengineer mit in die Analyse einbringt, entscheidend. Das Wissen des Reengineers über eine bestimmte Anwendung wird vor der Interpretation des zugehörigen GFRNs klassifiziert. Dies geschieht dadurch, daß die Prädikate in der GFRN-Spezifikation in drei unterschiedliche Typen unterteilt werden, die auch *Axiome* heißen. Man unterscheidet *starke*, *schwache* und *aufgeschobene* Axiome. Diese stellen die initiale und möglicherweise unvollständige Informationsmenge dar, die in die Analyse eingehen soll.

Die *starken Axiome* stellen unumstößliche Fakten dar und können während der Auswertung einer relationalen Datenbankanwendung nicht revidiert werden. An diese starken Axiome sind Analyseoperationen geknüpft, die vor der eigentlichen Inferenz bestimmte Eigenschaften überprüfen, z.B. im Applikationscode nach Clichés suchen oder die Ähnlichkeit von Attributnamen vergleichen<sup>1</sup>.

*Schwache Axiome* repräsentieren die vagen und möglicherweise falschen Annahmen des Reengineers, wie z.B. die Annahme über die Schlüsseleigenschaft eines bestimmten Attributs. Im Gegensatz zu den starken Axiomen können die initialen Werte für das Zutreffen der mit dem schwachen Axiom verknüpften Aussage während der Inferenz verändert und damit die zugehörige Aussage bestätigt oder widerlegt werden.

Die *aufgeschobenen Axiome* stellen dagegen Prädikate dar, an die meist sehr aufwendige Analyseoperationen gebunden sind. Diese sollen nur dann ausgeführt werden, wenn ein Hinweis auf die tatsächliche Existenz dieser einen bestimmten Eigenschaft vorliegt, die durch die Analyseoperation überprüft werden soll. Ein Beispiel für ein solches aufgeschobenes Axiom ist das Testen einer Teilmengenbeziehung zwischen Attributen, da hierfür zunächst ein Join über die Relationen ausgeführt werden muß, in denen die Attribute enthalten sind. Auch die aufgeschobenen Axiome gelten wie die starken Axiome als nicht revidierbar. Das liegt daran, daß die Fakten, die man aus den aufgeschobenen Analyseoperationen gewinnt, anhand des Codes, der Daten oder des Schemas abgeleitet werden und sich die Datenbank während der Analyse nicht

1. vgl. Abschnitt 1.2 zur Bedeutung von Clichés und der Ähnlichkeit von Attributnamen.

verändert. Die Ergebnisse der Analyseoperationen haben also während der gesamten Ermittlung der semantischen Informationen der Datenbank Gültigkeit.

Die Zuordnung der Prädikate zu den drei unterschiedlichen Axiomstypen kann nach der eigentlichen Spezifikation des Wissens in einem zweiten Schritt erfolgen und ermöglicht so eine Trennung von reiner Wissensspezifikation und einer schon vor der eigentlichen Inferenz möglichen manuellen Optimierung.

### 3.3 Eine GFRN-Spezifikation für das Eingangsszenario

Abbildung 5 zeigt einen Ausschnitt aus einer GFRN-Spezifikation für das Eingangsbeispiel. Diese Spezifikation verfolgt das Ziel, Fremdschlüsselbeziehungen zwischen den Relationen einer relationalen Datenbank Anwendung zu erkennen.

Die unterschiedlichen Axiomstypen werden in Abbildung 5 unterschiedlich dargestellt. Die starken Axiome **cycl\_join**, **sel\_dist** und **nsimilar** sind fett, die aufgeschobenen Axiome **validKey**, **validInd** und **tcomp** gestrichelt umrandet. Einfache Prädikate, wie **equiv.**, **IND** und **key** haben eine einfache Umrandung.

Implikation  $i_1$  modelliert die bereits in Abschnitt 1.2 aufgezeigte Heuristik, daß ein *cyclic\_join* Cliché (**cycl\_join**) über einer Menge von Attributen  $a$  mit Konfidenz 0.7 auf die Schlüsseleigenschaft (**key**) dieser Attribute hinweist. Die Konfidenz von 0.7 drückt die Erfahrung des Reengineers aus, daß ein *cyclic\_join* Cliché zwar ein starker Indikator, jedoch keine hinreichende Bedingung für die Schlüsseleigenschaft von Attributen ist. Eine Aussage läßt sich allerdings durch die Spezifikation weiterer Regeln bekräftigen oder widerlegen. Implikation  $i_3$  drückt beispielsweise aus, daß eine einmal angenommene Schlüsseleigenschaft (**key**) von Attributen mit absoluter Sicherheit, also einer Konfidenz von 1, widerlegt werden kann, wenn sich in der Datenbanksausprägung ein Gegenbeispiel für diese Eigenschaft (**validKey**) finden läßt.

Implikation  $i_2$  zeigt noch einmal die in Abbildung 4 aufgeführte GFRN-Regel, die besagt, daß ein *select\_distinct* Cliché (**sel\_dist**) über einer Attributmenge  $v_1$  mit Konfidenz 0.3 auf die Eigenschaft hinweist, daß alle Teilmengen  $v_2$  von  $v_1$  keine Schlüsselkandidaten (**key**) sind. Eine Verwendung des Schlüsselwortes *distinct* führt bei einer SQL-Anfrage, deren Ergebnisrelation mehrere gleiche Tupel enthält, zu einer Eliminierung der Duplikate. Bei manchen Anfragen läßt sich ein doppeltes Auftreten von Tupeln in der Ergebnisrelation von vornherein ausschließen, beispielsweise, wenn Tupel gesucht werden, in denen der Schlüssel enthalten ist. Verwendet man dort das Schlüsselwort *distinct*, hat dieses zwar keinen Einfluß auf die Ergebnisrelation, impliziert jedoch für die Auswertung der Datenbank, daß die Attribute der Ergebnisrelation wegen eines möglichen doppelten Vorkommens keine Schlüsselkandidaten sein können. Entsprechend der Erfahrung des Reengineers, daß Anwendungsprogrammierer nicht immer gewissenhaft mit der Verwendung des Schlüsselwortes *distinct* umgehen, wird Implikation  $i_2$  mit einer geringeren Konfidenz gewichtet als Implikation  $i_1$ .

Eine wichtige Heuristik, die der Erkennung von möglichen Fremdschlüsselbeziehungen zwischen Attributen von Relationen dient, ist das Vergleichen der Attributnamen der in Frage kommenden Attribute. Eine große Ähnlichkeit von Attributnamen (**nsimilar**) weist nämlich Implikation  $i_8$  entsprechend mit einer Konfidenz von 0.5 auf eine äquivalente Bedeutung

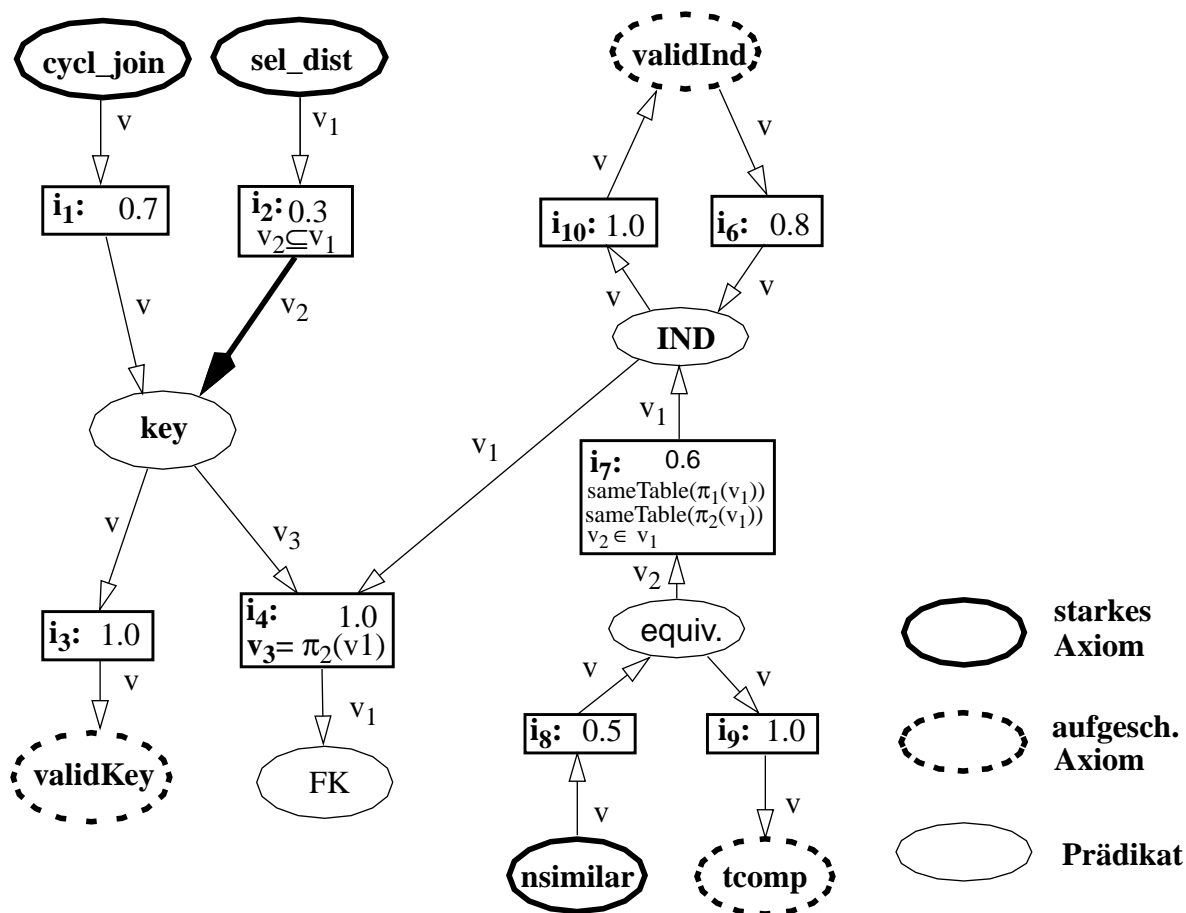


Abb. 5: Ausschnitt einer GFRN-Spezifikation für das Eingangsbeispiel

(**equiv.**) der Attribute hin. Diese Heuristik kann durch Implikation *i9* bestätigt oder widerlegt werden. Sie modelliert, daß Attribute mit äquivalenter Bedeutung (**equiv.**) mit einer Konfidenz von 1.0 zu einem gewissen Grad typkompatibel (**tcomp**) sein müssen.

Implikation *i10* besagt - ähnlich wie Implikation *i3* -, daß eine einmal festgestellte Teilmengenbeziehung (**IND**) zwischen zwei Attributmengen nur dann gelten kann, wenn die Daten der relationalen Datenbank kein Gegenbeispiel für eine solche Teilmengenbeziehung (**validIND**) enthalten. Die Gegenrichtung dieser Regel wird durch Implikation *i6* dargestellt: hier ist spezifiziert, daß eine in der momentanen Datenbanksausprägung zu findende Teilmengenbeziehung (**validIND**) zwischen den Daten zweier Tabellenspalten mit Konfidenz 0.8 auf eine tatsächliche, d.h. konzeptionelle Teilmengenbeziehung (**IND**) zwischen diesen Attributmengen hinweist.

Für Implikation *i10* ist zu beachten, daß die Analyseoperation, die an **validIND** geknüpft ist, die Größe der Datenbanksausprägungen mitberücksichtigen muß. Eine vorliegende Teilmengenbeziehung in einer Datenbank mit einer sehr großen Datenmenge weist mit einer größeren Konfidenz auf eine tatsächliche Teilmengenbeziehung hin, als in einer Datenbank mit einer kleinen Ausprägung.

Wie oben bereits erwähnt, weist eine äquivalente Bedeutung von Attributen derselben oder unterschiedlicher Tabellen auf Teilmengenbeziehungen zwischen diesen Attributen hin. Implikation  $i_7$  drückt diesen Sachverhalt für Attributpaare in der Menge  $v_2$  aus. Eine Teilmengenbeziehung (**IND**) liegt dann für die äquivalenten Attributpaare (**equiv.**) mit Konfidenz 0.6 vor, wenn alle Ergebnisattribute der Projektion auf das erste Attribut in einem Attributpaar aus  $v_2$  und alle Ergebnisattribute der Projektion auf das zweite Attribut im Attributpaar jeweils in derselben Tabelle vorkommen. Die Attributpaare aus  $v_2$ , die die *sameTable*-Bedingung<sup>1</sup> aus Implikation  $i_7$  erfüllen, werden in der Attributmenge  $v_1$  zusammengefaßt. Eine Teilmengenbeziehung existiert laut der Spezifikation im GFRN dann von der Tabelle, die das erste Attribut der Attributmenge enthält, zu der Tabelle, die das zweite Attribut des Attributpaares enthält.

Eine Fremdschlüsselbeziehung (**FK**) zwischen Tabellen läßt sich dann entsprechend der Implikation  $i_4$  aus einer Teilmengenbeziehung (**IND**) zwischen Attributen ableiten, die durch die Attributpaarmenge  $v_1$  repräsentiert wird. Hierbei darf die Ergebnisrelation  $v_3$  der Projektion auf das zweite Element jedes Attributpaares aus  $v_1$  allerdings nur Schlüsselkandidaten (**key**) enthalten.

### 3.4 Die formale Syntax und Semantik der GFRNs

Definition 1 zeigt die formale Beschreibung der hier vorgestellten Sprache der GFRNs.

#### Definition 1: Syntax der Generic Fuzzy Reasoning Nets

Ein Generic Fuzzy Reasoning Net ist ein 8-Tupel

$GFRN = \{P, I, E, F_r, F_b, cf, A^s, A^d\}$ , mit

$P = \{p_1, p_2, \dots, p_z\}$ : endliche Menge von *Prädikaten*

$I = \{i_1, i_2, \dots, i_m\}$ : endliche Menge von *Implikationen*, jede Menge ist ein Tupel

$i_g = (\iota_g, V_g, K_g)$  mit einem eindeutigen *Implikations-Identifizierer*  $\iota_g$ , einer

Menge von *Parametern*  $V_g = \{v_1, v_2, \dots, v_f\}$  und einer Menge von *Constraints*

$K_g = \{k_1, k_2, \dots, k_s\}$  über  $V_g$ , wobei  $k_x = (w, r, f^u, (w_1, w_2, \dots, w_u))$  mit

$w, w_1, w_2, \dots, w_u \in V_g, r \in \{\in, \subseteq, =\}$ , falls  $f^u \in F_r$  oder

$k_x = (f^u, (w_1, w_2, \dots, w_u))$ , falls  $f^u \in F_b$ .

$E = \{e_1, e_2, \dots, e_n\}, n \in \mathbb{N}^+$ : Menge von *Kanten*, wobei jede Kante ein Tupel

$e_g = (l_g, s_g, d_g, \alpha_g)$  mit einer *Position*  $l_g : (p_q, (\iota, V, K)) \in (P \times I)$ , einem

Vorzeichen  $s_g \in \{+, -\}$ , einer *Richtung*  $d_g \in \{\text{Vorbereich}, \text{Nachbereich}\}$  und

einem *Aktualisierungsparameter*  $\alpha_g \in V$  sei.

---

1. Ein Beispiel für die Auswertung der *sameTable*-Bedingung befindet sich in Kapitel 7, Seite 86.

$F_r = \{f_{r1}^{u1}, f_{r2}^{u2}, \dots, f_{rx}^{ux}\}$ : Menge der „relationalen“ Funktionen mit Stelligkeit

$uq \in IN$  wobei  $f_{ri}^{uq} : \mathfrak{R}^{uq} \rightarrow \mathfrak{R}$ . Dabei sei  $\mathfrak{R}^{uq}$  die Menge der Relationen mit Stelligkeit  $uq$ ,  $uq \in [u1, ux]$ .

$F_b = \{f_{b1}^{u1}, f_{b2}^{u2}, \dots, f_{bz}^{uz}\}$ : Menge der „boole’schen“ Funktionen mit Stelligkeit

$uy \in IN$ , wobei  $f_{bj}^{uy} : \mathfrak{R}^{uy} \rightarrow IB$ . Dabei sei wieder  $\mathfrak{R}^{uy}$  die Menge der Relationen mit Stelligkeit  $uy$  und  $IB$  die boole’sche Wertemenge.  $uy$  sei aus dem Intervall  $[u1, uz]$ .

$cf: I \rightarrow ]0, 1]$ : Konfidenz-Assoziation, assoziiert reelle Zahlen mit Implikationen.

$A^s = \{a^s_1, a^s_2, \dots, a^s_z\}$ : Menge der *starken Axiome*, jedes starke Axiom ist ein Tupel

$a^s_q = (p_q, \Omega_q)$ , mit  $p_q \in P$  und einer Funktion  $\Omega_q : \rightarrow \mathfrak{R}(U^{nq}) \times [0, 1] \times [0, 1]$ ,  $nq \in IN$ . Hierbei bezeichnet  $\mathfrak{R}(U^{nq})$  die Potenzmenge von  $nq$ -Tupeln der Elementemenge  $U$ .

$A^d = \{a^d_1, a^d_2, \dots, a^d_y\}$ : Menge der *aufgeschobenen Axiome*. Jedes aufgeschobene

Axiom ist ein Tupel  $a^d_q = (p_q, \omega_q)$  mit  $p_q \in P$  und einer Funktion  $\omega_q : \rightarrow \mathfrak{R}(U^{nq}) \times [0, 1] \times [0, 1]$ .

In diesem formalen Modell bezeichnet  $\Omega_q$  eine an starke Axiome gebundene Analyseoperation, die vor der eigentlichen Inferenz ausgeführt wird. Sie sucht in der relationalen Datenbank, also in der Schemadefinition, den Daten oder dem Applikationscode, nach bestimmten Eigenschaften dieser speziellen Datenbank. Analog repräsentiert  $\omega_q$  eine an aufgeschobene Axiome gebundene Analyseoperation, die erst bei Bedarf ausgeführt wird. Beide Funktionen liefern ein Tupel  $(c, n^+, n^-)$ . Hierbei bezeichnet  $n^+$  einen Wert aus dem Intervall  $[0, 1]$ , der die Notwendigkeit<sup>1</sup> einer Aussage  $p_q$  über der Konstanten-Menge  $c$  von  $nq$ -Tupeln über einem Universum  $U$  angibt. Analog dazu bezeichnet  $n^-$  die Notwendigkeit der negierten Aussage. Abbildung 6 zeigt eine kleine GFRN-Spezifikation, die auch in Abbildung 5 wiederzufinden ist. Hier ergibt sich

die Menge  $P$  zu  $P = \{cycl\_join, sel\_dist, key\}$ ,

$I$  zu  $I = \{(i_1, \{v\}, \emptyset), (i_2, \{v_1, v_2\}, \{v_2, \subseteq, (v_1)\})\}$ ,

$E$  zu  $E = \{((cycl\_join, (i_1, \{v\}, \emptyset)), +, VB, v), ((key, (i_1, \{v\}, \emptyset)), +, NB, v), ((sel\_dist, (i_2, \{v_1, v_2\}, \{v_2, \subseteq, (v_1)\})), +, VB, v_1), ((key, (i_2, \{v_1, v_2\}, \{v_2, \subseteq, (v_1)\})), -, NB, v_2)\}$ , und

---

1. vgl. Kapitel 2.6: Die Notwendigkeit einer Aussage gibt eine untere Schranke für die Sicherheit dieser Aussage an.

$F_r$  zu  $F_r = \emptyset$  und  $F_b$  zu  $F_b = \{ \subseteq^2 \}$ . Hierbei stehen *VB* und *NB* abkürzend für *Vor-* bzw. *Nachbereich*.

Die Konfidenzen werden über die Funktion  $cf$  den Implikationen folgendermaßen zugeordnet:  $cf(i_1) = 0.7$  und  $cf(i_2) = 0.3$ .

Die Menge der starken Axiome  $A^s$  hat das folgende Aussehen:  $A^s = \{cycl\_join, sel\_dist\}$ , und die Menge der aufgeschobenen Axiome ist leer.

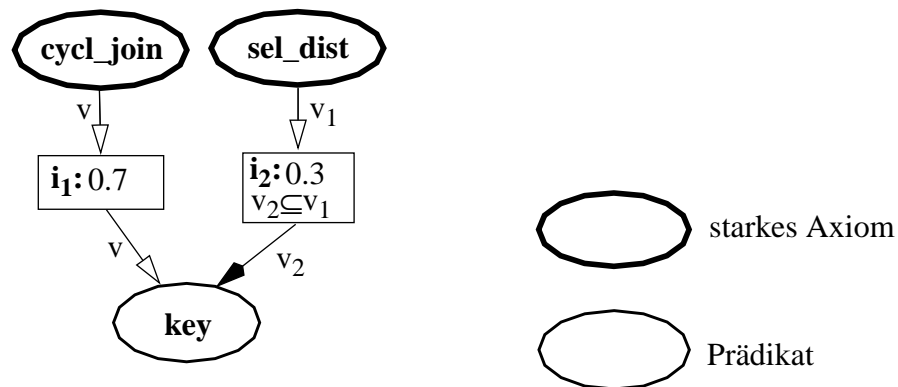


Abb. 6: Beispiel einer GFRN-Spezifikation

Der nachstehend aufgeführte Algorithmus beschreibt die Semantik der Generic Fuzzy Reasoning Nets. Er übersetzt eine gegebene GFRN-Spezifikation in einen äquivalenten Ausdruck der Prädikatenlogik erster Ordnung. Dazu wird jede Implikation des gegebenen GFRNs durchgegangen und in eine Formel dieses Logikkalküls überführt. Alle formalen Parameter einer Implikation werden hierbei mit dem All-Quantor versehen. Das Ergebnis des Algorithmus ergibt sich dann als Konjunktion aller Teilformeln für die einzelnen Implikationen.

### Definition 2: Semantik der Generic Fuzzy Reasoning Nets

**algorithm**  $GFRNToFormular (G):L^1$

- 1) **input:**  $G := (P, I, E, F_r, F_b, cf, A^s, A^d)$
- 2) **output:**  $F$ : eine geschlossene Formel in  $L^1$  mit gewichteten Implikationen.
- 3)
- 4) begin
- 5) **let**  $F = \tau$  // Tautologie
- 6) **for each**  $i:(\iota, V, K) \in I$  **do**
- 7)     **let**  $F = ' F \wedge (T'(i, V, K, G, \mathbf{False})) '$  **fi**
- 8) **od**
- 9) **return**  $F$
- 10) **end**

```

algorithm  $T'(i, V, K, G, CC):L^1$ 
11) begin
12) let  $F = F_d = \tau$  // Tautologie

13) if  $V = \emptyset$  then
14)   for each  $((p_q, i), s, \text{antecedent}, \alpha) \in E$  do
15)     let  $F = 'F \wedge s p_q(\alpha)'$ 
16)   od
17)   if  $CC = \text{False}$  then
18)     let  $((p_q, i), s, \text{consequent}, \alpha) \in E$ 
19)     let  $F = 'F \Rightarrow_{cf(i)} (s p_q(\alpha))'$ 
20)   fi
21) else

22)   let  $V_d = \{v' \mid v' \in V \wedge \neg \exists (v', r, f^u, \langle w_1, \dots, w_u \rangle) \in K (\exists z \in [1..u] (w_z \in V))\}$ 
      //  $V_d$  seien die Variablen in  $V$ , die nicht durch andere Variablen in  $V$  eingeschränkt
      // sind.

23)   let  $V_c = \{v' \mid v' \in V \wedge (\exists (l, s, \text{consequent}, v') \in E)\}$ 
      //  $V_c$  seien die Variablen in  $V$ , die in Prädikaten im Nachbereich von Implikation
      //  $i$  enthalten sind.

24)   if  $V_d \cap V_c \neq \emptyset$  then let  $v \in V_d \cap V_c$ 
25)     else let  $v \in V_d$  fi
      // wähle die nächste Variable aus  $V$ , die durch einen All-Quantor gebunden wird.

26)   let  $V = V \setminus v$ 

27)   for each  $k: (w, r, f^u, \langle w_1, \dots, w_u \rangle) \in K$  with  $\{w, w_1, \dots, w_u\} \cap V = \emptyset$  do
      // für jedes Constraint in  $K$ , das nur gebundene Variablen benutzt ...
28)     let  $K = K \setminus k$ 
29)     let  $F_d = 'F_d \wedge w r f^u(w_1, \dots, w_u)'$ 
30)   od

31)   for each  $k: (f^u, \langle w_1, \dots, w_u \rangle) \in K$  with  $\{w_1, \dots, w_u\} \cap V = \emptyset$  do
      // für jedes Constraint in  $K$ , das nur gebundene Variablen benutzt ...
32)     // für jedes Constraint in  $K$ , das nur gebundene Variablen benutzt ...
33)     let  $K = K \setminus k$ 
34)     let  $F_d = 'F_d \wedge f^u(w_1, \dots, w_u)'$ 
35)   od

36)   if  $CC = \text{True}$  or  $V_c \neq \emptyset$  then
37)     let  $F = '( \forall v )( F_d \Rightarrow_1 ( T'(i, V, K, G, CC) ) )'$ 
38)   else
39)     let  $((p_q, i), s, \text{consequent}, \alpha) \in E$ 
40)     let  $F = '( ( \forall v )( F_d \Rightarrow ( T'(i, V, K, G, \text{True}) ) ) ) \Rightarrow_{cf(i)} s p_q(\alpha)'$ 
41)   fi
42) fi
43) return  $F$ 
44) end

```

Der Algorithmus aus Definition 2 durchläuft nacheinander alle Implikationen der GFRN-Spezifikation  $G$ , die der Methode „ $GFRNToFormular(G):L^1$ “ als aktueller Parameter zu übergeben ist.  $G$  sei wie in Definition 1 definiert. Dazu initialisiert die Methode in Zeile (5) zunächst die Ergebnisformel des Algorithmus als Tautologie, d.h. als immer wahre Aussage. Diese Formel wird in Zeile (7) jeweils konjunktiv um die logischen Formeln verlängert, die sich aus den einzelnen Implikationen der GFRN-Spezifikation durch den Aufruf der rekursiven Funktion  $T'(i, V, K, G, CC)$  ableiten lassen. Der formale Parameter  $i$  repräsentiert die aktuell betrachtete Implikation von  $G$  und  $V$  und  $K$  die Variablen und Constraints in Implikation  $i$ . Der Parameter  $CC$  steht abkürzend für „*Consequent Complete*“ und drückt mit einem Wert von **False** aus, daß der Nachbereich der Implikation  $i$  in der Formel durch den Algorithmus noch nicht berücksichtigt worden ist.

In der Zeile (12) initialisiert die rekursive Funktion  $T'(i, V, K, G, CC)$  in jedem ihrer Aufrufe die lokale Variable  $F_d$  und den Rückgabeparameter  $F$  der Funktion als Tautologie.

Die Zeile (13) stellt die Abbruchbedingung für die Rekursion dar. Hier wird für die aktuelle Implikation geprüft, ob alle Variablen, über die Bedingungen in der Implikation  $i$  notiert sind, bereits gebunden sind. Ist dies der Fall, ergibt sich die Rückgabeformel aus den Formeln für den Vor- und Nachbereich von  $i$ . Zunächst setzt der Algorithmus dazu die Vorbedingung für die Formel  $F$  aus der Konjunktion aller Aussagen im Vorbereich der Implikation  $i$  zusammen. Dies geschieht in den Zeilen (14) bis (16) durch die Anweisung der Zeile (15) „ $F = 'F \wedge s p_q(\alpha)'$ “. Der Ausdruck  $p_q(\alpha)$  bezeichnet hierbei die Aussage, die das Prädikat  $p_q$  in der Kante  $((p_q, i), s, antecedent, \alpha)$  spezifiziert.  $\alpha$  ist der Aktualisierungsparameter, d.h. die Variable der Eingangskante  $((p_q, i), s, antecedent, \alpha)$ . Der Parameter  $s$  der Anweisung entspricht dem Vorzeichen dieser Kante. Ein positives Vorzeichen wird dabei in den leeren String und ein negatives in das *NOT*-Symbol  $\neg$  der Logik übersetzt, um die resultierende Formel an Konventionen der Formelschreibung anzupassen. Anschließend bestimmt der Algorithmus in Zeile (18) einen Term für den Nachbereich der Implikation, falls dieser noch nicht berücksichtigt worden ist, und hängt ihn als Konsequenz an die Formel  $F$  an (Zeile (19)). Dabei ergibt sich die Gewichtung der Formel direkt aus der Konfidenz  $cf(i)$  der Implikation  $i$ .

Die Abbruchbedingung des rekursiven Algorithmus  $T'$  in Zeile (13) ist nicht erfüllt, falls die Parametermenge  $V$  noch nicht leer ist. D.h., daß die Implikation  $i$  nach wie vor Variablen enthält, die bis jetzt nicht gebunden werden konnten. Ab Zeile (21) wird nun die nächste zu bindende Variable bestimmt. Diese Variable darf nur von bereits gebundenen Variablen abhängen. Deshalb berechnet  $T'$  in den Zeilen (22) und (23) zunächst zwei Mengen: die Menge  $V_d$  enthält alle Variablen aus  $V$ , die nicht durch andere Variablen aus  $V$  gebunden sind, die Menge  $V_c$  alle Variablen, die in Prädikaten im Nachbereich von  $i$  auftreten.  $V_c$  soll sicherstellen, daß zuerst die Variablen im Nachbereich der Implikation  $i$  gebunden werden. Aus diesen Mengen wählt der Algorithmus in Zeile (24) und (25) eine Variable  $v$  bevorzugt aus der Menge  $V_c$  aus, die in weiteren Schritten allquantifiziert wird. Ist die Schnittmenge der beiden Mengen nicht leer, existieren in Prädikaten im Nachbereich von  $i$  also freie Variablen, wählt  $T'$  die Variable  $v$  aus dieser Schnittmenge, ansonsten aus  $V_d$  und entfernt sie anschließend aus der Menge  $V$ .

Nun werden alle Constraints aus  $K$ , die nur gebundene Variablen enthalten, betrachtet und durch Konjunktion dieser Bedingungen in die Formel  $F_d$  in Zeile (29) aufgenommen. Dazu berücksichtigt der Algorithmus nacheinander die Constraints, die relationale Funktionssymbole

und die, die boole'sche Funktionssymbole enthalten (Zeile (27)).

Durch erneute rekursive Aufrufe der Funktion  $T'(i, V, K, G, CC)$  erweitert sich die Ergebnisformel  $F$  schrittweise mit Hilfe der Teilformel  $F_d$ . Wurde der Nachbereich bereits berücksichtigt oder befinden sich in der Menge  $V_c$  noch ungebundene Variablen, wird versucht, diese in weiteren rekursiven Schritten zu binden, und  $F$  setzt sich analog Zeile (37) zusammen.  $F$  ergibt sich damit aus einer über  $v$  allquantifizierten Formel mit  $F_d$  als Vorbedingung und einer rekursiv zu berechnenden Konsequenz. Ist die Menge  $V_c$  allerdings leer oder wurde der Nachbereich von  $i$  noch nicht berücksichtigt, setzt sich  $F$  aus einer allquantifizierten Formel über  $F_d$  mit einer noch zu berechnenden Konsequenz als Vorbedingung und der Formel über dem Prädikat im Nachbereich von  $i$  als Konsequenz entsprechend Zeile (40) zusammen.

### 3.4.1 Anwendung des Übersetzungsalgorithmus auf das Eingangsbeispiel

Abbildung 7 zeigt noch einmal einen verkürzten Ausschnitt aus der GFRN-Spezifikation für das Eingangsbeispiel. An diesem soll die Übersetzung eines GFRNs in eine  $L^I$ -Formel demonstriert werden.

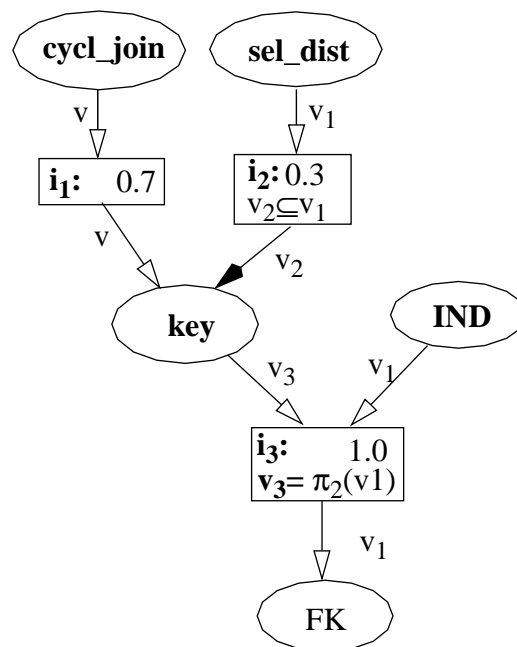


Abb. 7: Ausschnitt einer GFRN-Spezifikation für das Eingangsbeispiel

Der Algorithmus initialisiert die Ergebnisformel  $F$  zunächst mit der Tautologie  $\tau$  und besucht dann nacheinander alle Implikationen – im Beispiel in der Reihenfolge ihrer Numerierung.

Für Implikation  $i_j$  wird in Zeile (7) der rekursive Algorithmus  $T'$  mit den aktuellen Parametern  $(1, \{v\}, \emptyset, G, \mathbf{False})$  aufgerufen. Die Mengen  $V_d$  und  $V_c$  enthalten nach den Berechnungen in Zeile (22) und (23) jeweils die Variable  $v$ . Diese ist deshalb in Zeile (25) zum Binden auszuwählen und aus  $V$  zu entfernen. Der nun folgende Aufruf von  $T'$  in Zeile (37) findet mit  $(1, \emptyset, \emptyset, G, \mathbf{False})$  als aktuelle Parameter statt. Hier greift die Abbruchbedingung des Algorithmus in

Zeile (13).

Die Rückgabeformel  $F$  setzt sich dann in Zeile (15) zu

$$F = \tau \wedge cycl\_join(v), \text{ in Zeile (19) zu:}$$

$$F = \tau \wedge cycl\_join(v) \Rightarrow_{0,7} key(v), \text{ dann in Zeile (37) zu}$$

$$F = ((\forall v)(\tau \Rightarrow_1 \tau \wedge cycl\_join(v) \Rightarrow_{0,7} key(v))) \text{ und in Zeile (7) zu}$$

$$F = \tau \wedge ((\forall v)(\tau \Rightarrow_1 \tau \wedge cycl\_join(v) \Rightarrow_{0,7} key(v))) \text{ zusammen.}$$

Als nächstes muß nun eine Verlängerung dieses Zwischenergebnisses  $F$  um die logische Formel, die sich aus Implikation  $i_2$  ergibt, vorgenommen werden. Dazu ruft sich der rekursive Algorithmus  $T'$  mit den Parametern  $(2, \{v_1, v_2\}, \{v_2 \subseteq v_1\}, G, \mathbf{False})$  auf. Da die Menge  $V$  nicht leer ist, springt der Algorithmus in die Zeile (21). Ab hier wählt er zunächst wieder eine Variable zum Binden aus.  $V_d$  ergibt sich zu  $v_1$ ,  $V_c$  zu  $v_2$ . Da die Schnittmenge aus  $V_d$  und  $V_c$  leer ist, fällt die Wahl somit auf  $v_1$ . Ein Constraint ist in diesem Schritt noch nicht in die Formel integrierbar, da keines die Bedingungen aus Zeile (27) bzw. Zeile (31) erfüllt. Die Menge  $V_c$  ist nicht leer. Deshalb ergibt sich die Formel  $F$  dieser Rekursion analog zu Zeile (37) als:

$$F = (\forall v_1)(\tau \Rightarrow_1 (T'(2, \{v_2\}, \{v_2 \subseteq v_1\}, G, \mathbf{False}))).$$

In dem erneuten Aufruf der Funktion  $T'$  ist die Menge  $V$  noch immer nicht leer. Als nächste Variable zum Binden sucht  $T'$   $v_2$  aus und entfernt sie aus  $V$ . Nun kann das Constraint  $v_2 \subseteq v_1$  mit in die Formel eingebaut werden, und  $F_d$  berechnet sich entsprechend Zeile (29) zu

$$F_d = \tau \wedge v_2 \subseteq v_1.$$

Auch jetzt ist die Menge  $V_c$  nicht leer. Die Formel dieser Rekursion resultiert deshalb entsprechend der Zeile (37) zu:

$$F = (\forall v_2)(\tau \wedge v_2 \subseteq v_1 \Rightarrow_1 (T'(2, \emptyset, \emptyset, G, \mathbf{False}))).$$

Bei diesem rekursiven Aufruf von  $T'$  greift nun wiederum die Abbruchbedingung, und  $F$  setzt sich in dieser Rekursion in Zeile (15) zu

$$F = \tau \wedge sel\_dist(v_1) \text{ und in Zeile (19) zu}$$

$$F = \tau \wedge sel\_dist(v_1) \Rightarrow_{0,3} \neg key(v_2) \text{ zusammen.}$$

Diese Formel wird an Zeile (37) zurückgeliefert und dort zu

$$F = (\forall v_2)(\tau \wedge v_2 \subseteq v_1 \Rightarrow_1 (\tau \wedge sel\_dist(v_1) \Rightarrow_{0,3} \neg key(v_2))) \text{ erweitert.}$$

Dieses Ergebnis ist wiederum an Zeile (37) weiterzureichen und einzusetzen. Hier berechnet sich  $F$  dann zu

$$F = (\forall v_1)(\tau \Rightarrow_1 ((\forall v_2)(\tau \wedge v_2 \subseteq v_1 \Rightarrow_1 (\tau \wedge sel\_dist(v_1) \Rightarrow_{0,3} \neg key(v_2))))))$$

und die Ergebnisformel für Implikation  $i_2$  ist damit komplett. Sie wird nun in Zeile (7) mit der Formel für Implikation  $i_1$  zu

$$F = \tau \wedge ((\forall v)(\tau \Rightarrow_1 \tau \wedge cycl\_join(v) \Rightarrow_{0,7} key(v)) \wedge (\forall v_1)(\tau \Rightarrow_1 ((\forall v_2)(\tau \wedge v_2 \subseteq v_1 \Rightarrow_1 (\tau \wedge sel\_dist(v_1) \Rightarrow_{0,3} \neg key(v_2))))))$$

verknüpft.

Auch der Aufbau der Formel für  $i_3$  startet mit dem Aufruf in Zeile (7), nämlich mit

$$F = F \wedge (T'(3, \{v_1, v_3\}, \{v_3 = \pi_2(v_1)\}, G, False)).$$

Die Berechnungen der Mengen  $V_d$  und  $V_c$  in den Zeilen (22) und (23) ergeben beide als Ergebnis die Menge  $\{v_1\}$ .  $v_1$  wird deshalb zum Binden ausgewählt und aus der Menge  $V$  entfernt. Da  $V_c$  nicht leer ist, resultiert die Formel  $F$  dieser Rekursionsstufe in Zeile (37) zu

$$F = (\forall v_1)(\tau \Rightarrow_1 (T'(3, \{v_3\}, \{v_3 = \pi_2(v_1)\}, G, False))).$$

In diesem erneuten Aufruf von  $T'$  ergibt sich die Menge  $V_d$  zu  $v_3$ ,  $V_c$  bleibt hingegen leer. Als nächstes bindet  $T'$  also die Variable  $v_3$  und entfernt sie aus  $V$ . Nun kann das Constraint  $v_3 = \pi_2(v_1)$  in Zeile (34) in die Formel eingebracht werden. Die Formel  $F$  berechnet sich dann in Zeile (40) zu

$$F = ((\forall v_3)(\tau \wedge v_3 = \pi_2(v_1) \Rightarrow (T'(3, \emptyset, \emptyset, G, True)))) \Rightarrow_{1,0} FK(v_1),$$

da die Menge  $V_c$  leer ist. In der folgenden Rekursionsstufe greift in Zeile (13) die Abbruchbedingung des rekursiven Algorithmus  $T'$ , und der Vorbereitung von Implikation  $i_3$  ist zu berücksichtigen. Die Formel  $F$  dieser Rekursion ergibt sich deshalb in Zeile (15) zu  $F = \tau \wedge IND(v_1) \wedge key(v_3)$ . Sie wird in die nächsthöhere Rekursionsebene gereicht und dort in Zeile (40) eingesetzt.  $F$  berechnet sich dann zu

$$((\forall v_3)(\tau \wedge v_3 = \pi_2(v_1) \Rightarrow (\tau \wedge IND(v_1) \wedge key(v_3)))) \Rightarrow_{1,0} FK(v_1).$$

Diese Formel ist in Zeile (37) einzusetzen. Die Ergebnisformel für Implikation  $i_3$  hat damit das

folgende Aussehen

$$F = (\forall v_1)(\tau \Rightarrow_1((\forall v_3)(\tau \wedge v_3 = \pi_2(v_1) \Rightarrow (\tau \wedge IND(v_1) \wedge key(v_3)))) \Rightarrow_{1,0} FK(v_1)))$$

und wird an das Teilergebnis für Implikation  $i_1$  und  $i_2$  angehängt:

$$F = \tau \wedge ((\forall v)(\tau \Rightarrow_1 \tau \wedge cycl\_join(v) \Rightarrow_{0,7} key(v))) \quad (i_1)$$

$$(\forall v_1)(\tau \Rightarrow_1((\forall v_2)(\tau \wedge v_2 \subseteq v_1 \Rightarrow_1(\tau \wedge sel\_dist(v_1) \Rightarrow_{0,3} \neg key(v_2)))))) \wedge \quad (i_2)$$

$$(\forall v_1)(\tau \Rightarrow_1((\forall v_3)(\tau \wedge v_3 = \pi_2(v_1) \Rightarrow (\tau \wedge IND(v_1) \wedge key(v_3)))) \Rightarrow_{1,0} FK(v_1))). \quad (i_3)$$

Diese Formel läßt sich nun noch durch Entfernen der Tautologien und einiger Implikationspfeile bereinigen. Die Formel in Prädikatenlogik erster Ordnung für das GFRN aus Abbildung 6 sieht dann folgendermaßen aus:

$$F = ((\forall v)(\tau \Rightarrow_1 \tau \wedge cycl\_join(v) \Rightarrow_{0,7} key(v))) \wedge \quad (\text{Implikation } i_1)$$

$$(\forall v_1)((\forall v_2)((v_2 \subseteq v_1)(sel\_dist(v_1) \Rightarrow_{0,3} \neg key(v_2)))) \wedge \quad (\text{Implikation } i_2)$$

$$(\forall v_1)((\forall v_3)((v_3 = \pi_2(v_1))(IND(v_1) \wedge key(v_3)))) \Rightarrow_{1,0} FK(v_1))). \quad (\text{Implikation } i_3)$$



## 4 Das formale Modell der Inferenzmaschine: Ein unscharfes Petrinetz

Dieses Kapitel beschreibt die Ausführungsmaschine für GFRNs. Sie basiert auf einer speziellen Klasse von unscharfen Petrinetzen, die sich wegen ihrer Struktur besonders für nichtmonotones Schließen in Expertensystemen eignen.

In [KM96] stellen die Autoren A. Konar und A. K. Mandal ein unscharfes Petrinetzmodell für logisches Schließen unter Unsicherheit vor. Es weist jedoch einige Eigenschaften auf, die zu Problemen bei der Interpretation der Ergebnisse führen, die sich bei der Ausführung eines solchen unscharfen Petrinetzes ergeben. Deshalb wurde es für die besondere Problemstellung der Inferenzmaschine angepaßt und erweitert.

Abschnitt 4.1 stellt zunächst den Aufbau und das Schaltverhalten einer einfachen Petrinetzklasse als Grundlage für unscharfe Petrinetze vor. Diese Petrinetzklasse heißt *Stellen/Transitions-Netz (S/T-Netz)* [Rei82]. Anschließend wird das unscharfe Petrinetzmodell von Konar und Mandal erläutert. Nach Konar und Mandals Ansatz modellierte Netze werden zur Unterscheidung der vorgestellten Petrinetzmodelle in dieser Arbeit *K/M-Netze* genannt. Ein Beispiel zeigt im Anschluß daran die eben erwähnte Problematik bei der Interpretation der Berechnungsergebnisse. Die Beschreibung der Erweiterungen dieses Modells – der sogenannten *Fuzzy Reasoning Nets (FRN)* – erfolgt in Abschnitt 4.3.

Kapitel 5 als Bindeglied zwischen den GFRNs aus Kapitel 3 und den FRNs erläutert den *Expansionsalgorithmus*, der eine gegebene GFRN-Spezifikation in ein ausführbares *Fuzzy Reasoning Net* übersetzt.

### 4.1 S/T-Netze

Unter einem *S/T-Netz* versteht man einen gerichteten Graphen mit zwei disjunkten Knotenmengen, den Stellen und den Transitionen. Die Stellen als passive Elemente des S/T-Netzes dienen der temporären Aufnahme und Weitergabe von nichtunterscheidbaren Marken. Die Bewegung dieser Marken durch das S/T-Netz wird von den Transitionen als aktive Knoten des Netzes bewirkt. Die Besonderheit, daß in einem S/T-Netz nur Stellen mit Transitionen bzw. Transitionen mit Stellen durch gerichtete Kanten verbunden sein dürfen, macht S/T-Netze zu *bipartiten* Graphen. Formal läßt sich ein S/T-Netz wie folgt definieren:

#### Definition 3: Netz, S/T-Netz

Ein *S/T-Netz* ist ein 6-Tupel  $N = (S, T, F, K, G, M_0)$ , wobei

- (1)  $S = \{s_1, s_2, \dots, s_n\}$  eine endliche Menge von *Stellen* und
- (2)  $T = \{t_1, t_2, \dots, t_m\}$  eine endliche Menge von *Transitionen* ist.
- (3)  $S$  und  $T$  sind disjunkte Mengen, d.h.  $(S \cap T) = \emptyset$ .
- (4) Die *Flußrelation*  $F \subseteq (S \times T) \cup (T \times S)$  definiert eine endliche Menge gerichteter Kanten zwischen Stellen und Transitionen.
- (5)  $N' = (S, T, F)$  wird auch *Netz* genannt.

- (6) Die *Kapazitätsfunktion*  $K: S \rightarrow \mathbb{N} \cup \{\infty\}$  definiert eine (unbeschränkte) Kapazität jeder einzelnen Stelle, und die
- (7) *Gewichtsfunktion*  $G: F \rightarrow \mathbb{N} - \{0\}$  bestimmt zu jedem Pfeil ein Gewicht.
- (8) Die *Anfangsmarkierung*  $M_0: S \rightarrow \mathbb{N} \cup \{\infty\}$  definiert für jede Stelle eine initiale Belegung mit nichtunterscheidbaren Marken, die die Kapazitätsfunktion respektiert, d.h. jede Stelle trägt maximal so viele Marken, wie es die Kapazitätsfunktion erlaubt. Es gilt also für alle Stellen  $s \in S$  :  $M(s) \leq K(s)$ .

Ein S/T-Netz ist somit in dieser Arbeit ein Netz, auf dem zusätzlich eine Anfangsmarkierung definiert ist. Im folgenden werden deshalb die Begriffe *S/T-Netz* und *Netz* synonym verwendet.

Graphische Darstellungen von S/T-Netzen bilden Stellen üblicherweise als Kreise, Transitionen als Striche oder Rechtecke und die Flußrelation als Pfeile ab.

Für die weitere Verwendung der S/T-Netze in dieser Diplomarbeit sei die Kapazität jeder Stelle mit unendlich und das Gewicht jeder Kante mit 1 angenommen. Die Kapazitäts- und Gewichtsfunktion werden deshalb in den noch folgenden Abbildungen und Erläuterungen weggelassen.

Abbildung 8 zeigt ein S/T-Netz mit den Stellen  $S = \{s_1, s_2, s_3, s_4, s_5\}$ , den Transitionen  $T = \{t_1, t_2, t_3, t_4\}$  und den Kanten  $F = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}\}$ .

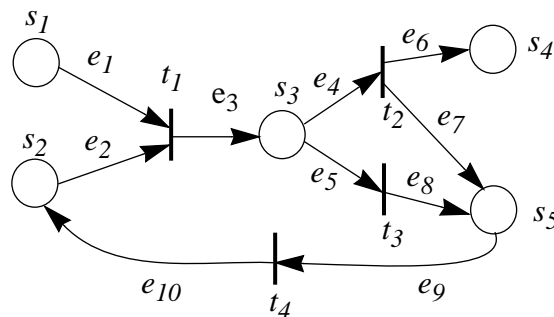


Abb. 8: Ein S/T-Netz

Zur Beschreibung des Schaltverhaltens eines S/T-Netzes, das den Fluß der Marken durch das Netz steuert, sind die Definitionen der *Markierung* des *Vor-* und *Nachbereichs*, der *Aktivierung* sowie der *Folgemarkierung* notwendig.

Die Markierung eines Netzes als Verallgemeinerung der in Definition 3 aufgeführten Anfangsmarkierung definiert eine Belegung der Stellen eines Petrinetzes mit nichtunterscheidbaren Marken.

#### Definition 4: Markierung

Sei  $N = (S, T; F)$  ein Netz.

Die Abbildung  $M: S \rightarrow \mathbb{N} \cup \{\infty\}$  heißt *Markierung* von  $N$ .

Der *Vorbereich* eines Knotens  $k \in \{S \cup T\}$  im Netz bezeichnet die Menge von Knoten, von denen aus gerichtete Kanten nach  $k$  einlaufen. Analog ist der *Nachbereich* eines Knotens  $k \in \{S \cup T\}$  die Menge von Knoten, in die von  $k$  ausgehende gerichtete Kanten eingehen.

### Definition 5: Vorbereich

Sei  $N = (S, T, F)$  ein Netz.

Für die Stelle  $s \in S$  ist der Vorbereich  $\circ s$  definiert als

$$\circ s := \{ts(s, t) \in F\}.$$

$\circ t := \{ss(t, s) \in F\}$  heißt Vorbereich einer Transition  $t \in T$ .

### Definition 6: Nachbereich

Sei  $N = (S, T, F)$  ein Netz.

Für die Stelle  $s \in S$  ist der Nachbereich  $s^\circ$  definiert als

$$s^\circ := \{ts(s, t) \in F\}.$$

$t^\circ := \{ss(t, s) \in F\}$  heißt Nachbereich einer Transition  $t \in T$ .

Die beiden nachfolgenden Definitionen *Aktivierung* und *Folgemarkierung* bestimmen das Schaltverhalten eines Netzes. Eine Transition ist aktiviert, wenn in allen Stellen ihres Vorbereichs eine Marke liegt. Die Transition erzeugt dann in allen Stellen ihres Nachbereichs eine Marke und löscht jeweils eine Marke in den Stellen ihres Vorbereichs. Man sagt auch, daß die Transition *feuert*. Die Markierung des Netzes, die nach dem Feuern einer Transition entsteht, wird *Folgemarkierung* genannt.

### Definition 7: M-aktiviert

Sei  $N = (S, T, F)$  ein Netz und  $M$  eine Markierung von  $N$ .

Eine Transition  $t \in T$  heißt *M-aktiviert* genau dann, wenn jede Stelle  $s \in S$  im Vorbereich von  $t$  eine Marke enthält, d.h.  $\forall (s \in \circ t): M(s) \geq 1$ .

### Definition 8: Folgemarkierung

Sei  $N = (S, T, F)$  ein Netz und  $t \in T$  eine *M-aktivierte* Transition. Die *Folgemarkierung*  $M'$  von  $M$  ist bestimmt durch<sup>1</sup>:

$$M'(s) = \begin{cases} M(s) - 1, & \text{falls } (s \in \circ t, s \notin t^\circ) \\ M(s) + 1, & \text{falls } (s \in t^\circ, s \notin \circ t) \\ M(s), & \text{sonst} \end{cases} \quad (1)$$

Die folgende Abbildung zeigt ein Beispiel für das Schaltverhalten eines S/T-Netzes. Die Marken in diesem Netz sind durch schwarze Punkte in den Stellen des Netzes dargestellt. Dabei re-

1. Das Gewicht jeder Kante wird mit 1 angenommen.

präsentiert Abbildung 9 (a) das Netz vor dem Schalten der Transition  $t_1$  und Abbildung 9 (b) nach Schalten von  $t_1$ .



Abb. 9: Schaltverhalten eines S/T-Netzes

#### 4.2 Das unscharfe Petrinetzmodell von Konar und Mandal

In [KM96] stellen die Autoren Konar und Mandal ein unscharfes Petrinetzmodell zum logischen Schließen in Expertensystemen vor. Die Wissensbasis eines Expertensystems, das sich mit realen Problemen befaßt, weist meistens eine Datenmenge auf, die unvollständig oder unpräzise ist. Außerdem kann die Wahrheit der Ableitungsregeln oft nicht als vollkommen sicher angesehen werden. Aus diesem Grund sind Unsicherheiten und auch Inkonsistenzen im initialen oder abgeleiteten Wissen nicht immer vermeidbar. Trotz allem ist eine Ableitung von Wissen zur Lösung praktischer Probleme mit akzeptablen Ergebnissen mit diesen Expertensystemen möglich.

In der Veröffentlichung von Konar und Mandal steht daher der Aspekt des Managements unsicheren Wissens in Expertensystemen und insbesondere die Behandlung von Inkonsistenzen im Vordergrund. Ein unscharfes Petrinetz stellt sich für diese Art des Schlußfolgerns als besonders geeignet heraus.

Die Ableitungsregeln, die das in einem Expertensystem vorhandene Wissen repräsentieren, können in einfacher und intuitiver Weise mit einem solchen Netz modelliert werden. Dabei repräsentieren die Stellen im Vor- und Nachbereich einer Transition die Vorbedingungen bzw. Konsequenzen einer solchen Ableitungsregel, und die Transition, die diese Stellen mit Hilfe gerichteter Kanten verknüpft, die Implikation. Eine Markierungsfunktion ordnet jeder Stelle den Wahrheitswert<sup>1</sup> der Aussage zu, die mit einer Stelle assoziiert ist. Die Transitionen enthalten darüber hinaus einen Gewichtungsfaktor, eine sogenannte *Konfidenz*. Diese gibt den Grad an, zu dem eine Ableitungsregel als sicher gilt, und ermöglicht die Berücksichtigung von Unsicherheiten im modellierten Wissen. Über Schwellwerte an den Transitionen kann eine untere Schranke für den Wahrheitsgrad eingestellt werden, den jede in die Transition einfließende Aussage erfüllen muß, bevor die Transition schaltet und somit neues Wissen abgeleitet wird. Einen weiteren Vorteil von unscharfen Petrinetzen als Grundlage einer Inferenz-Maschine in Expertensystemen bietet der hohe Grad an Parallelität bei der Ausführung eines solchen Netzes,

1. Konar und Mandal verwenden in ihrem Ansatz die Fuzzy-Logik zum logischen Schließen. Der Wahrheitswert einer Aussage gibt den Zugehörigkeitsgrad dieser Aussage zu der Menge der wahren Aussagen an (vgl. Kapitel 2.5).

der von der Besonderheit des Markenflusses beim Schalten einer Transition herrührt. Da hierbei keine Marken aus dem Vorbereich einer Transition abgezogen werden, können alle Transitionen gleichzeitig schalten.

Konar und Mandal verwenden zur Verrechnung der logischen Regeln in einem solchen unscharfen Petrinetz die Fuzzy-Logik wegen ihrer Eigenschaft, menschliche Schlußfolgerungsmechanismen nachempfinden zu können. Diese stellt sich außerdem in ihren Berechnungen im Vergleich zu beispielsweise probabilistischen Verfahren als besonders einfach dar. Die Marken in den Stellen repräsentieren den Wahrheitswert einer Aussage. Dieser gibt den Zugehörigkeitsgrad einer Aussage zu der Menge der *wahren Aussagen*<sup>1</sup> an. Die Konfidenzen und Schwellwerte an den Transitionen entsprechen reellen Zahlen aus dem Intervall  $[0,1]$ , und die zur Inferenz notwendigen Berechnungen lassen sich auf einfache Fuzzy-AND- und -OR-, d.h. Minimum- und Maximum-Operationen, reduzieren.

Konar und Mandal definieren ein unscharfes Petrinetz folgendermaßen:

**Definition 9: FPN (K/M-Netz)**

Ein FPN ist ein 9-Tupel  $FPN = \{P, Tr, D, I, O, cf, th, n, b\}$ . Hierbei sind

$P = \{p_1, p_2, \dots, p_n\}$  eine endliche Menge von *Stellen*,  
 $Tr = \{tr_1, tr_2, \dots, tr_m\}$  eine endliche Menge von *Transitionen*,  
 $D = \{d_1, d_2, \dots, d_n\}$  eine endliche Menge von *Aussagen* und es gilt  
 $P \cap Tr \cap D = \emptyset, |P| = |D|$ .

Weiterhin sind

$I: Tr \rightarrow P^\infty$ : eine *Eingangs-Funktion*, die die Transitionen mit ihren Eingangsstellen verknüpft und

$O: Tr \rightarrow P^\infty$ : eine *Ausgangs-Funktion*, die die Transitionen mit ihren Ausgangsstellen verbindet.

Die Funktionen

$cf, th: Tr \rightarrow [0, 1]$  ordnen jeder Transition

und die Funktion

$n: P \rightarrow [0, 1]$  jeder Stelle eine reelle Zahl aus dem Intervall  $[0, 1]$  zu.

Die Funktion

$b: P \rightarrow D$  assoziiert jede Stelle mit einer Aussage  $d \in D$ .

In der FPN-Terminologie heißt der Wert  $n$ , den die Funktion  $n$  jeder Stelle zuweist, *Fuzzy-Belief*. Dieser bezeichnet, wie bereits erwähnt, den Zugehörigkeitsgrad dieser Aussage zu der Menge der *wahren Aussagen*.

---

1. vgl. Kapitel 2.5.

Die *Konfidenz*  $cf$  einer Transition gibt an, zu welchem Grad die Implikation, die über diese Transition modelliert wurde, sicher ist.

Unterschiede zwischen K/M-Netzen und den in Abschnitt 4.1 beschriebenen S/T-Netzen sind unter anderem in der Markierung der Netze zu finden. Während in S/T-Netzen die Marken nichtunterscheidbar sind, bestehen die Marken eines K/M-Netzes aus reellen Zahlen aus dem Intervall  $[0, 1]$ . Außerdem werden die Stellen eines K/M-Netzes mit logischen Aussagen und die Transitionen mit Konfidenzen versehen, was zur Modellierung von Schlußfolgerungsregeln für unsichere Logik notwendig ist. Unterschiede der K/M-Netze zu den S/T-Netzen ergeben sich auch im Schaltverhalten, das in den folgenden Abschnitten erläutert wird.

#### 4.2.1 Das Schaltverhalten der K/M-Netze

Zur Beschreibung des Schaltverhaltens eines K/M-Netzes ist die folgende Definition hilfreich.

##### Definition 10: Aktivierung

Eine Transition  $tr_j$  heißt *aktiviert*, falls  $AND\{n_i: p_i \in I(tr_j)\} > th_j$ , wobei  $n_i = n(p_i)$  und  $th_j = th(tr_j)$ . Die Funktion  $AND\{x\}$  bezeichnet die Fuzzy-AND-Operation der Fuzzy-Logik und wird durch die Bildung des Minimums über die Parameter aus der Menge  $x$  ausgerechnet. Eine aktivierte Transition schaltet und sendet ein *Fuzzy-Truth-Token* (FTT) über alle seine Ausgangskanten. Der Wert des FTT ist eine Funktion des Konfidenzfaktors  $cf(tr_j)$  und der Fuzzy-Beliefs der Eingangsstellen.

Eine Transition  $j$  in einem K/M-Netz ist also schaltfähig, wenn die Fuzzy-Beliefs  $n_i$  aller Stellen  $p_i$  im Vorbereitung der Transition größer als der Schwellwert  $th_j$  der Transition sind.

Bei der Ausführung eines K/M-Netzes werden an Transitionen und Stellen neue Fuzzy-Werte berechnet. Alle Stellen im Netz enthalten eine initiale Markierung. Das Schaltverhalten eines K/M-Netzes ist über die Berechnung von *Fuzzy-Truth-Token* an den Transitionen und von *Fuzzy-Beliefs* in den Stellen definiert.

##### 4.2.1.1 Die Berechnung der Fuzzy-Truth-Token

Abbildung 10 zeigt eine Transition mit  $v$  Eingangsstellen. In die Berechnung des sogenannten *Fuzzy-Truth-Token* (vgl. Definition 10) an einer solchen Transition  $t_q$  gehen die Fuzzy-Beliefs der Stellen  $p_i$  mit  $1 \leq i \leq v$  im Vorbereitung dieser Transition ein. Die Transition  $t_q$  kann schalten, wenn das Minimum der Fuzzy-Beliefs der Eingangsstellen größer als der Schwellwert der Transition ist. Das Fuzzy-Truth-Token, das in diesem Fall neu berechnet wird, ergibt sich entsprechend aus dem Minimum des Konfidenzfaktors  $cf$  der Transition und dem Minimum der Fuzzy-Beliefs der Eingangsstellen.

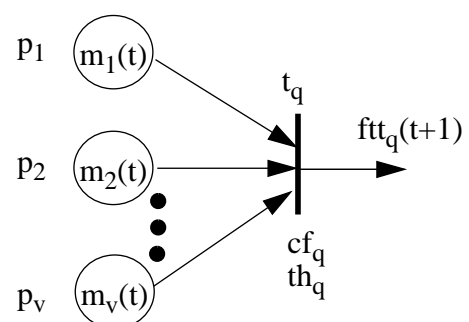


Abb. 10: Transition mit  $v$  Eingangsstellen

Diesen Sachverhalt stellt Gleichung (2) noch einmal dar.

$$ftt_{t+1}(tr_j) = \begin{cases} x \wedge cf(tr_j), & \text{falls } x > th(tr_j) \\ 0, & \text{sonst} \end{cases}, \quad (2)$$

wobei

$$x = \bigwedge_{1 \leq j \leq v} \{n_t(p_j)\}. \quad (3)$$

Die Operation  $\wedge$  bezeichnet wieder die Fuzzy-AND-Operation, die mit der Minimumfunktion implementiert ist.

#### 4.2.1.2 Die Berechnung der Fuzzy-Beliefs

Der Fuzzy-Belief einer Stelle  $p_i$  (vgl. Abbildung 11) des K/M-Netzes resultiert aus dem Maximum der Fuzzy-Truth-Token der Transitionen, in deren Nachbereich  $p_i$  sich befindet. Existiert keine solche Transition, in deren Nachbereich die Stelle  $p_i$  enthalten ist, wird der alte Fuzzy-Belief für  $p_i$  beibehalten. Dieser Zusammenhang ist formal in Gleichung (4) beschrieben.

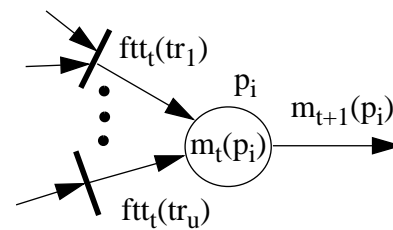


Abb. 11: Ausgangsstelle von  $u$  Transitionen

$$m_{t+1}(p_i) = \begin{cases} \bigvee_{1 \leq z \leq u} ftt_t(tr_z), & \text{falls } u > 0 \\ m_t(p_i), & \text{sonst.} \end{cases} \quad (4)$$

Hierbei bezeichnet  $\bigvee$  die Fuzzy-OR-Operation, die über die Maximumbildung berechnet wird.

Im Gegensatz zu herkömmlichen Petrinetzen werden die Marken der Eingangsstellen beim Schalten einer Transition nicht verbraucht, sondern lediglich kopiert. Dieser Mechanismus ist für die logische Inferenz in Expertensystemen sinnvoll, da zum Folgern einer Konsequenz  $B$  einer Implikation  $A \rightarrow B$  zunächst die Vorbedingung  $A$  erfüllt sein muß. Diese Vorbedingung gilt allerdings auch noch nach der Schlußfolgerung.

Die Berechnung der Fuzzy-Beliefs in einem unscharfen Petrinetz entsprechend der Gleichungen (2) und (4) kann parallel erfolgen und wird solange wiederholt, bis sich keine Änderung der Werte mehr ergibt. Diesen Zustand nennen Konar und Mandal *steady-state* oder *Gleichgewichtszustand*.

Konar und Mandal stellen in [KM96] einen Algorithmus zur Berechnung dieses Gleichgewichtszustandes vor. Um jedoch eine möglichst geschlossene Darstellung des in dieser Diplomarbeit erweiterten Ansatzes sicherstellen zu können, geht erst Kapitel 4.3.3 auf diesen Algorithmus ein.

### 4.2.2 Anwendung des Modells von Konar und Mandal

Die folgenden Abbildungen zeigen einen kleinen Ausschnitt aus einer GFRN-Spezifikation und das zugehörige K/M-Netz. An diesem sollen die Eigenschaften des Ansatzes erläutert werden, aus denen sich auch die Anregungen für die in Abschnitt 4.3 durchgeführten Erweiterungen ergeben. Dazu ist zuvor der Begriff des *K/M-Axioms* zu definieren.

#### Definition 11: K/M-Axiom

Eine Stelle, die über keine Eingangskante verfügt, heißt nach Konar und Mandal *Axiom* und wird in dieser Diplomarbeit zur Unterscheidung von den in Kapitel 3 definierten starken, schwachen und aufgeschobenen Axiomen als *K/M-Axiom* bezeichnet.

Das GFRN aus Abbildung 12 modelliert über die Implikation  $i_2$  mit einer Konfidenz von 0.8, daß ein im Applikationscode gefundenes *cyclic\_exclusion* Cliché<sup>1</sup> über einer Menge von Attributen auf die Schlüsseleigenschaft dieser Attribute hinweist.

Des weiteren enthält das GFRN ein Prädikat **join**, das das Interesse widerspiegelt, Verbunde über Attribute verschiedener Tabellen im Applikationscode einer relationalen Datenbankanwendung zu finden. Implikation  $i_5$  spezifiziert mit diesem Prädikat die Regel, daß die Existenz eines solchen Verbundes über eine Menge von Attributen im Code auf eine Teilmengenbeziehung zwischen diesen Attributen hinweist.

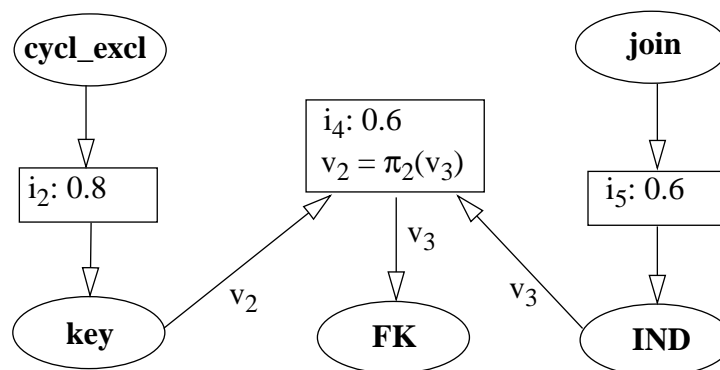


Abb. 12: Ausschnitt aus einer GFRN-Spezifikation

Die letzte Regel, die dieses GFRN enthält, sagt über Implikation  $i_4$  aus, daß eine Fremdschlüsselbeziehung aus einer Teilmengenbeziehung über eine Attributmeng  $v_3$  abgeleitet werden kann, wenn die Ergebnisrelation  $v_2$  der Projektion auf das zweite Element ( $\pi_2$ ) in der Attributmeng  $v_3$  nur Schlüsselattribute enthält.

1. Clichés sind vordefinierte Anfragemuster im Applikationscode einer RDBA (vgl. Kapitel 1.2)



für ein FPN detailliert. Die Gewichte, die an den Transitionen aus Abbildung 13 vermerkt sind, ergeben sich direkt aus den Konfidenzen der Implikationen im zugehörigen GFRN.

Das FPN drückt die Information aus, daß im Applikationscode ein *cyclic-exclusion* Cliché über das Attribut *name* gefunden wurde. Diese Tatsache weist mit einer Sicherheit von 0.8 auf die Schlüsseleigenschaft dieses Attributs hin. Die Stelle  $\neg\mathbf{key}(n)$  hält jedoch einen Wert von 0.3 gegen diese Aussage. Außerdem wurde ein *join* Cliché über die Attribute *name* und *developer* gefunden, das als Hinweis auf eine Teilmengenbeziehung zwischen diesen Attributen dient. Diese Teilmengenbeziehung gilt zum Grad 0.6 als wahr<sup>1</sup>, während die Stelle  $\neg\mathbf{IND}(d,n)$  aussagt, daß diese Beziehung mit einem Wahrheitsgrad von 0.3 nicht existiert. Da eine Schlüsseleigenschaft des Attributs *name* ( $\mathbf{key}(n)$ ) zum Grad 0.8 als wahr angesehen werden kann und eine Teilmengenbeziehung zwischen den Attributen *developer* und *name* ( $\mathbf{IND}(d,n)$ ) zu einem Grad von 0.6, kann das Zutreffen einer Fremdschlüsselbeziehung von *developer* auf *name* ( $\mathbf{FK}(d,n)$ ) zu einem Grad von 0.6 abgeleitet werden. Die Stelle  $\neg\mathbf{FK}(d,n)$  hat keinen Einfluß auf dieses Ergebnis, da keine Kante zwischen dieser Stelle und Transition  $tr_3$  existiert.

Ein Ziel, das Konar und Mandal mit ihrem Ansatz verfolgen, ist die Elimination von Inkonsistenzen, die durch das gleichzeitige Auftreten einer Aussage und ihrer Negation in demselben FPN herrühren. Dazu wird – vereinfacht beschrieben – nach einer ersten Analyse des FPNs für jedes inkonsistente Paar von Aussagen die Aussage mit dem niedrigeren Fuzzy-Belief auf 0 gesetzt, was Konar und Mandal auch *Erdung* nennen. Anschließend werden die Eingangskanten dieser Stellen eliminiert, um eine Beeinflussung weiterer Auswertungsschritte durch diese Stellen zu vermeiden. Mit den initialen Werten der Marken in den Stellen und den initialen Schwellwerten an den Transitionen wird die Analyse nun neu gestartet.

Wie Abbildung 13 zeigt, treten Inkonsistenzen jedoch sehr häufig auf, wenn zu jeder GFRN-Regel auch ihre Kontraposition modelliert wird. Gerade aber diese Kontraposition in Verbindung mit den aufgeschobenen Axiomen ermöglicht das nichtmonotone Schließen und ist somit unverzichtbar. Konar und Mandal verfolgen für das Management von Inkonsistenzen und Unsicherheiten jedoch eine andere Strategie als diese Arbeit, was auch einige unerwünschte und deshalb anpassungswürdige Effekte birgt.

Eine wichtige Eigenschaft ist, daß Inkonsistenzen in Konar und Mandals Ansatz bei der Inferenz nicht direkt verarbeitet werden können, d.h. es erfolgt keine Verrechnung der Fuzzy-Beliefs, die für und gegen eine Aussage sprechen. Aus diesem Grund ist die Interpretation von Analyseergebnissen nicht immer eindeutig. Abbildung 13 zeigt hierfür ein Beispiel. Das FPN enthält einen Hinweis auf das Zutreffen der Aussage „Es existiert eine Teilmengenbeziehung zwischen den Attributen *developer* und *name*“ mit einem Wahrheitsgrad von 0.6. Gegen diese Aussage spricht ein Wert von 0.2. Der intuitive *absolute* Fuzzy-Belief, der sich aus einer Verrechnung dieser beiden Werte ergibt, wäre damit 0.4. Trotzdem geht in die Herleitung einer Fremdschlüsselbeziehung zwischen *developer* und *name* nur der positive Hinweis für die Teilmengenbeziehung ein, während der negative vollständig vernachlässigt wird. Der gefolgerte Fuzzy-Wert von 0.6, der aussagt, daß eine Fremdschlüsselbeziehung zwischen *developer* und *name* mit einem Wahrheitsgrad von 0.6 existiert, erscheint somit intuitiv zu hoch zu sein.

---

1. Die Fuzzy-Beliefs einer Stelle geben den Zugehörigkeitsgrad einer Aussage zur Menge der wahren Aussagen an, vgl. Abschnitt 4.2.

Außerdem werden Widersprüche, die nach einer ersten Auswertung des FPNs bis zu einem Gleichgewichtszustand erkennbar sind, nicht wirklich behoben, sondern lediglich gelöscht und damit ignoriert. Dazu wird diejenige Aussage eines inkonsistenten Paares mit dem niedrigeren Fuzzy-Belief aus dem FPN entfernt und anschließend eine neue Auswertung gestartet. Dieses Verfahren läßt wiederum eine Verrechnung der inkonsistenten Fuzzy-Beliefs vermissen, was aus den bereits genannten Gründen zu Endergebnissen der Analyse führt, die intuitiv zu hoch sind.

### 4.3 Das unscharfe Petrinetzmodell der Inferenzmaschine - Fuzzy Reasoning Nets

Dieser Abschnitt beschreibt das unscharfe Petrinetzmodell der Inferenzmaschine, die Gegenstand dieser Diplomarbeit ist. Das Modell wurde aus dem von Konar und Mandal entwickelt und so verändert, daß zu jedem Zeitpunkt eine intuitive Interpretation der Analyseresultate durch eine direkte Verrechnung der Fuzzy-Beliefs einer Aussage und ihrer Negation möglich ist. Ein einfacher Vergleich dieser Werte dient der Ortung von Inkonsistenzen als Grundlage der geforderten Interaktion zwischen Analysewerkzeug und Anwender. Da dieses unscharfe Petrinetzmodell der Ausführung der in Kapitel 3 beschriebenen *Generic Fuzzy Reasoning Nets* dient, wird in Analogie hierzu das nun definierte unscharfe Petrinetzmodell *Fuzzy Reasoning Net (FRN)* genannt.

Den Anfang dieses Abschnitts bildet die Beschreibung des formalen Modells und seines Schaltverhaltens.

Die anschließende Stabilitätsanalyse widmet sich den Problemen, die im Zusammenhang mit möglichen Zyklen in unscharfen Petrinetzen das Erlangen aussagekräftiger Analyseergebnisse erschweren oder sogar verhindern.

Abschließend wird ein Verfahren zum Aufbrechen dieser problembehafteten Zyklen vorgestellt.

#### 4.3.1 Das formale Modell

Die wesentliche Änderung, die an Konar und Mandals Ansatz vorgenommen wurde, ist das Zusammenfassen einer Aussage und ihrer Negation in eine Stelle. Die Markierung jeder Stelle besteht damit aus zwei Fuzzy-Belief-Werten, die eine untere Schranke für die Sicherheit<sup>1</sup> der mit der Stelle assoziierten Aussage bzw. ihrer Negation angeben. Diese Maßnahme führt dazu, daß Inkonsistenzen im Wissen durch einfachen Vergleich der Fuzzy-Belief-Werte einer Stelle schnell erkennbar sind. Außerdem kann eine direkte Verrechnung beider Werte zu aussagefähigeren Analyseergebnissen führen. Um trotzdem weiterhin eine eindeutige Zuordnung einer einzelnen Aussage, d.h. entweder der eigentlichen Aussage oder ihrer Negation, zu einer logischen Implikation aus der GFRN-Spezifikation vornehmen zu können, müssen die Kanten des FRNs mit Vorzeichen versehen werden. Diese geben eindeutig an, in welche Regel ein positiver und in welche ein negativer Fuzzy-Belief eingeht. Diese Maßnahmen machen das Schaltverhalten und die Stabilitätsanalyse etwas komplizierter, da die Kantenvorzeichen zu berücksichtigen sind.

---

1. Diese untere Schranke wird auch *Notwendigkeit* genannt, vgl. Abschnitt 2.6.

Die nachfolgende Definition zeigt die formale Syntax der Fuzzy Reasoning Nets. Anschließend wird das Schaltverhalten dieses Modells beschrieben.

**Definition 12: Syntax der Fuzzy Reasoning Nets**

Ein Fuzzy Reasoning Net ist ein 9-Tupel

$$FRN = (P, T, F, D, m, b, cf, th, s).$$

Dabei sind  $P = (p_1, p_2, \dots, p_n)$  eine endliche Menge von *Stellen* und  $T = (t_1, t_2, \dots, t_m)$  eine endliche Menge von *Transitionen*. Diese beiden Mengen  $P$  und  $T$  definieren die Knoten des unscharfen Petrinetzes. Beide Mengen sollen paarweise elementfremd sein, d.h.

$$P \cap T = \emptyset.$$

Die *Flußrelation*  $F$  definiert gerichtete Kanten zwischen den Knoten des FRNs. Dabei dürfen Kanten allerdings nur zwischen Stellen und Transitionen bzw. zwischen Transitionen und Stellen verlaufen. Somit gilt:  $F \subseteq (P \times T) \cup (T \times P)$ .

$D$  definiert eine endliche Menge von *Aussagen* und die Funktion  $b: P \rightarrow D$  ordnet jeder Stelle eine solche Aussage zu.

Die *Markierungsfunktion*  $m$  assoziiert jede Stelle mit einem Paar  $(m^+, m^-)$ . Beide Werte sind aus dem Intervall  $[0,1]$  und werden *Fuzzy-Belief* genannt. Der positive Fuzzy-Belief  $m^+$  drückt aus, zu welchem Grad die einer Stelle  $p \in P$  zugeordnete Aussage  $d \in D$  notwendig ist, der negative Fuzzy-Belief  $m^-$  hingegen, zu welchem Grad die negierte Aussage notwendig ist. Des weiteren wird als *absoluter Fuzzy-Belief (AFB)* oder vereinfacht nur *Fuzzy-Belief* einer Stelle die absolute Differenz aus positivem und negativem Fuzzy-Belief bezeichnet. Die Markierung einer Stelle, d.h. ihr positiver und ihr negativer Fuzzy-Belief, heißen *Fuzzy-Belief-Marking (FBM)*.

Die Funktionen  $cf$  und  $th$  ordnen jeder Transition eine *Konfidenz* und einen *Schwellwert* aus dem Intervall  $[0,1]$  zu. Die Konfidenz  $cf$  beschreibt, zu welchem Grad die Implikation gilt, die durch die entsprechende Transition modelliert wird. Der an einer Transition notierte Schwellwert  $th$  dient zur Steuerung des Schaltverhaltens dieser Transition und hat initial den Wert 0.

Schließlich weist die Funktion  $s: F \rightarrow \{+, -\}$  jeder Kante ein Vorzeichen zu, das die Berechnung neuer Fuzzy-Werte bei der Ausführung eines unscharfen Petrinetzes steuert.

Unabhängig von dieser formalen Definition des unscharfen Petrinetzmodells wird für jede Stelle eines Netzes ein Inkonsistenzmaß definiert. Eine Inkonsistenz, die auch *Widersprüchlichkeit* genannt wird, liegt immer dann vor, wenn keiner der beiden Fuzzy-Beliefs einer Stelle den Wert 0 trägt. Formal ist dieses Widersprüchlichkeitsmaß folgendermaßen definiert:

**Definition 13: Inkonsistenz**

Die Funktion  $incons: P \rightarrow [0,1]$  ist definiert als  $incons(p_i) = m^+ \dot{\cup} m^-$ .

### 4.3.2 Das Schaltverhalten der FRNs

Bei der Ausführung eines FRNs werden in jedem Ausführungsschritt ähnlich zu den K/M-Netzen jeweils zwei neue Fuzzy-Werte berechnet. An den Transitionen des FRNs berechnen sich neue *Fuzzy-Truth-Token* (FTT), in den Stellen neue *Fuzzy-Beliefs*. Dies entspricht der Berechnung einer neuen Markierung des FRNs. In die Fuzzy-Beliefs gehen die Fuzzy-Truth-Token der Transitionen ein, in deren Nachbereich die entsprechende Stelle liegt.

#### 4.3.2.1 Die Berechnung der Fuzzy-Truth-Token

Die Fuzzy-Truth-Token an den Transitionen ergeben sich in Abhängigkeit von den Fuzzy-Beliefs der Stellen im Vorbereich der entsprechenden Transition und dem Kantenvorzeichen der jeweils in die Transition einlaufenden Kante. Die Berechnung des Fuzzy-Truth-Tokens an Transition  $t_q$  zum Zeitpunkt  $t+1$  zeigen Abbildung 14 und Gleichungen (5) bis (7). Als erstes werden die Fuzzy-Belief Werte der Stellen im Vorbereich von  $t_q$  in Abhängigkeit vom Vorzeichen der von der jeweiligen Stelle in die Transition einlaufenden Kante analog zu Gleichung (5) verrechnet:

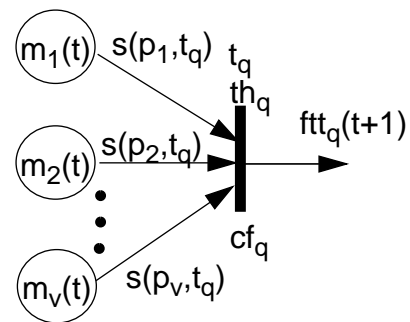


Abb. 14: Berechnung des FTT

$$AFB_{t+1}(p_z, t_q) = \begin{cases} m^+ - m^-, & \text{falls } m^+ > m^- \wedge s(p_z, t_q) = + \\ m^- - m^+, & \text{falls } m^- > m^+ \wedge s(p_z, t_q) = - \\ 0, & \text{sonst} \end{cases} \quad (5)$$

Ist das Vorzeichen der Kante positiv, ist die Differenz  $AFB_{t+1}(p_z, t_q)$  aus positivem und negativem Fuzzy-Belief zu bilden. Ist diese Differenz negativ, wird sie auf 0 gesetzt. Trägt die Kante hingegen ein negatives Vorzeichen, berechnet sich  $AFB_{t+1}(p_z, t_q)$  aus der Differenz des negativen und positiven Fuzzy-Beliefs. Ist dieser Wert negativ, wird wiederum auf 0 aufgerundet. Sind alle Fuzzy-Beliefs auf diese Art und Weise verrechnet, wird das Minimum  $\rho$  über diese Differenzen nach Gleichung (6) gebildet.

$$\rho = \bigwedge_{1 \leq z \leq v} \{AFB_{t+1}(p_z, t_q)\} \quad (6)$$

Ist  $\rho$  größer als der Schwellwert  $th_q$  der Transition  $t_q$ , schaltet diese, und das neue Fuzzy-Truth-Token  $ftt_{t+1}(t_q)$  zum Zeitpunkt  $t+1$  berechnet sich nach Gleichung (7) entsprechend aus dem

Minimum der minimalen Differenz der Fuzzy-Beliefs und der Konfidenz der Transition zu:

$$ftt_{t+1}(t_q) = \begin{cases} cf(t_q) \wedge \rho, & \text{falls } (\rho > th_q) \\ 0, & \text{sonst.} \end{cases} \quad (7)$$

Die Berechnung der neuen Fuzzy-Beliefs in den Stellen eines FRNs erfolgt durch Bestimmung einer neuen Markierung im Netz. Diese besteht für jede Stelle aus einem Markenpaar. Somit sind für jede Stelle zwei neue Fuzzy-Belief-Werte zu bestimmen, nämlich der positive und der negative.

#### 4.3.2.2 Die Berechnung der Fuzzy-Beliefs

Die Berechnung der Fuzzy-Beliefs findet nach der Bestimmung der Fuzzy-Truth-Token statt. Die Fuzzy-Beliefs, die zum Zeitpunkt  $t+1$  ermittelt werden, ergeben sich somit aus Fuzzy-Truth-Token aus dem Zeitpunkt  $t+1$ . In den positiven Fuzzy-Belief  $m^+_{t+1}(p_i)$  der Stelle  $p_i$  zum Zeitpunkt  $t+1$  gehen alle Fuzzy-Truth-Token der Transitionen ein, in deren Nachbereich  $p_i$  sich befindet, und die mit  $p_i$  über positive Kanten verbunden sind. Diese Fuzzy-Truth-Token werden mit der Fuzzy-OR-, also der Maximumfunktion verknüpft und ergeben so den neuen positiven Fuzzy-Belief. Existieren keine positiv markierten Kanten, die in die Stelle  $p_i$  einlaufen, wird der alte positive Fuzzy-Belief beibehalten.

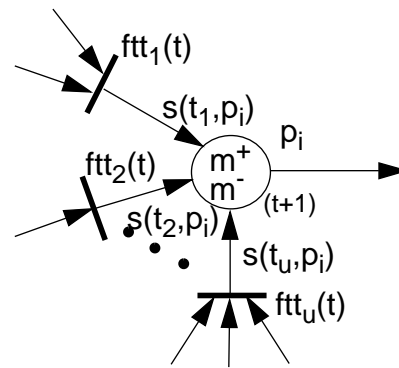


Abb. 15: Berechnung der Fuzzy-Beliefs

Analog stellt sich die Berechnung des negativen Fuzzy-Beliefs  $m^-_{t+1}(p_i)$  dar. In diesen Wert gehen alle Fuzzy-Truth-Token ein, die über negative Kanten in die Stelle  $p_i$  einlaufen. Auch diese Werte werden mit der Fuzzy-OR-Operation verknüpft und ergeben den neuen negativen Fuzzy-Belief. Existieren allerdings keine negativen Kanten, wird auch hier der ursprüngliche Wert für den negativen Fuzzy-Belief übernommen. Die Gleichungen (8) und (9) fassen diesen Sachverhalt noch einmal zusammen.

$$m^+_{t+1}(p_i) = \begin{cases} \bigvee_{1 \leq z \leq u \wedge s(t_z, p_i) = +} ftt_{t+1}(t_z) \\ m^+_t(p_i), & \text{sonst} \end{cases} \quad (8)$$

$$m^-_{t+1}(p_i) = \begin{cases} \bigvee_{1 \leq z \leq u \wedge s(t_z, p_i) = -} ftt_{t+1}(t_z) \\ m^-_t(p_i), & \text{sonst} \end{cases} \quad (9)$$

Vor der ersten Ausführung des FRNs erhält jede Stelle ein initiales Markenpaar. Die Werte dieser initialen Markierung ergeben sich entweder aus Berechnungen von Analyseoperationen oder werden auf Null gesetzt. Stellt eine Stelle im FRN eine Instanz eines starken oder aufgeschobenen Axioms dar, berechnen die an diese Axiome gebundenen Analyseoperationen initiale Markierungen für die zugehörige Stelle. Die Fuzzy-Beliefs für Instanzen schwacher Axiome bestimmt der Reengineer. Alle übrigen Stellen erhalten als initiale Fuzzy-Beliefs Null-Werte.

Das Abziehen von Marken aus Stellen im Vorbereitungsbereich einer schaltenden Transition, wie es beim Schalten von Transitionen in herkömmlichen Petrinetzen zu finden ist, ist bei unscharfen Petrinetzmodellen, die logische Implikationen der Form  $A \rightarrow B$  modellieren, nicht sinnvoll. Das Verbrauchen der Marken würde hier bedeuten, daß nach Folgern der Aussage  $B$  die Aussage  $A$  nicht mehr erfüllt wäre. Dieses entspricht jedoch nicht den Regeln der logischen Inferenz. Im Gegensatz zu herkömmlichen Petrinetzen werden beim Schalten einer Transition also keine Marken produziert bzw. verbraucht, sondern allenfalls kopiert und überschrieben. Somit ändert sich die Anzahl der initialen Marken nicht.

Diese besondere Eigenschaft, daß nämlich Position und Anzahl der Marken während der Ausführung eines FRNs fest bleiben, macht unscharfe Petrinetze für die Inferenzmaschine so besonders effizient, da jeweils alle Transitionen gleichzeitig aktivierbar, also schaltfähig sind. Das erklärt die hohe Parallelität bei der Berechnung der Fuzzy-Werte an Transitionen und Stellen.

### 4.3.3 Algorithmus und Stabilitätsanalyse

Die Berechnung der Fuzzy-Truth-Token und der Fuzzy-Beliefs wird entsprechend der Gleichungen (5) bis (9) nacheinander solange iteriert, bis sich keine Änderungen der Fuzzy-Beliefs in den Stellen des FRNs mehr ergeben. Das Hintereinanderausführen dieser beiden Berechnungsschritte, d.h. die gleichzeitige, einmalige Neuberechnung der Fuzzy-Truth-Token an allen Transitionen und anschließendes gleichzeitiges Ermitteln der neuen Fuzzy-Beliefs in allen Stellen definieren *einen Auswertungsschritt*.

Der Zustand, in dem sich durch Iteration weiterer Auswertungsschritte keine Änderungen der Fuzzy-Beliefs der Stellen mehr ergeben, heißt *steady-state* oder auch *Gleichgewichtszustand*. Entsprechend wird der Zeitpunkt  $\tilde{t}$ , zu dem der Gleichgewichtszustand zum ersten Mal nach Beginn der Auswertung angenommen wird, *Gleichgewichtszeit* genannt. Formal lassen sich diese Eigenschaften durch Definition 14 beschreiben:

#### Definition 14: Gleichgewichtszustand, Gleichgewichtszeit

Ein FRN befindet sich im *Gleichgewichtszustand* (*steady-state*), wenn für alle Stellen  $p_i$  in diesem Netz die Bedingung:

$$m_{\tilde{t}+1}(p_i) = m_{\tilde{t}}(p_i), 1 \leq i \leq n$$

gilt.

$$\left| \begin{array}{c} \min \\ t \end{array} \right| = \tilde{t}$$

heißt *Gleichgewichtszeit*.

Dieser Gleichgewichtszustand läßt sich leider nicht für alle FRNs erreichen. Probleme bereiten hierbei FRNs, die Zyklen aufweisen, in denen sich die Fuzzy-Belief-Werte der Stellen unendlich oft periodisch wiederholen und das Netz nie zu einem stabilen Zustand gelangen kann. Solche Zyklen werden *Begrenzungskreise* genannt. Für die logische Inferenz mit solchen zyklischen Netzen ist es deshalb erforderlich, die Existenz von Begrenzungskreisen zu erkennen und diese zu eliminieren.

Abschnitt 4.3.4 zeigt ein Verfahren zum Auflösen von Zyklen in einem FRN. Dabei wird eine Transition auf dem Zyklus ausgewählt und deren Schwellwert so weit heraufgesetzt, daß die Transition permanent nicht mehr schalten kann. Diese Veränderung des FRNs bedeutet auch eine ungenauere Darstellung des im zugehörigen GFRN spezifizierten Wissens und kann eine Verfälschung der Analyseergebnisse mit sich führen. Bei der Wahl der Transition ist deshalb darauf zu achten, daß diese auf einem Schlußfolgerungspfad im FRN liegt, der am wenigsten zum Endergebnis beiträgt und die Verfälschung der Analyseresultate möglichst minimal bleibt. Das Ermitteln von Kreisen in einem FRN kann durch eine statische Analyse desselben erfolgen. Durch eine Beobachtung der Werteverläufe in den Stellen, die auf einem Kreis liegen, kann dann eine periodische Wiederholung der Fuzzy-Beliefs erkannt werden. Der Rest dieses Abschnitts zeigt die Kriterien, die zur Erkennung von Begrenzungskreisen führen.

Der folgende Abschnitt listet einige Definitionen und Erkenntnisse auf, die zur Beschreibung des Algorithmus, der ein FRN in den Gleichgewichtszustand überführen soll, und der damit verbundenen Stabilitätsprobleme notwendig sind. Einige dieser Definitionen und Anmerkungen zur Stabilität in unscharfen Petrinetzen sind bereits in [KM96] nachzulesen. Diese mußten für die FRNs angepaßt und erweitert werden.

#### **Definition 15: Stellen-Vorbereich**

Sei  $N = (P, T, F, D, m, b, cf, th, s)$  ein Fuzzy-Reasoning-Net.

Der *Stellen-Vorbereich*  $\circ\circ p$  einer Stelle  $p \in P$  ist definiert als die Menge der Stellen, die sich im Vorbereich der Transitionen befinden, die wiederum zum Vorbereich der Stelle  $p$  gehören. Formal:

$$\circ\circ p = \{ \forall p' (: p' \in P) \wedge (p' \in \circ t) \wedge (t \in \circ p) \wedge (t \in T) \}$$

#### **Definition 16: Schlußfolgerungspfad**

Ein Pfad in einem FRN zwischen zwei Stellen, auf dem je zwei direkt aufeinanderfolgende Stellen durch gerichtete Kanten und eine dazwischenliegende Transition verbunden sind, heißt *Schlußfolgerungspfad*.

#### **Definition 17: dominante Kante**

Eine Kante heißt in einem Auswertungsschritt  $i$  *dominant*, wenn sie in Schritt  $i$

- (a) unter allen in eine Transition einlaufenden Kanten den kleinsten  $AFB^1$  entspre-

chend Gleichung (5) führt, bzw.

- (b) das größte Fuzzy-Truth-Token aller Transitionen besitzt, die über Kanten mit dem gleichen Vorzeichen in eine Stelle einlaufen. Für jede Stelle können somit zwei dominante Kanten existieren, die in sie einlaufen.

Abbildung 16 zeigt ein Beispiel für die beiden Fälle der Definition. Die Werte an den Kanten in Abbildung 16 (a) stellen die  $AFB^1$  dar. Die Kante mit dem niedrigsten Fuzzy-Belief, hier der Wert 10, ist dominant. In Abbildung 16 (b) ist die Kante dominant, die das größte Fuzzy-Truth-Token einer der Transitionen im Vorbereich der Stelle  $p_i$  in Abhängigkeit von den Kantenvorzeichen trägt. Hier sind das die positive Kante, die eine Transition mit einem FTT von 40 mit der Stelle  $p_i$ , und die negative Kante, die  $p_i$  und eine Transition mit einem FTT von 70 verbindet.

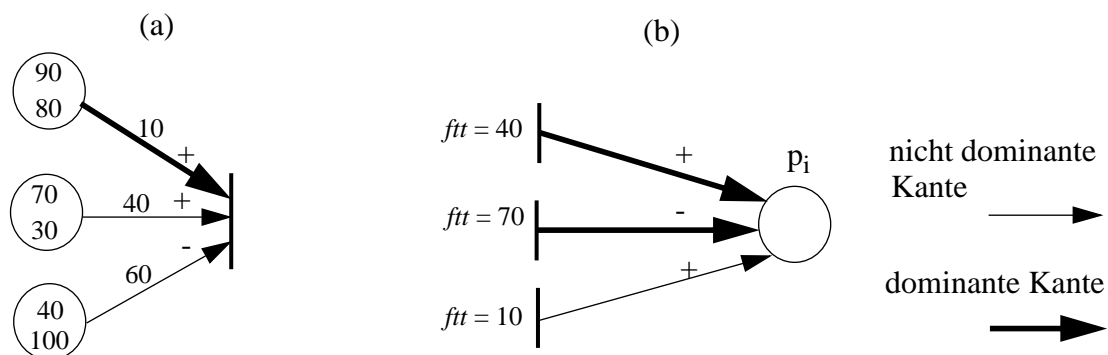


Abb. 16: Dominante und nicht dominante Kanten

### Definition 18: periodische Oszillation, Begrenzungskreis

Ein FRN weist *periodische Oszillationen (PO)* auf, wenn sich die Fuzzy-Belief-Werte in jeder Stelle auf irgendeinem Zyklus in diesem FRN nach einer endlichen Anzahl von Auswertungsschritten  $r_1$  von Auswertungsschritt  $r_1$  bis  $r_2$  periodisch wiederholen.

Bleibt diese periodische Oszillation auf dem Zyklus für eine unendliche Anzahl von Auswertungsschritten im FRN erhalten, d.h. strebt  $r_2$  gegen unendlich, heißt der Zyklus *Begrenzungskreis*.

### Definition 19: Kontrapositionsstelle (KS)

Eine Stelle auf einem Schlußfolgerungspfad heißt *Kontrapositionsstelle (KS)*, wenn sich die Vorzeichen der Kanten unterscheiden, die entlang des Schlußfolgerungspfades in diese Stelle ein- und von ihr ausgehen.

- 
1. vgl. Definition 12: die absolute Differenz aus positivem und negativem Fuzzy-Belief wird als AFB bezeichnet.
  1. vgl. Definition 12.

Abbildung 17 (b) zeigt eine solche Kontrapositionsstelle bezüglich eines (gestrichelt dargestellten) Pfades.

### Definition 20: Dominanzkreis (DK)

Ein Zyklus in einem FRN heißt *Dominanzkreis* (DK), wenn alle Kanten entlang des Kreises für eine Dauer von  $t$  Auswertungsschritten dominant bleiben.

Für die Formulierung der Abbruchbedingung des Auswertungsalgorithmus und die Überprüfung der Begrenzungskreis-Eigenschaft für einen Zyklus ist es notwendig, jede Stelle mit einem *Status* zu versehen.

### Definition 21: Status einer Stelle

Einer Stelle wird der Status *steady-state* ( $ss$ ) zugewiesen, falls sie ein K/M-Axiom repräsentiert, oder alle Stellen in ihrem Vorbereich den Status  $ss$  haben.

Eine Stelle erhält den Status *periodic oscillation* ( $po$ ), falls sie

- (a) auf einem Dominanzkreis mit  $k$  Transitionen liegt, der ab einem Zeitpunkt  $t$  für eine gerade Anzahl von Kontrapositionsstellen  $k$  Auswertungsschritte und für eine ungerade Anzahl von Kontrapositionsstellen  $2k$  Auswertungsschritte existiert<sup>1</sup> und
- (b) die Fuzzy-Belief-Werte aller Stellen auf diesem Kreis zum Zeitpunkt  $t$  gleich den Werten dieser Stellen zum Zeitpunkt  $t + per^2$  mit  $per = k$  bzw.  $per = 2k$ , d.h.  $FBM_t(p_i) = FBM_{t+per}(p_i)$ , und
- (c) die Werte von Zeitpunkt  $t + 1$  bis Zeitpunkt  $t + per - 1$  unterschiedlich sind.

Andernfalls erhält eine Stelle den Status *changing* ( $ch$ ).

Die nun folgenden Lemmata und Theoreme geben die notwendigen Kriterien zum Erkennen von Begrenzungskreisen an. Erst damit ist es möglich, den Auswertungsalgorithmus und insbesondere die Abbruchbedingung zu formulieren. Lemma 1 bezieht sich auf die Berechnung der Fuzzy-Truth-Token auf einem Schlußfolgerungspfad an Transitionen, die direkt auf eine KS bzw. eine Nicht-KS folgen. Lemma 2 vergleicht unter Zuhilfenahme des Ergebnisses aus Lemma 1 für einen gegebenen Schlußfolgerungspfad die Auswirkung von KS und Nicht-KS auf unterschiedliche Startwerte als Eingabe für den Pfad. Anschließend zeigt Theorem 1 die Länge der Periode auf Dominanzkreisen. Mit diesen Ergebnissen können schließlich in Theorem 2 die Kriterien aufgezeigt werden, die ein Dominanzkreis aufweisen muß, um als *Begrenzungskreis* zu gelten.

---

1. Die Periode  $per$ , mit der sich die Werte auf einem Dominanzkreis mit  $k$  Kontrapositionsstellen wiederholen, beträgt maximal  $2k$ . Diese Behauptung wird in dem Beweis zu Theorem 1 gezeigt.  
2.  $per$  steht für die Periodenlänge in einem DK.

**Lemma 1:**

Abbildung 17 (a) zeigt eine Nicht-Kontrapositionsstelle und Abbildung 17 (b) eine Kontrapositionsstelle. Beide Stellen liegen auf einem Pfad  $pf$ , dessen Verlauf gestrichelt dargestellt ist.  $\gamma$  ist hierbei ein Fuzzy-Wert, der über eine Kante in die Stelle eingeht, die nicht entlang des Pfades verläuft.  $s$  bzw.  $\neg s$  bezeichnen die Kantenvorzeichen.

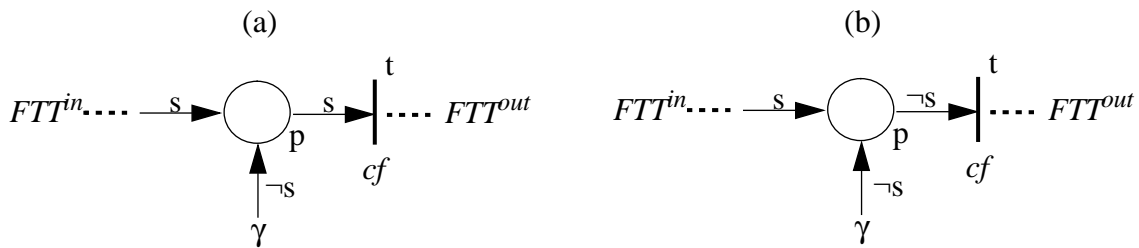


Abb. 17: (a) Eine Nicht-KS und (b) eine KS

Das Fuzzy-Truth-Token  $FTT^{out}$  nach einer Nicht-Kontrapositionsstelle  $p$  berechnet sich bei Eingabe  $FTT^{in}$  in  $p$  zu

$$FTT^{out} = \text{Min}((FTT^{in} - \gamma), cf)$$

und nach einer Kontrapositionsstelle zu

$$FTT^{out} = \text{Min}((\gamma - FTT^{in}), cf).$$

Der Beweis dieses Lemmas ist trivial und soll an dieser Stelle nicht gezeigt werden. Der Vollständigkeit halber wird er in Anhang A aufgeführt.

Lemma 1 sagt also aus, daß die Berechnung an KS und Nicht-KS unabhängig von der Konstellation der Kantenvorzeichen entlang des Schlußfolgerungspfades, auf dem diese Stellen liegen, durchgeführt werden kann. Für die Berechnung der FTT an einer Transition, die auf eine Nicht-KS folgt, ist es also unerheblich, ob die Vorzeichen der Kanten, die in die Nicht-KS entlang des Pfades ein- bzw. von ihr ausgehen, beide positiv oder beide negativ sind. Entsprechendes gilt für Transitionen, die auf eine Kontrapositionsstelle folgen. Hier ist es für die Berechnung ebenfalls ohne Bedeutung, ob die Eingangskante in die KS entlang des Pfades positiv und die Ausgangskante entlang des Pfades negativ oder umgekehrt ist. Lemma 2 geht nun darauf ein, welchen Einfluß die beiden unterschiedlichen Stellentypen entlang eines Schlußfolgerungspfades auf die Berechnungsergebnisse haben, die sich aus unterschiedlichen Eingabewerten für einen Pfad ergeben.

**Lemma 2:**

Gegeben sei ein Pfad mit  $k$  Kontrapositionsstellen (KS). Für eine Eingabe von  $FTT_I^{in}$  und

$FTT_2^{in}$  mit  $FTT_1^{in} = FTT_2^{in} + \alpha$ ,  $\alpha > 0$ , gilt:

$$FTT_1^{out} = FTT_2^{out} + \varepsilon; \varepsilon \in [0, \alpha], \text{ für } k \text{ gerade und}$$

$$FTT_1^{out} + \varepsilon = FTT_2^{out}; \varepsilon \in [0, \alpha], \text{ für } k \text{ ungerade.}$$

**Beweis:**

An dieser Stelle soll wegen der Länge des Beweises nur die Beweisidee anhand einiger Abbildungen vorgestellt werden. Der vollständige Beweis befindet sich in Anhang A.

Der Beweis gliedert sich in 3 Teile. Teil 1 und 2 untersuchen die Ergebnisse  $FTT_1^{out}$  und  $FTT_2^{out}$  der Eingaben  $FTT_1^{in}$  und  $FTT_2^{in}$  nach dem Passieren einer Nicht-Kontrapositionsstelle bzw. einer Kontrapositionsstelle auf einem einelementigen Pfad. Der dritte Teil weist die Behauptung des Lemmas mittels vollständiger Induktion über die Anzahl der KS auf einem Pfad nach. Zu Teil 1 und Teil 2 werden hier lediglich Abbildungen für die angesprochenen Fälle und die sich dafür ergebenden Berechnungsalternativen der Ausgabeergebnisse aufgeführt. Für Teil 3 sei vollständig auf Anhang A verwiesen.

**Teil 1: Betrachte eine Nicht-Kontrapositionsstelle:**

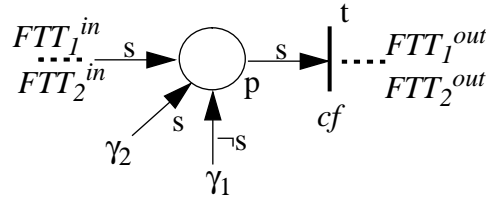


Abb. 18: Eine Nicht-KS

Teil 1 des Beweises betrachtet eine Nicht-KS. Für die Eingabe  $FTT_1^{in}$  und  $FTT_2^{in}$  in die Stelle  $p$  berechnen sich die Fuzzy-Truth-Token  $FTT_1^{out}$  und  $FTT_2^{out}$  an Transition  $t$  zu:

$$FTT_1^{out} = \begin{cases} \text{Min}(cf, FTT_1^{in} - \gamma_1), & \text{falls } FTT_1^{in} > \gamma_2 \\ \text{Min}(cf, \gamma_2 - \gamma_1), & \text{falls } FTT_1^{in} < \gamma_2 \end{cases}$$

$$FTT_2^{out} = \begin{cases} \text{Min}(cf, FTT_1^{in} - \alpha - \gamma_1), & \text{falls } FTT_2^{in} > \gamma_2 \\ \text{Min}(cf, \gamma_2 - \gamma_1), & \text{falls } FTT_2^{in} < \gamma_2 \end{cases}$$

Für den Beweis der Aussage werden grob die Fälle  $FTT_1^{in} > \gamma_2$  und  $FTT_1^{in} < \gamma_2$  und alle sich daraus ergebenden Unterfälle untersucht. Betrachtet man die Ergebnisse  $FTT_1^{out}$  und  $FTT_2^{out}$  im Vergleich zueinander, erhält man als Ergebnis für Teil 1 den

folgenden Sachverhalt:

$$\neg FTT_1^{in} = FTT_2^{in} + \alpha, \alpha > 0 \Rightarrow FTT_1^{in} = FTT_2^{out} + \varepsilon, \varepsilon > 0.$$

Ist  $FTT_1^{in}$  vor Passieren der Nicht-KS größer als  $FTT_2^{in}$ , so ist nach Passieren der Nicht-KS  $FTT_1^{out}$  ebenfalls größer als  $FTT_2^{out}$ .

**Teil 2: Man betrachte eine Kontrapositionsstelle:**

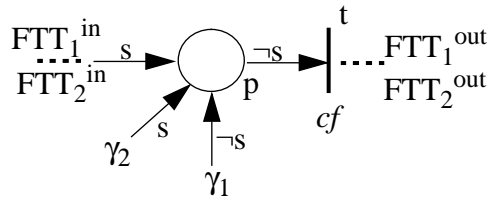


Abb. 19: Eine KS

Wiederum gelte:  $FTT_1^{in} = FTT_2^{in} + \alpha, \alpha \geq 0$ .

Für eine Kontrapositionsstelle berechnen sich  $FTT_1^{out}$  und  $FTT_2^{out}$  nach den folgenden Gleichungen:

$$FTT_1^{out} = \begin{cases} \text{Min}(cf, \gamma_1 - FTT_1^{in}), & \text{falls } FTT_1^{in} > \gamma_2 \\ \text{Min}(cf, \gamma_1 - \gamma_2), & \text{falls } FTT_1^{in} < \gamma_2 \end{cases}$$

$$FTT_2^{out} = \begin{cases} \text{Min}(cf, \gamma_1 - FTT_1^{in} + \alpha), & \text{falls } FTT_2^{in} > \gamma_2 \\ \text{Min}(cf, \gamma_1 - \gamma_2), & \text{falls } FTT_2^{in} < \gamma_2 \end{cases}$$

In Analogie zu Teil 1 dieses Beweises vergleicht Teil 2 ebenfalls die oben erwähnten Fälle für die Werte von  $FTT_1^{in}$  und  $\gamma_2$  miteinander. Für Kontrapositionstellen, die in Teil 2 des Beweises betrachtet werden, dreht sich das Ergebnis aus Teil 1 allerdings um. Hier gilt:

$$\neg FTT_1^{in} = FTT_2^{in} + \alpha, \alpha > 0 \Rightarrow FTT_1^{in} + \varepsilon = FTT_2^{out}, \varepsilon > 0.$$

**Teil 3: Vollständige Induktion**

Mit Hilfe dieser Zwischenergebnisse aus Teil 1 und Teil 2 dieses Beweises läßt sich die

Behauptung des Lemmas in Teil 3 mit Hilfe der vollständigen Induktion über die Anzahl der KS auf einem Pfad leicht zeigen.

Theorem 1 zeigt nun, daß die Periode auf einem Dominanzkreis mit  $k$  Transitionen höchstens  $2k$  beträgt.

**Theorem 1:**

Sei  $p_1, \dots, p_k$  von Zeitpunkt  $r_s$  bis  $r_{s+m}$  ein Dominanzkreis. Es gelte weiterhin für alle  $i \in [1, k]$ , daß  $FBM_{r_s}(p_i) = FBM_{r_s+per}(p_i)$ , wobei  $per$  die Periode des Dominanzkreises ist.

Dann oszilliert das  $FBM$  auf  $p_1, \dots, p_k$  mit Periode  $per = 2k$ , wenn der Kreis eine ungerade Anzahl von KS, bzw. mit Periode  $per = k$ , wenn der Kreis eine gerade Anzahl von KS enthält, und es gilt  $s + x \cdot per < m, x > 0$ .

**Beweis:**

Sei der Wert von  $s$  größer als die Länge des längsten Schlußfolgerungspfades.

**Teil 1:** Kreis mit  $z$  KS,  $z$  gerade.

Man wähle zum Zeitpunkt  $s < t < s + per$  das Fuzzy-Truth-Token, das maximal ist. O.B.d.A. sei dies das FTT, das die Transition direkt hinter Stelle  $p_n, n \in [1, k]$  erzeugt. Dieses FTT wird mit  $FTT_t(n)$  bezeichnet.

Da die Anzahl der KS nach Voraussetzung auf dem Kreis gerade ist und alle Kanten auf dem Kreis dominant sind, gilt nach weiteren  $k$  Auswertungsschritten (Länge des Kreises), daß  $FTT_{t+k}(n)$  nach Lemma 2 wieder maximal sein muß. Somit wiederholt sich das auf der Stelle  $p_n$  erzeugte FTT wegen der Dominanz der Kanten auf dem Kreis alle  $k$  Schritte, solange  $t + x \cdot k < s + m$ . Die Periode des Kreises ist also  $per = k$ .

**Teil 2:** Kreis mit  $z$  KS,  $z$  ungerade.

Man wähle wiederum zum Zeitpunkt  $t > s$  das Fuzzy-Truth-Token mit maximalem Wert. O.B.d.A. sei dies das FTT, das die Transition direkt hinter Stelle  $p_n, n \in [1, k]$  erzeugt. Dieser wird mit  $FTT_t(n)$  bezeichnet. Nach  $k$  Schritten ist nach Lemma 2 das  $FTT_{t+k}(n)$  wegen der ungeraden Anzahl an KS minimal. Damit ist das FTT zum Zeitpunkt  $t + 2 \cdot k$   $FTT_{t+2k}(n)$  ebenfalls nach Lemma 2 wieder maximal. Somit gilt  $FBM_{t+i \cdot 2k}(n) = FBM_{t+(i+1)2k}(n)$ , solange  $t + (i + 1)2k < s + m$ . Die Periode  $per$  des Kreises ergibt sich damit zu  $per = 2k$ .

**Theorem 2:**

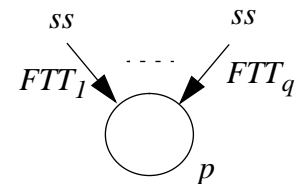
Sei  $p_1, \dots, p_k$  ein Dominanzkreis, und alle Stellen dieses Kreises haben Status  $po$ . Besitzen alle Stellen im Vorbereich von  $p_1, \dots, p_k$  den Status  $po$  oder  $ss$  und sind nach dem Dominantwerden aller Kanten auf dem DK mindestens

$BRS_{min} = \text{Max}(\text{kgV}(\text{per}_1, \dots, \text{per}_y))$  viele Schritte ausgewertet worden, so oszillieren die Fuzzy-Beliefs der Stellen  $p_1, \dots, p_k$  unendlich. Der Kreis  $p_1, \dots, p_k$  heißt dann *Begrenzungskreis*. (BRS steht für *Auswertungsschritt*, engl: Belief Revision Step)

### Beweis:

**Teil 1:** Alle Stellen im Stellen-Vorbereich der Stellen des DK haben Status  $ss$

Man betrachte Stelle  $p$ . Diese Stelle liege auf einem Kreis  $K_i$  mit Periode  $per$ . Alle Stellen im Vorbereich dieser Stelle (außer der Stelle im Vorbereich entlang des Kreises  $K_i$ ) haben den Status  $ss$ . Da alle Kanten auf dem Kreis  $K_i$  für eine ausreichende Anzahl von Schritten dominant sind, leitet jede Stelle auf  $K_i$  ihren Fuzzy-Belief zu ihrer direkt folgenden Stelle auf dem Kreis weiter. Nach  $per^1$  vielen Schritten erhält jede Stelle zum Zeitpunkt  $t$  ihren zum Zeitpunkt  $t - per$  erzeugten Fuzzy-Belief zurück. Die Fuzzy-Truth-Token  $FTT_1$  bis  $FTT_q$ , die an den direkt auf Stellen im Stellenvorbereich der Stellen  $p$  auf dem Kreis  $K_i$  folgenden Transitionen erzeugt werden, bleiben für die weiteren Ausführungsschritte konstant. Deshalb haben diese keinen weiteren Einfluß auf die Fuzzy-Beliefs und damit auf die FTT, die im Kreis  $K_i$  wandern. Sie bleiben also unverändert und werden lediglich in jedem Auswertungsschritt innerhalb des Kreises weitergereicht. Somit bleibt auch die Dominanz der Kanten im Kreis erhalten, und damit gilt für alle Stellen  $p_j$  auf dem Kreis:



$$FBM_t(p_j) = FBM_{t+x \cdot per}(p_j), x \in \mathbb{N}.$$

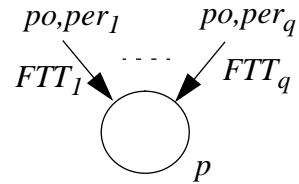
Die Werte auf den Stellen oszillieren folglich unendlich.

**Teil 2:** Die Stellen im Stellen-Vorbereich haben Status  $ss$  oder Status  $po$ .

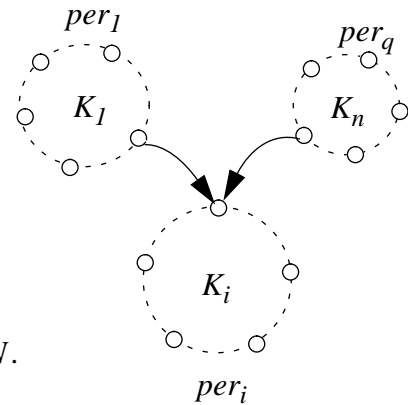
---

1. vgl. Lemma 2: die Periode  $per$  ist abhängig von der Anzahl der KS auf dem DK und beträgt maximal  $2k$ .  $k$  entspricht der Anzahl der Transitionen auf dem DK.

Nach Fall 1 dieses Beweises sind nur die Stellen im Stellen-Vorbereich der Stellen auf dem Kreis  $K_i$  beachtenswert, die den Status  $po$  tragen. Dies seien o.B.d.A. die Stellen, aus denen  $FTT_1$  bis  $FTT_q$  resultieren. Zusätzlich sei zu jeder dieser Stellen die Periode  $per_y$  des Kreises  $K_y$  bekannt, auf dem sich dieses Stellen des Stellenvorbereiches befinden.



Jede dieser Stellen oszilliert mit Periode  $per_y$ . Nach  $BRS = kgV(per_1, \dots, per_q)$  vielen Auswertungsschritten wurde die Auswertung für alle Kombinationen von  $FTT_i, i \in [1, q]$  ausgeführt. Gilt dennoch weiterhin, daß  $FBM_t(p_j) = FBM_{t+per}(p_j)$ , wobei  $t > BRS$  und  $per$  die Periode der Stelle  $p_j$  ist, oszilliert  $p_j$  unendlich, d.h.

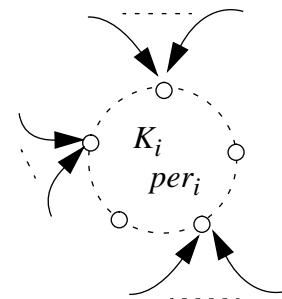


$$FBM_{t+x \cdot per}(p_j) = FBM_{t+(x+1) \cdot per}(p_j), x \in IN.$$

Haben mehrere Stellen des Kreises  $K_i$  in ihrem Stellenvorbereich Stellen mit Status  $po$ , muß das FRN mindestens

$$BRS_{min} = Max(kgV(per_w, \dots, per_z))$$

Schritte ausgewertet werden, bevor der Einfluß aller FTT, die von außerhalb die FBMs der Stellen auf  $K_i$  beeinflussen können, überprüfbar ist. Gilt nach diesen  $BRS_{min}$  vielen Auswertungsschritten weiterhin



$$\forall FBM_{t+per}(p_j) = FBM_{t+x \cdot per}(p_j), x \in IN, t > BRS_{min}$$

oszillieren die FBMs der Stelle  $p_j$  unendlich.

Ein Zyklus weist nach Theorem 2 also dann die Begrenzungskreis-Eigenschaft auf, wenn alle Stellen auf diesem Kreis den Status  $po$  haben und zusätzlich alle Stellen im Stellen-Vorbereich dieser Kreis-Stellen den Zyklus  $ss$  oder  $po$  haben.

Mit diesen Vorbemerkungen läßt sich nun der Algorithmus für die Ausführung eines FRNs bis zum Gleichgewichtszustand beschreiben.

```

procedure evaluate(FRN) {
    Weise jeder Stelle einen initialen Status zu1;
    Repeat
    
```

- (1) Berechne gleichzeitig die Fuzzy-Truth-Token an allen Transitionen entsprechend Gleichung (7);
- (2) Berechne gleichzeitig die Fuzzy-Beliefs an allen Stellen entsprechend Gleichung (8) und (9);
- (3) Berechne den Status jeder Stelle neu;
- (4) Prüfe alle Zyklen des FRNs auf Begrenzungskreis-Eigenschaft und trage sie ggf. als Begrenzungskreis ein;

**Until**

(für alle Stellen im FRN: aktueller Fuzzy-Belief = = vorheriger Fuzzy-Belief) OR  
(kein Zyklus im FRN hat Begrenzungskreis-Eigenschaft)

**EndRepeat;**

**If** (für alle Stellen im FRN: aktueller Fuzzy-Belief = = vorheriger Fuzzy-Belief)

**Then** print(„steady-state wurde erreicht“);

**Else** print(„Netz enthält einen Begrenzungskreis.“);

}

Der Algorithmus führt in einer Schleife die parallele Berechnung der neuen Fuzzy-Truth-Token und anschließend der neuen Fuzzy-Beliefs in den Schritten 1) und 2) durch. Anschließend erfolgt eine Überprüfung des Netzes auf mögliche Begrenzungskreise. Die Schleife wird solange iteriert, bis ein stabiler Zustand erreicht worden ist, d.h. bis sich in einem Auswertungsschritt zum ersten Mal kein Fuzzy-Belief einer Stelle mehr verändert hat oder mindestens ein Zyklus des Netzes die Begrenzungskreis-Eigenschaft aufweist. Implementierungsdetails zur Erkennung von Begrenzungskreisen zeigt Kapitel 6.2.

#### 4.3.4 Das Beheben von Begrenzungskreisen

In [KM96] wird ein Verfahren zum Beheben von Begrenzungskreisen vorgestellt. Dieser Abschnitt zeigt eine stark an [KM96] angelehnte Methode, die Kreise, in denen unendliche periodische Oszillationen auftreten, zum einen so aufrichtet, daß ein Gleichgewichtszustand in dem FRN erreicht werden kann. Zum anderen verfälscht sie die resultierenden Fuzzy-Beliefs der Stellen so wenig wie möglich.

Zuerst wird jedoch die Definition für den Begriff *Fuzzy-Gain* vorangestellt, der im weiteren Verlauf dieses Abschnitts mehrfach Verwendung findet:

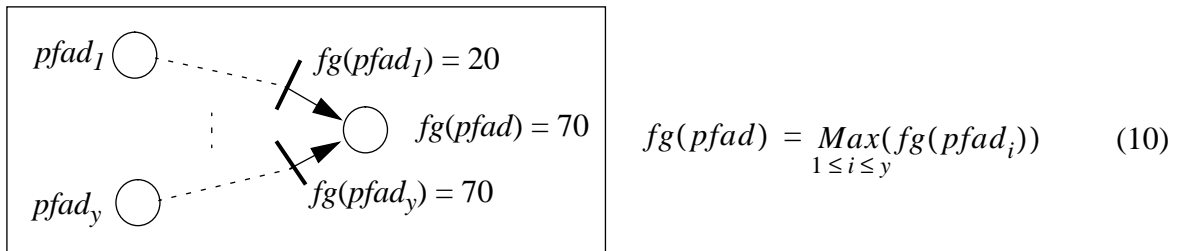
#### Definition 22: Fuzzy-Gain (FG)

Der *Fuzzy-Gain* eines azyklischen Schlußfolgerungspfades zwischen zwei Stellen berechnet sich aus dem Minimum des absoluten Fuzzy-Beliefs der Stelle, in der der Pfad startet, dem Fuzzy-Belief der Stelle, in der der Pfad endet, und den Sicherheitsfaktoren der Transitionen auf diesem Pfad. Die Berechnung des Fuzzy-Gains entlang eines einzelnen Pfades zeigt Abbildung 20 an einem Beispiel.

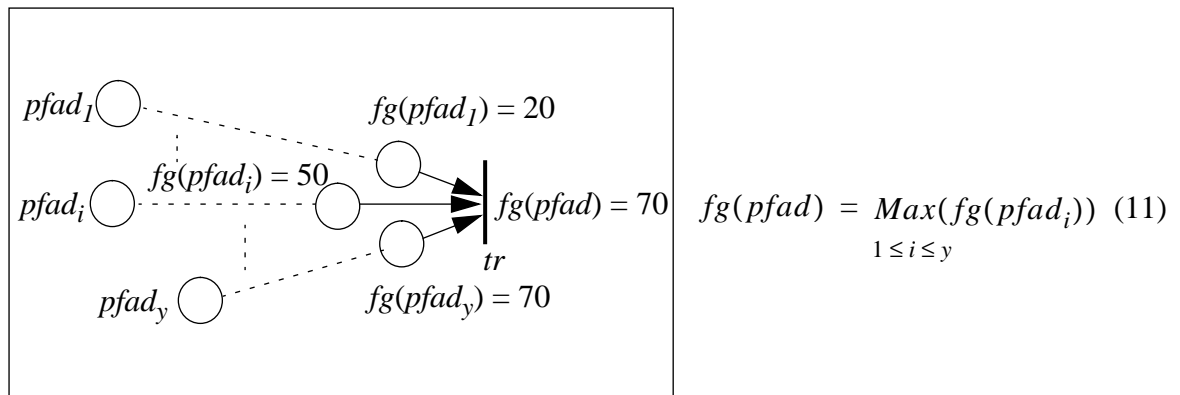
---

1. K/M-Axiome (vgl. Definition 11) erhalten den Status *ss*, alle übrigen Stellen den Status *ch*.

Existieren mehrere parallele Schlußfolgerungspfade zwischen zwei Stellen, ist für den *Gesamt-Fuzzy-Gain* das Maximum der individuellen Pfad-Gains zu wählen. Die Formel für die Berechnung dieser Fuzzy-Gains ist in Gleichung (10) zu sehen.  $fg(pfad_i)$  bezeichnet den Fuzzy-Gain des Teilpfades  $i$ , der in einem K/M-Axiom<sup>1</sup> startet,  $fg(pfad)$  den *Gesamt-Fuzzy-Gain*.



Der *Fuzzy-Gain* eines Pfades, der in den K/M-Axiomen startet und in einer geg. Transition endet, ist definiert als das Maximum der *Fuzzy-Gains* aller Pfade, die in Axiomen starten und in den Stellen im Vorbereich der Transition enden. Gleichung (11) zeigt eine Formalisierung dieses Sachverhalts. Hierbei bezeichnet  $fg(pfad)$  den Fuzzy-Gain des Pfades, der in Transition  $tr$  endet.



Die Berechnung des Fuzzy-Gains für die praktische Anwendung des FRNs für die Inferenzmaschine, genauer für die Elimination von Begrenzungskreisen in dem FRN, muß jeweils nur zwischen den K/M-Axiomen und den Transitionen auf einem solchen Begrenzungszyklus durchgeführt werden. Die Berechnung kann schrittweise gleichzeitig bei der Bestimmung der Pfade in einem FRN erfolgen. Dazu werden für jede Stelle des Pfades zunächst zwei Werte gespeichert, nämlich ein *positiver* und ein *negativer* Fuzzy-Gain. Der *positive* Fuzzy-Gain berücksichtigt alle Werte, die über positive Kanten entlang des Pfades weitergereicht, der *negative* Fuzzy-Gain entsprechend die Werte, die über negative Kanten weitergereicht werden. Der gewünschte Fuzzy-Gain in einer Transition ergibt sich dann in Abhängigkeit vom Kantenvorzeichen der in diese Transition einlaufenden Kanten zu dem positiven bzw. dem negativen Fuzzy-Gain der Stelle im Vorbereich dieser Transition.

Die Fuzzy-Gains des K/M-Axioms, in dem der Pfad startet, ergeben sich direkt aus dem absoluten Fuzzy-Belief dieser Stelle, d.h. der positive Fuzzy-Gain wird auf den AFB<sup>2</sup> gesetzt, wenn

1. vgl. Definition 11.

die Kante, die aus dem K/M-Axiom ausläuft, positiv ist, der negative Fuzzy-Gain bekommt dann den Wert Null. Ist die Kante jedoch negativ, erhält der negative Fuzzy-Gain den Wert des AFB und der positive Fuzzy-Gain erhält den Wert Null. Trägt die aus dem K/M-Axiom auslaufende Kante ein positives Vorzeichen, berechnet sich der Fuzzy-Gain des Pfades, der nur aus dem K/M-Axiom und der darauffolgenden Transition besteht, aus dem positiven Fuzzy-Gain des K/M-Axioms, ansonsten aus dem negativen. Bei der Verlängerung des Pfades um eine weitere Stelle muß das Kantenvorzeichen zwischen dieser neuen Stelle und der vorausgehenden Transition berücksichtigt werden. Ist dieses Vorzeichen positiv, ergibt sich der positive Fuzzy-Gain des Pfades, der in der neu hinzugefügten Stelle endet, aus dem Minimum des Fuzzy-Gains und der Konfidenz der vorausgehenden Transition. Der negative Fuzzy-Gain ergibt sich dann zu Null. Analog gestaltet sich die Berechnung bei einem negativen Kantenvorzeichen zwischen letzter Transition und neu zum Pfad hinzugefügter Stelle.

Abbildung 20 zeigt die Bestimmung des Fuzzy-Gains während der Berechnung eines Schlußfolgerungspfades an einem Beispiel.

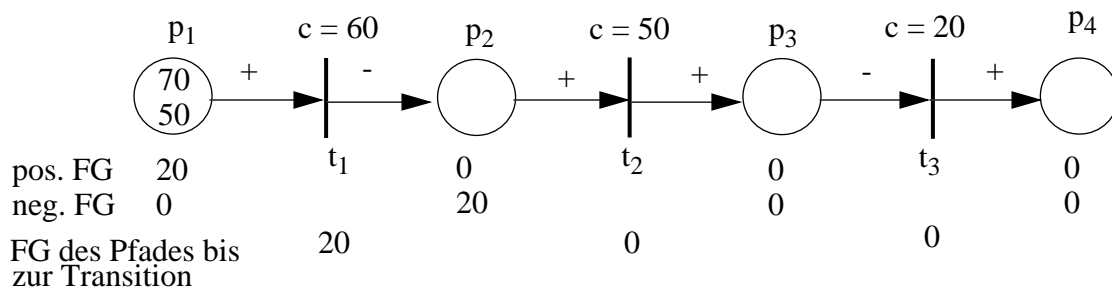


Abb. 20: Berechnung der Fuzzy-Gains eines Pfades

Anfangs bestehe der Pfad nur aus der Stelle  $p_1$  mit den Fuzzy-Beliefs (70,50). Die Fuzzy-Gain-Werte für diesen Pfad ergeben sich zu (20,0), da die Kante, die entlang des Pfades aus dem KM-Axiom  $p_1$  ausläuft, ein positives Vorzeichen trägt. Dieser Pfad werde nun um eine Transition  $t_1$  mit der Konfidenz  $c = 60$  und einer positiv markierten Kante zwischen  $p_1$  und dieser Transition verlängert. Der Fuzzy-Gain dieses Pfades ergibt sich somit zu 20. Bei der Verlängerung des Pfades um eine Kante mit negativem Vorzeichen und einer Stelle  $p_2$  berechnet sich der negative Fuzzy-Gain in dieser Stelle aus dem Minimum zwischen dem Fuzzy-Gain der Transition  $t_1$  von 20 und der Konfidenz von 60 zu 20. Der positive Fuzzy-Gain der Stelle  $p_2$  erhält den Wert Null. Da die Kante zwischen  $p_2$  und  $t_2$  ein positives Vorzeichen trägt, der positive Fuzzy-Gain von  $p_2$  jedoch den Wert Null hat, berechnet sich der Fuzzy-Gain des Pfades, der in Transition  $t_2$  endet, ebenfalls zu Null und damit alle weiteren Fuzzy-Gains entlang des Pfades. Der Gesamt-Fuzzy-Gain eines Pfades, der mit dem Pfadstück  $p_1 t_1 p_2 t_2$  startet, trägt also den Wert Null, egal, in welcher Stelle er endet.

Das Ziel des nun vorgestellten Verfahrens ist es, Begrenzungskreise mit minimalen Verfälschungen in den Berechnungen der Fuzzy-Beliefs der Stellen aufzubrechen. Da auf einem Kreis

- 
- Die Abkürzung AFB steht für „Absoluter Fuzzy-Belief“ und bezeichnet die absolute Differenz aus dem positiven und negativen Fuzzy-Belief einer Stelle, vgl. Definition 12.

mit Begrenzungskreis-Verhalten alle Kanten permanent dominant sein müssen, reicht es aus, die Dominanz einer dieser Kanten zu unterdrücken. Das wird durch Erhöhen des Schwellwertes einer Transition auf dem Kreis erreicht, was das Schalten dieser ausgewählten Transition verhindert.

Von der Wahl der Transition, deren Schwellwert erhöht wird, hängt die Güte der anschließenden Fuzzy-Beliefs der Stellen im Gleichgewichtszustand und damit auch die Güte der Analyseergebnisse beim Einsatz eines FRNs zur Wissensrepräsentation ab. Es muß deshalb gefordert werden, daß das Verfahren die Transition so auswählt, daß sie auf dem Pfad mit dem niedrigsten Fuzzy-Gain liegt, das Ergebnis also am wenigsten verfälscht wird.

Nachdem der Gesamt-Fuzzy-Gain für einen Schlußfolgerungspfad zwischen den K/M-Axiomen und jeder einzelnen Stelle im Vorbereich einer Transition auf dem Zyklus mit Begrenzungskreis-Verhalten bestimmt worden ist, berechnet sich der neue Schwellwert an der Transition folgendermaßen: in Abhängigkeit vom Vorzeichen der Kante, die von den Stellen im Vorbereich der entsprechenden Transition in die Transition einläuft, werden die Fuzzy-Gains, die über diese Kanten in die Transition hinlaufen, mit der Minimumfunktion verknüpft. Das Ergebnis ergibt den neuen Schwellwert der Transition. Abbildung 21 verdeutlicht diesen Zusammenhang.

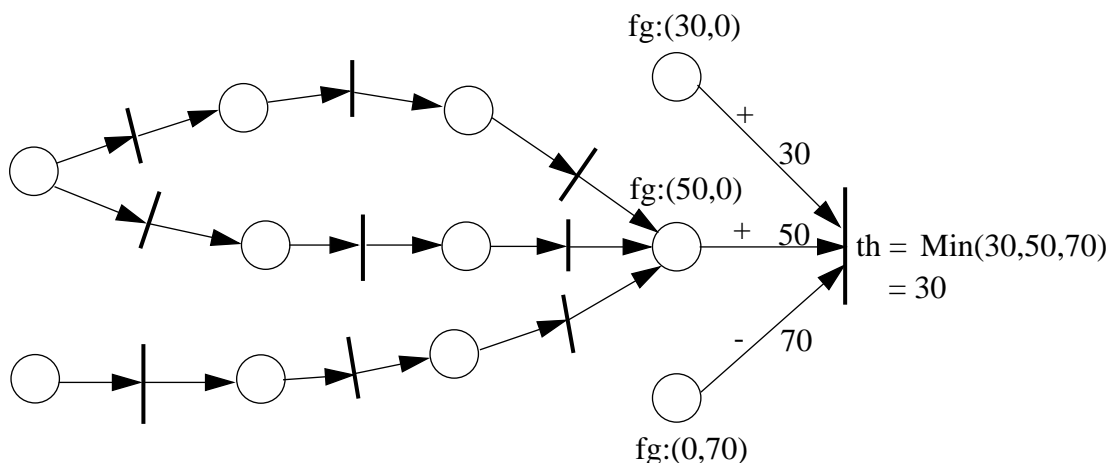


Abb. 21: Berechnung des neuen Schwellwertes einer Transition

Der Fuzzy-Gain ist laut Definition 22 nur für azyklische Pfade definiert. Aus diesem Grund ist bei der Berechnung eines Schlußfolgerungspfades, der in einen Kreis mit Begrenzungskreis-Verhalten hineinführt und vorher möglicherweise andere Zyklen durchlaufen hat, der Fuzzy-Gain nur für den azyklischen Anteil des Pfades zu ermitteln.

Der letzte Abschnitt beschreibt bisher nur, wie der Schwellwert einer Transition zu berechnen ist, die auf dem Kreis mit Begrenzungskreis-Verhalten liegt. Was jetzt noch fehlt ist, wie die Auswahl dieser Transition, die ja auf dem Pfad mit dem kleinsten Fuzzy-Gain liegen soll, vorzunehmen ist. Diese Wahl ist jedoch mit Problemen behaftet, da das Unterdrücken der Dominanz einer Kante in einem Kreis mit Begrenzungskreis-Verhalten einen Begrenzungskreis in

einen Nachbarzyklus induzieren kann, wie Abbildung 22 verdeutlicht. Da die Kantenvorzeichen hierzu nicht beachtet werden, sind sie in der Abbildung auch nicht eingetragen.

Wählt man beispielsweise Transition  $t_3$  aus, um mit ihr die permanente Dominanz der Kante  $d_1$  zu unterdrücken, wird damit die Kante  $d_2$  automatisch dominant. Das kann dazu führen, daß durch das Aufbrechen des Begrenzungskreises  $p_2p_3p_4p_5$  jetzt der Kreis  $p_1p_2p_1$  Begrenzungskreis-Verhalten zeigt.

Um dieses Problem adäquat zu berücksichtigen, bestimmt der nachfolgend aufgeführte Algorithmus nach dem oben beschriebenen Verfahren die potentiellen neuen Schwellwerte aller Transitionen auf dem Begrenzungskreis. Für die Elimination eines Begrenzungskreises wählt er dann die Transition mit potentiell niedrigstem neuem Schwellwert aus. Diese Transition erhält den gerade neu berechneten Schwellwert, und das FRN wird erneut auf Begrenzungskreise getestet. Erreicht das FRN jetzt einen Gleichgewichtszustand, ist die Berechnung abgeschlossen. Andernfalls werden wiederum für alle Transitionen nach obigem Verfahren neue Schwellwerte bestimmt, diejenige mit dem niedrigsten ausgewählt, usw.

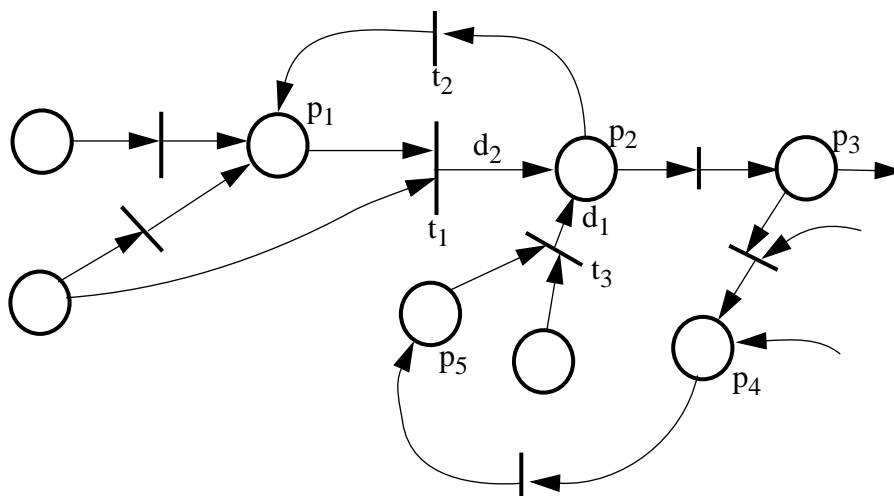


Abb. 22: Induktion von Begrenzungskreis-Verhalten in Nachbarzyklen

Auf diese Weise werden alle Zyklen mit Begrenzungskreiseigenschaft erkannt und aus dem FRN entfernt.

Der nachfolgend aufgeführte Algorithmus faßt die Strategie zur Auswertung eines FRNs noch einmal zusammen:

**procedure** evaluate(FRN fpn){

- 1) Werte das übergebene FRN aus;
- 2) **If** (fpn enthält Begrenzungskreise) {
- 3)   **do**{

- 4) Wähle einen Begrenzungskreis aus;
- 5) Berechne den Fuzzy-Gain zu jeder Transition entlang der Schlußfolgerungspfade,  
die diesen Zyklus enthalten;
- 6) Wähle die Transition auf dem Kreis aus, die den niedrigsten Fuzzy-Gain hat;
- 7) Setze den Schwellwert der Transition auf den Fuzzy-Gain-Wert;
- 8) Werte das FRN erneut aus;
- 9) **until** (FRN ist im steady-state);
- 10) **until** (FRN ist im steady-state);
- 11)}

## 5 Überführung eines GFRNs in ein FRN: Die Expansion

Kapitel 3 führt die Generic Fuzzy Reasoning Nets ein, mit deren Hilfe das generische Wissen zur Analyse einer relationalen Datenbank spezifiziert werden kann. Dieses Modell ermöglicht die Beschreibung des Wissens auf abstrakte Weise. Kapitel 4 beschreibt ein ausführbares Modell für eine Inferenzmaschine, das auf unscharfen Petrinetzen basiert. Der nun folgende Abschnitt als Bindeglied zwischen diesen beiden Kapiteln stellt die Überführung einer abstrakten GFRN-Spezifikation in ein ausführbares FRN dar. Diese Übersetzung wird im folgenden *Expansion* genannt.

Die Expansion ist kein zentrales Thema dieser Diplomarbeit, wie die Spezifikation und Implementierung der Ausführungsmaschine auf Basis der FRNs. Dieses Kapitel stellt deshalb nur eine grobe Idee vor, wie die GFRNs in FRNs übersetzt werden. Eine Formalisierung und Implementierung des Expansionsalgorithmus erfolgt in weiterführenden Arbeiten und im *Varlet-Projekt*<sup>1</sup>.

### 5.1 Die Abbildung einer GFRN-Regel in einem FRN

Eine GFRN-Regel beschreibt generisches Wissen über relationale Datenbanken in Form von Heuristiken. Zur Expansion einer solchen Regel müssen Analyseoperationen<sup>2</sup> zunächst Fakten<sup>3</sup>, die eine solche Regel spezifiziert, in der zu analysierenden Datenbank finden. Diese Analyseoperationen zur Herleitung einer initialen Menge von Fakten sind an die Prädikate geknüpft, die als *starke Axiome*<sup>4</sup> klassifiziert wurden. Betrachtet man beispielsweise die Implikation  $i_1$  der GFRN-Spezifikation aus Abbildung 5, wird der Applikationscode zunächst nach *cyclic-join* Clichés durchsucht. Ausgehend von den Ergebnissen dieser Suche ist Implikation  $i_1$  für jede Attributmengemenge  $v$ , über die ein *cyclic-join* Cliché gefunden wurde, zu expandieren. Obwohl der Applikationscode des Szenarios aus Kapitel 1.2 nur ein solches *cyclic-join* Cliché enthält, existieren in realen relationalen Datenbankanwendungen gewöhnlich mehrere Anfragen dieser Form. Diese können jeweils verschiedene Attributmengen  $v$  spezifizieren. Jedes gefundene *cyclic-join* Cliché über eine andere Attributmengemenge  $v$  führt zu einer Instanz des **cycl\_join** Prädikats des GFRNs. Zu jeder GFRN-Regel existieren im zugehörigen FRN also im allgemeinen mehrere Instanzen.

Weiterhin stellt sich die Frage, wie die Instanz einer solchen Regel im FRN abzubilden ist. Intuitiv wird für jede Instanz einer Implikation des GFRNs eine Transition im FRN erzeugt und diese mit den zugehörigen Stellen verbunden, die die Instanzen der Prädikate im Vor- und Nachbereich der Implikation repräsentieren. Die Semantik einer GFRN-Regel<sup>5</sup> entspricht jedoch den Formeln der Prädikatenlogik erster Ordnung [DRSW86]. Eine Formel der Prädikatenlogik der Form  $A \rightarrow B$  impliziert auch immer ihre Kontraposition, also die Formel  $\neg B \rightarrow \neg A$ . Die eben vorgeschlagene Abbildung einer GFRN-Implikation durch eine Transition im FRN ist damit also nicht ausreichend. Um die Semantik der GFRNs adäquat im FRN zu modellieren,

1. vgl. Kapitel 1.3.

2. Diese Analyseoperationen sind an die starken Axiome gebunden; vgl. Definition 1 in Kapitel 3.4.

3. z. B. das Vorhandensein von *cyclic-join* Clichés.

4. Die unterschiedlichen Axiomstypen wurden bereits in Abschnitt 3.2 vorgestellt.

5. Die Semantik der GFRNs wird in Abschnitt 3.4 beschrieben.

muß auch hier zu jeder GFRN-Regel die Kontraposition explizit mitberücksichtigt werden.

Bevor Kapitel 5.2 den Expansionsalgorithmus informal beschreibt, erläutert der folgende Abschnitt deshalb zunächst die für die Expansion wichtige *Kontrapositionsregel*.

Wie bereits oben erwähnt, lautet die Kontraposition zu einer einfachen Regel  $A \rightarrow B$   $\neg B \rightarrow \neg A$ . Die Expansion einer solchen Regel, die nur aus zwei Prädikaten besteht, ist einfach. Abbildung 23 zeigt eine Übersetzung einer aus 3 Prädikaten bestehenden GFRN-Regel in das FRN-Modell. Diese Regel besteht aus den Prädikaten  $p_1$ ,  $p_2$  und  $p_3$ , der Implikation  $i$  und trägt die Aussage  $p_2 \wedge \neg p_3 \rightarrow p_1$ . Es wird zur Vereinfachung des Beispiels davon ausgegangen, daß für jedes der Prädikate  $p_1$ ,  $p_2$  und  $p_3$  jeweils nur eine Instanz im FRN existiert. Die Transition  $tr_1$  modelliert im FRN die Aussage der GFRN-Regel  $p_2 \wedge \neg p_3 \rightarrow p_1$ . Ihre Kontraposition lautet:  $\neg(p_1) \rightarrow \neg(p_2 \wedge \neg p_3)$ . Diese Aussage muß vor ihrer Übersetzung in ein FRN mit Hilfe logischer Äquivalenzumformungen so normiert werden, daß auf der rechten Seite nur noch ein Prädikat steht. Dazu werden die DeMorgan'sche Regel [Schö92] sowie die Schreibvereinfachung  $a \rightarrow b$  für die Formel  $\neg a \vee b$  [Schö92] angewendet:

$$\begin{aligned} \neg(p_1) \rightarrow \neg(p_2 \wedge \neg p_3) &| \text{DeMorgan} \\ \neg p_1 \rightarrow \neg p_2 \vee p_3 &| \neg a \vee b \equiv a \rightarrow b \\ p_1 \vee \neg p_2 \vee p_3 &| \text{DeMorgan} \\ \neg(\neg p_1 \wedge p_2) \vee p_3 &| \neg a \vee b \equiv a \rightarrow b \\ \neg p_1 \wedge p_2 \rightarrow p_3 & \end{aligned}$$

Aufgrund der Kommutativität der logischen *UND*-Verknüpfung [Schö92] ergibt sich noch eine weitere Kontrapositionsregel. Wandelt man die Ausgangsgleichung  $\neg(p_1) \rightarrow \neg(p_2 \wedge \neg p_3)$  zunächst um in  $\neg(p_1) \rightarrow \neg(\neg p_3 \wedge p_2)$  und wendet dann, wie gerade gezeigt, obige Äquivalenzumformungen an, ergibt sich somit für eine aus drei Prädikaten bestehende GFRN-Regel neben der Kontrapositionsregel  $\neg p_1 \wedge p_2 \rightarrow p_3$  noch die Regel  $\neg p_1 \wedge \neg p_3 \rightarrow \neg p_2$ . Die erste Kontraposition wird in Abbildung 23 durch Transition  $tr_2$ , die zweite Kontraposition durch Transition  $tr_3$  abgebildet. Die Konfidenzen der Implikationen einer GFRN-Regel übertragen sich unverändert auf die Konfidenzen der Transitionen im FRN, die diese Implikation und ihre Kontraposition repräsentieren. Negierte Prädikate werden mit der jeweiligen Transition im FRN durch Kanten mit einem negativen Kantenvorzeichen versehen.

Da die starken und aufgeschobenen Axiome eines GFRNs als unumstößliche Fakten gelten<sup>1</sup>, können die Werte dieser Prädikate während des Inferenzprozesses im FRN nicht verändert werden. Deshalb dürfen auch durch die Modellierung der Kontraposition zu einer GFRN-Regel die Stellen im FRN, die solche Axiome im GFRN repräsentieren, keine eingehenden Kanten bekommen.

---

1. vgl. Abschnitt 3.2.

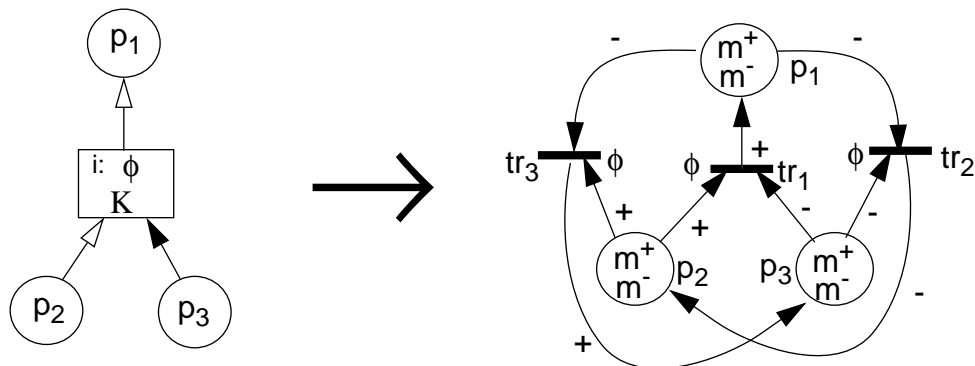


Abb. 23: Expansion einer einfachen GFRN-Regel mit 3 Prädikaten

Es ergibt sich somit die folgende Ausnahme zur Kontrapositionsregel: für GFRN-Regeln, die ein starkes oder aufgeschobenes Axiom im Vorbereitungsbereich der Implikation enthalten, wird keine Kontrapositionsregel modelliert. GFRN-Regeln, die hingegen über ein starkes oder aufgeschobenes Axiom im Nachbereich der Implikation verfügen, werden nur mit Hilfe der Kontraposition expandiert.

## 5.2 Der Inferenzalgorithmus

Die Inferenzmaschine unterstützt aus Gründen der Laufzeitminimierung eine verspätete Ausführung der Analyseoperationen, die an aufgeschobene Axiome<sup>1</sup> gebunden sind. Da deshalb Informationen erst verspätet abgeleitet werden, kann auch die Expansion einer Wissensspezifikation nur in mehreren Schritten und von Auswertungsphasen unterbrochen stattfinden. So ist es möglich, daß einmal gewonnene Analyseergebnisse durch das Hinzufügen neuer Erkenntnisse, wie es bei der Expansion der aufgeschobenen Axiome der Fall sein könnte, revidiert werden müssen. Betrachtet man dazu Abbildung 5 aus Kapitel 3, ist dies zum Beispiel der Fall, wenn eine positive Notwendigkeit<sup>2</sup> für die Schlüsseleigenschaft eines Attributs gefunden wurde (**key**), die Analyseoperation *validKey* jedoch ein oder mehrere Gegenbeispiele gegen diese Eigenschaft in den Daten findet. Die Inferenzmaschine muß somit nichtmonotones Schließen unter Unsicherheit unterstützen. Auch diese Anforderung wird mit Hilfe der Kontrapositionsregel gewährleistet.

Der Inferenzalgorithmus muß bei der Expansion zusätzlich berücksichtigen, ob es sich bei den zu expandierenden Prädikaten um einfache Prädikate oder um Axiome handelt, da er starke, aufgeschobene und schwache Axiome, wie bereits erwähnt, jeweils zu unterschiedlichen Zeitpunkten expandieren muß. Abbildung 24 zeigt einen Algorithmus für diesen Vorgang. Zuerst führt der Algorithmus alle Analyseoperationen aus, die an starke Axiome gebunden sind. Entsprechend der Ergebnisse dieser Operationen erzeugt er für jeden „Treffer“ der Analyseopera-

1. vgl. Abschnitt 3.2.

2. vgl. Kapitel 2.6: die Notwendigkeit ist ein Begriff aus der Possibilistischen Logik und gibt eine untere Schranke für die Sicherheit einer Formel an.

tionen eine Stelle für die Aussage des zugehörigen Prädikats im FRN. Die formalen Parameter der Prädikate werden dabei in den erzeugten Stellen durch Konstanten als aktuelle Parameter ersetzt, die die Analyseoperation ermittelt hat. Ein Beispiel für eine solche Analyseoperation ist das Durchsuchen des Applikationscodes nach bestimmten Clichés, wie etwa den *select-distinct* Clichés. Für jedes gefundene Cliché wird mit den entsprechenden Attributnamen als Konstanten, die in dieses Cliché eingehen, eine Stelle im FRN erzeugt. In Abhängigkeit von der jeweiligen Analyseoperation trägt der Algorithmus die positiven und negativen Fuzzy-Beliefs als initiale und unumstößliche Werte ein.

```

algorithm inferenz
begin
  erzeuge leeres FRN;
  erzeuge entsprechend den Ergebnissen der an starke Axiome gebundenen Analyseoperationen neue Stellen im FRN;
  repeat
    erzeuge Stellen entsprechend der schwachen Axiome (interaktiv hinzugefügte Annahmen);
    expandiere das FRN vorwärts;
    expandiere das FRN rückwärts, und
    führe die Analyseoperationen der aufgeschobenen Axiome aus, die Prädikaten für neu erzeugte Stellen zugeordnet sind;
    füge die Transitionen ein;
  repeat werte das FRN aus until FRN ist im Gleichgewichtszustand;
  until FRN kann nicht mehr vergrößert werden;
end;

```

Abb. 24: Der Inferenzalgorithmus

Die sich anschließenden Schritte erfolgen iterativ, bis keine weiteren Informationen mehr ableitbar sind. Zunächst hat der Reengineer die Möglichkeit, interaktiv neue Annahmen über die jeweilige Datenbankanwendung hinzuzufügen. Diese Option entspricht wiederum der Forderung nach einem nichtmonotonen Inferenzprozeß und nach Interaktion durch den Anwender. Anschließend wird das FRN basierend auf dieser initialen Stellenmenge mit ihren aktuellen Parametern in weiteren Expansionsschritten um weitere Stellen ergänzt. Dies geschieht in einem *Vorwärts-* und in einem *Rückwärts-Expansionsschritt*.

Der *Vorwärtsschritt* durchsucht das bisherige FRN nach Aussagen, die Instanzen der Prädikate im Vorbereich einer Implikation  $i$  im zugehörigen GFRN darstellen. Dann testet er, ob alle formalen Parameter von  $i$  vollständig durch die Konstanten der gefundenen Stellen im FRN bestimmt sind. Falls auch dieser Test erfolgreich verläuft, erzeugt der Algorithmus für jedes Prädikat im Nachbereich von Implikation  $i$  Stellen entsprechend der aktuellen Parameter, sofern sie noch nicht vorhanden sind.

Der *rückwärtige Expansionsschritt* wird für die Expansion derjenigen GFRN-Regeln benötigt, in deren Vorbereich ein aufgeschobenes Axiom steht. Die Analyseoperationen, die an aufgeschobene Axiome gebunden sind, werden nur dann ausgeführt, wenn ein positiver Fuzzy-Be-

lief-Wert auf die Notwendigkeit einer Aussage hinweist. Dann muß der Rückwärtsschritt eine Stelle für dieses aufgeschobene Axiom im Vorbereich einer Implikation erzeugen, deren Nachbereich u.U. schon vollständig expandiert ist. Deshalb testet er vom Nachbereich einer Implikation ausgehend, ob alle formalen Parameter einer Implikation  $i$  durch die aktuellen Parameter der gefundenen Aussagen bestimmt sind. In diesem Fall werden im FRN Stellen für jedes Prädikat im Vorbereich der Implikation  $i$  entsprechend der aktuellen Parameter erzeugt.

Als nächstes sind nun für die erzeugten Stellen im FRN und in Übereinstimmung mit der zugrundeliegenden GFRN-Spezifikation die Transitionen zu erzeugen. Für jede Implikation im GFRN sind wegen der oben erläuterten Kontrapositionsregel mindestens zwei Transitionen einzufügen. Eine Ausnahme bilden jedoch Kontrapositionen, die starke oder aufgeschobene Axiome involvieren. Hier wird die jeweilige Regel vernachlässigt, die einlaufende Kanten in die für solche Axiome erzeugten Stellen im FRN zur Folge haben würde, d.h. die Regel an sich oder aber ihre Kontraposition.

Der Algorithmus expandiert also eine GFRN-Regel ausgehend von den Ergebnissen der Analyseoperationen, die an die starken Axiome gebunden sind. Bisher wurde jedoch noch keine Aussage über die Güte dieser Ergebnisse gemacht. Das soll jetzt nachgeholt werden. Beispielsweise liefert eine Analyseoperation, die den Applikationscode nach Clichés durchsucht, definitive Fakten über die Existenz bzw. Abwesenheit eines Anfragemusters über eine bestimmte Attributmenge. Nur in dem Fall, daß ein Cliché tatsächlich gefunden werden konnte, erzeugt der Algorithmus auch eine Stelle im FRN. Anders verhält es sich hingegen bei einer Analyseoperation, die die Ähnlichkeit von Attributnamen bewerten soll, um daraus später eine äquivalente Bedeutung schließen zu können<sup>1</sup>. Die Werte für eine Ähnlichkeit zwischen zwei Attributen sind unscharf und können zwischen 0 und 1 liegen. Es stellt sich nun die Frage, nach welchen Kriterien für Analyseoperationen, die unscharfe Aussagen liefern, Stellen erzeugt werden sollen. Es ist sicherlich nicht sinnvoll, für Attribute mit geringer Ähnlichkeit eine Stelle anzulegen. Liefert eine solche Operation etwa das Ergebnis, daß die Attributnamen *Name* und *Nummer* nur zum Grad 0.01 auf einer Skala von 0 bis 1 ähnlich sind, läßt sich durch die weitere Betrachtung der Stelle „nsimilar((Name,Nummer),0.01,0)“ voraussichtlich keine äquivalente Bedeutung der Attribute *Name* und *Nummer* schließen. Das Erzeugen der Stelle „nsimilar((Name,Nummer),0.01,0)“ liefert also für den Inferenzprozeß keine neue, verwertbare Information. Expandiert der Algorithmus ausgehend von dieser Stelle trotzdem weitere GFRN-Implikationen, sind sie auch bei der Expansion aller weiteren Regeln und später bei der Inferenz stets mitzubersichtigen und würden die Inferenz unnötig verlangsamen.

Aus diesem Grund hat der Reengineer die Möglichkeit, die Expansion von GFRN-Implikationen auf Grundlage des bisherigen FRNs zu steuern. Dazu werden die absoluten Fuzzy-Belief-Werte der Stellen im FRN betrachtet, die in einem Vorwärts- oder Rückwärtsschritt zum Erzeugen weiterer Stellen führen. Ist mindestens einer dieser Fuzzy-Beliefs kleiner als ein zu bestimmender Wert aus dem Intervall  $[0,1]$  – er wird im Algorithmus mit  $\epsilon$  bezeichnet –, expandiert der Algorithmus die momentan betrachtete GFRN-Implikation für diese Stellen nicht weiter. Die Zahl  $\epsilon$  soll es dem Reengineer ermöglichen, eine untere Schranke für den absoluten Fuzzy-Belief festzulegen, ab dem die Berücksichtigung einer Aussage für bestimmte Parameter noch sinnvoll erscheint.

---

1. vgl. Implikation  $i_8$  in Abbildung 5 auf Seite 28.

Der Algorithmus wertet das so erzeugte FRN nun solange aus, bis es den Gleichgewichtszustand erreicht<sup>1</sup>. Anschließend betrachtet er die Stellen, die Instanzen von Prädikaten in der GFRN-Implikationen darstellen, in denen sich auch aufgeschobene Axiome befinden. Tragen diese Stellen einen absoluten Fuzzy-Belief größer als  $\varepsilon$ , werden nun die Analyseoperationen der aufgeschobenen Axiome ausgeführt, die Bestandteil der zugehörigen GFRN-Implikation sind. Entsprechend der Ergebnisse dieser Analyse erweitert der Algorithmus das FRN um weitere Stellen und Transitionen. Wiederum schließt sich eine weitere Auswertung des so gewonnenen FRNs an, bis ein stabiler Zustand angenommen wird.

Die Inferenz ist beendet, wenn keine weiteren Knoten mehr zum FRN hinzugefügt werden können und sich das Netz in einem Gleichgewichtszustand befindet.

Kapitel 7 zeigt zusammenfassend die Anwendung des Inferenzalgorithmus, also die Expansion und Auswertung für die GFRN-Spezifikation aus dem Eingangsszenario.

---

1. vgl. Abschnitt 4.3

## 6 Design und Implementierung

Das Design, das aus den in Kapitel 4 vorgestellten Konzepten resultiert, wird in Abschnitt 6.1 vorgestellt. Anschließend folgt in 6.2 eine kurze Erläuterung einiger Implementierungsdetails zur Erkennung und Elimination von Begrenzungskreisen. Die Werkzeuge zur Implementierung der Ausführungsmaschine sind Bestandteil der Beschreibungen in Abschnitt 6.4.

### 6.1 Das Design der Ausführungsmaschine

Abbildung 25 zeigt ein verkürztes Design der Ausführungsmaschine. Detailliertere Angaben zu der Struktur und den Methoden der Klassen sind Anhang B zu entnehmen. Die Klasse zur Implementierung der in Kapitel 4 vorgestellten Fuzzy Reasoning Nets trägt im Design den Namen **FPN**. Die Begriffe FPN und FRN werden in diesem Kapitel deshalb synonym verwendet.

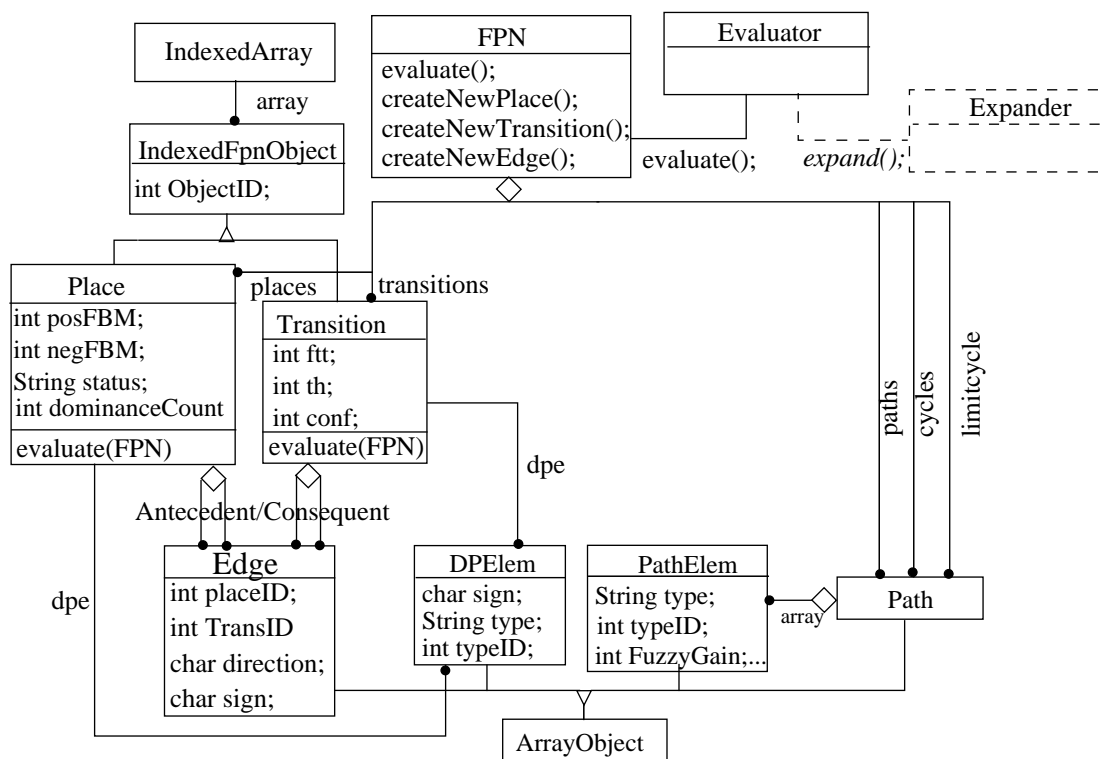


Abb. 25: Grobdesign der Ausführungsmaschine

Die Klasse **Evaluator** soll die Expansion einer GFRN-Spezifikation in ein FRN über das Modul **Expander** und die anschließende Ausführung des FRNs über das Modul **FPN** steuern. Da die Klasse **Expander** in dieser Diplomarbeit jedoch nicht implementiert wurde, ist sie im Design gestrichelt dargestellt.

Die ausführbare Methode (main-Methode) der Klasse **Evaluator** muß eine Definition des auszuwertenden FRNs sowie den Aufruf *evaluate()* enthalten. Ein Beispiel für den Aufbau einer solchen main-Methode wird ebenfalls in Anhang B gezeigt.

Ein **FPN** besteht aus einer Menge von Stellen (**Place**) und Transitionen (**Transition**). Die Stellen und Transitionen sind beide vom Typ **IndexedFpnObject**, um einen eindeutigen Index auf den Knoten eines FRNs definieren zu können. Diese Indizes werden zur Definition der Kanten (**Edge**) zwischen Stellen und Transitionen benötigt. Die Kanten sind direkt in den Knoten des FRNs gespeichert. Dabei wird zwischen Kanten im Vor- und im Nachbereich eines Knotens unterschieden. Bei der Definition einer Kante zwischen zwei FPN-Knoten vom Typ **IndexedFpnObject** sind die IDs der dazugehörigen Stelle und Transition sowie ihre Richtung und ihr Vorzeichen anzugeben.

Zusätzlich zur Aussage, die eine Stelle repräsentiert, und ihrem positiven und negativen Fuzzy-Belief enthält eine Stelle eine Variable *status* und eine Variable *dpe* zur Erkennung und Elimination von Begrenzungskreisen. Details zur Bedeutung dieser Variablen werden in Abschnitt 6.2 näher erläutert. Auch in den Transitionen ist diese Variable *dpe* zusätzlich zu ihren Fuzzy-Truth-Token (int *ft*;) , ihrem Schwellwert (int *th*;) und ihrer Konfidenz (int *conf*;) enthalten.

Ein FPN enthält außerdem die Komponenten *paths*, *cycles* und *limitcycle*. Diese Variablen sind Instanzen der Klasse **Path**. Die Variable *paths* enthält alle azyklischen Schlußfolgerungspfade des auszuwertenden FRNs. Alle in *path* gespeicherten Pfade beginnen mit einer Stelle, die über keine Eingangskanten verfügt. Die Variable *cycles* umfaßt alle Zyklen und *limitcycle* eventuell gefundene Begrenzungskreise des Netzes.

Jeder Pfad ist aus einzelnen Pfadelementen von Typ **PathElem** aufgebaut. Jedes dieser Pfadelemente enthält Typ und ID des **IndexedFpnObjects**, das sie repräsentieren. Jeder Pfad beinhaltet des weiteren mehrere Variablen zur Speicherung verschiedener Fuzzy-Gains. Diese werden zur Auswahl einer geeigneten Transition benötigt, deren Schwellwert zur Eliminierung von Begrenzungskreisen verändert wird<sup>1</sup>. Der Fuzzy Gain soll auf azyklischen Pfaden gewährleisten, daß das Beheben eines Begrenzungskreises nur mit einer minimalen Verfälschung der resultierenden Analyseergebnisse einhergeht.

Während die Berechnung der Pfade in *paths* nur dazu dient, die Fuzzy-Gains der Pfade und eventuell vorhandene Zyklen zu bestimmen, werden die Zyklen in *cycles* und *limitcycle* zur Erkennung und Eliminierung der Begrenzungskreise herangezogen.

Die Klassen **Edge**, **PathElem**, **Path** und **DPElem** werden unter der Oberklasse **ArrayObject** zusammengefaßt.

## 6.2 Implementierungsdetails

Kapitel 4.3.4 stellt ausführlich das Verfahren zum Eliminieren von Begrenzungskreisen in einem FRN vor. Abschnitt 6.2 geht nun detaillierter auf die Strategie zur Erkennung und Lokalisierung der Begrenzungskreise ein. Abbildung 26 stellt deshalb den Algorithmus aus Kapitel 4.3.3 noch einmal dar. Dieser wurde zur Erläuterung der Strategie an wesentlichen Stellen verfeinert.

Der Algorithmus weist zu Anfang der Auswertung in Zeile 1 jeder Stelle des Netzes einen initialen Status zu. Dabei erhalten alle Stellen, deren Vorbereich leer ist, den Status *ss* für „steady-

---

1. vgl. Kapitel 4.3.4.

state“ und alle übrigen Stellen den Status *ch* für „changing“. Außerdem erhält jede Stelle und jede Transition eine Liste von *Dominanzpfadausdrücken* (*DPA* bzw. *DPE*: engl.: *dominance path expression*). Ein DPA dient der Erkennung von Dominanzkreisen<sup>1</sup> (DK) im auszuführenden FRN. Er dokumentiert zu diesem Zweck, entlang welchen Pfades, d.h. über welche *dominanten* Kanten, die Berechnung eines aktuellen Fuzzy-Truth-Token bzw. der Fuzzy-Beliefs stattgefunden hat. Jeder DPA setzt sich aus einzelnen Elementen vom Typ **DPElem**<sup>2</sup> zusammen. Diese bestehen aus dem Kantenvorzeichen zwischen vorausgehendem und aktuellem Knoten im FPN entlang des Pfades, den der DPA repräsentiert, und einer Kennung<sup>3</sup> des aktuellen Knotens im FPN. Die Kennung des Knotens besteht aus seinem Typ (p oder t) und seiner ID im FPN. Ein DPA ist somit ein Pfad, der nur aus Knoten des FRNs besteht, die durch *dominante* Kanten miteinander verbunden sind.

```

procedure evaluate(FPN) {
  (1) Weise jeder Stelle einen initialen Status zu4;
  (2) Weise jeder Stelle einen initialen Dominanzpfadausdruck (DPA) zu;
  Repeat
    // Berechnung der Fuzzy-Truth-Token entsprechend Gleichung (7)
    (3) Repeat für alle Transitionen{
      Berechne für die aktuelle Transition ein neues FTT;

      //Berechnung der neuen DPAs
      kopiere (falls die Stelle kein K/M-Axiom ist) die DPAs der Stelle, die mit
      der aktuellen Transition über eine dominante Kante verbunden ist, und speichere
      sie als neue DPAs der Transition;

      verlängere jeden einzelnen DPA um ein Element (Vorzeichen der dominanten
      Kante, „t“, ID der aktuellen Transition);
    }
    //Berechnung der Fuzzy-Beliefs entsprechend Gleichungen (8) und (9)
    (4) Repeat für alle Stellen{
      Berechne für die aktuelle Stelle neue Fuzzy-Beliefs;

      //Berechnung der neuen DPAs
      kopiere (falls vorhanden) die DPAs der Transitionen5, die mit der aktuellen
      Stelle über dominante Kanten verbunden sind, und speichere sie als neue
      DPAs der Stelle;

      verlängere jeden einzelnen DPA um ein Element (Vorzeichen der dominanten
      Kante, „p“, ID der aktuellen Stelle);

      füge einen neuen DPA der Form („p“, ID der aktuellen Stelle) zu den DPAs
      hinzu;
    }
    //Berechne den Status jeder Stelle neu
    (5) Repeat für jede Stelle {

```

---

1. vgl. Kapitel 4.3.3.

2. vgl. Abbildung 25.

3. vgl. Definition der Klasse **DPElem** auf Seite 115.

4. K/M-Axiome (vgl. Definition 11) erhalten den Status *ss*, alle übrigen Stellen den Status *ch*.

5. Es kann zwei solche Kanten geben: eine positive und eine negative.

```

If alle Stellen im Stellen-Vorbereich der aktuellen Stelle haben Status ss
  Then weise der aktuellen Stelle den Status ss zu;
If ((status == ch) OR (status == po)){
  Then Teste, ob die Stelle auf einem Dominanzkreis liegt, und speichere in
    der Variable DominanceCount, seit wie vielen Auswertungsschritte
    dieser bereits existiert;
  Fi;
}
//Prüfe alle Zyklen des FRNs auf Begrenzungskreis-Eigenschaft und trage sie ggf.
//als Begrenzungskreis ein;
(6) Repeat für jeden Zyklus{
  Wähle beliebige Stelle auf dem Kreis;
  If ((Stelle enthält einen kompletten DPA) und (aktuelle Fuzzy-Beliefs der Stelle
    haben sich nach einem Durchlauf durch den Kreis nicht verändert) und
    (DPA existiert bereits per viele Auswertungsschritte))
  Then {
    weise jeder Stelle auf dem Kreis den Status po zu;
    lösche kompletten DPA;
    If alle Stellen im Stellen-Vorbereich der Stellen auf dem Zyklus haben
      Status po oder ss
      Then trage aktuellen Zyklus in die Variable limitcycle ein;

    Else DominanceCount = 1; Speichere aktuelle Fuzzy-Beliefs der Stelle;
    Lösche kompletten DPA;
  }
  Fi;
}
}
Until
  (für alle Stellen im FPN: aktueller Fuzzy-Belief == vorheriger Fuzzy-Belief) OR
  (kein Zyklus im FPN hat Begrenzungskreis-Eigenschaft)

EndRepeat;

If (für alle Stellen im FPN: aktueller Fuzzy-Belief == vorheriger Fuzzy-Belief)
  Then print(„steady-state wurde erreicht“);
  Else print(„Begrenzungskreis existiert“);
}

```

Abb. 26: Der Auswertungsalgorithmus aus Kapitel 4.3.3 mit Verfeinerungen

Initial wird in Zeile 2 der DPA der Stellen im FRN mit Vorbereich auf den Wert „p“ gefolgt von der ID dieser Stellen gesetzt. Die Dominanzpfadausdrücke aller weiteren Stellen (K/M-Axiome) und der Transitionen bleiben leer.

In jeder Iteration der **Repeat**-Schleife berechnet der Algorithmus in den Schritten 3 und 4, wie

bereits in Kapitel 4.3.3 beschrieben, neue Fuzzy-Truth-Token und neue Fuzzy-Beliefs in den Transitionen bzw. den Stellen des auszuwertenden FRNs. Zusätzlich werden in diesen Schritten für die Knoten des FRNs neue DPAs erzeugt. In Schritt 3 registriert der Algorithmus bei der Berechnung der Fuzzy-Truth-Token in einer Transition das Vorzeichen der dominanten Kante, die in diese Transition einläuft und den neuen Wert des FTT bestimmt<sup>1</sup>. Dann kopiert er die Dominanzpfadausdrücke aus der Stelle, von der diese dominante Kante ausgeht, in die Transition hinein und verlängert jeden dieser Ausdrücke um ein Pfadelement, bestehend aus Kantenvorzeichen und Kennung der aktuellen Transition. Analog wird bei der Berechnung der DPAs für die Stellen in Schritt 4 vorgegangen. Da hier jedoch jeweils zwei Kanten, die in eine Stelle einlaufen, gleichzeitig dominant sein können, werden die Dominanzpfadausdrücke entlang dieser beiden Pfade aus dem Vorbereich einer Stelle kopiert und entsprechend verlängert. Zusätzlich erzeugt jede Stelle einen neuen DPA, der nur aus ihrer Kennung besteht.

Ein Beispiel für die Berechnung der Dominanzpfadausdrücke präsentiert Abbildung 27. Hier ist ein Kreis von drei Stellen auf einem zyklischen Pfad dargestellt. Vor der Auswertung (Runde 0) erhalten die Stellen jeweils als initialen DPA ihre Kennung ( $p1$ ,  $p2$  und  $p3$ ). Im ersten Auswertungsschritt (Runde 1) werden zuerst die Fuzzy-Truth-Token der Transitionen aktualisiert. Gleichzeitig verlängert der Algorithmus für jede Transition die Dominanzpfadausdrücke der Stellen aus ihrem Vorbereich entsprechend der Kantenvorzeichen. Somit ergibt sich in Transition  $t1$  der DPA  $p1+t1$ . Analog hat die Transition  $t2$  den DPA  $p2-t2$  und  $t3$  den DPA  $p3-t3$ . In derselben Runde werden auch in den Stellen neue Dominanzpfadausdrücke berechnet. In diese gehen die gerade neu berechneten Dominanzpfadausdrücke der Transitionen ein.

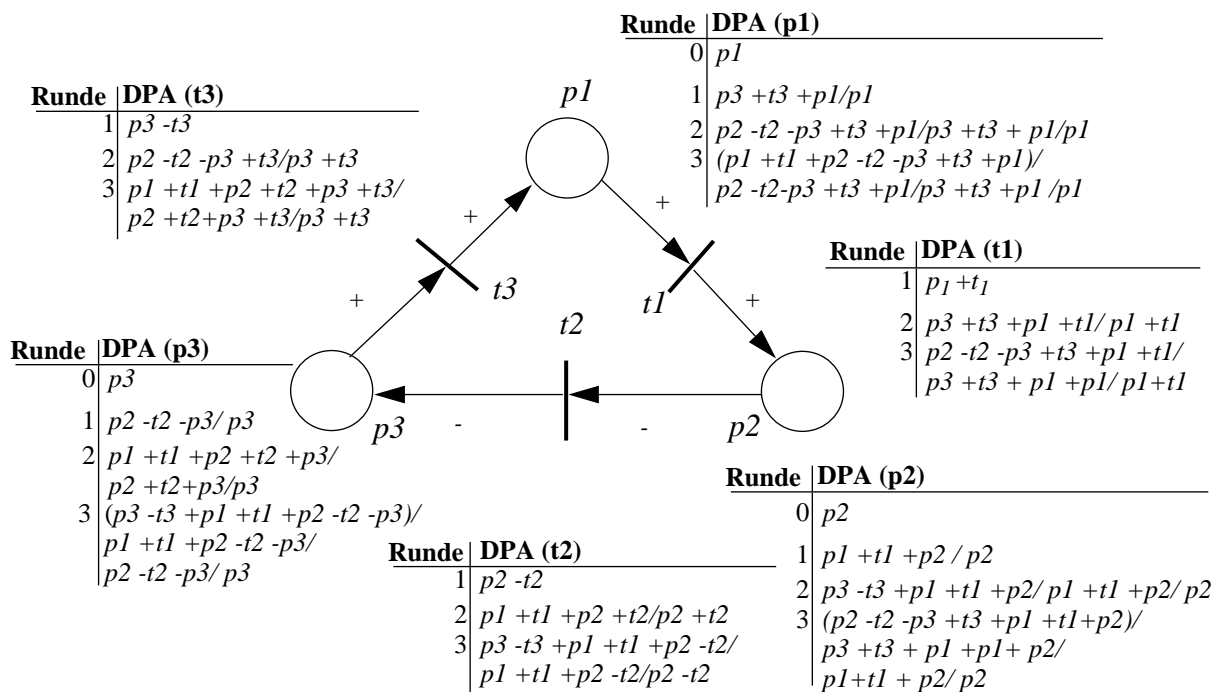


Abb. 27: Berechnung der Dominanzpfadausdrücke

1. vgl. Gleichung (6): der minimale AFB aller Stellen im Vorbereich einer Transition bestimmt deren FTT.

Für die Stelle  $p1$  verlängert der Algorithmus also die Dominanzpfadausdrücke der Transition  $t3$  aus Runde 1, usw. Zusätzlich erzeugt er für jede Stelle, die kein Axiom repräsentiert, in jeder Auswertungsrunde einen neuen initialen DPA. Nach drei Auswertungsrunden erkennt jede Stelle des Beispielnetzes einen *kompletten* DPA. In einem *kompletten* DPA entsprechen das erste und das letzte Element der Kennung der aktuellen Stelle. Dieser ist in Abbildung 27 jeweils mit runden Klammern versehen worden, da nach Erkennen eines DPAs, wie der Algorithmus zeigt, dieser lediglich durch Erhöhen eines Zählers registriert und anschließend gelöscht wird.

Ein Begrenzungskreis mit  $k$  Transitionen in einem FRN muß die folgenden Kriterien erfüllen<sup>1</sup>:

- (1) Der Kreis ist ein Dominanzkreis,
- (2) der Dominanzkreis existiert mindestens *per* Auswertungsschritte, dabei ist *per* abhängig von der Anzahl der Kontrapositionsstellen auf dem Kreis,
- (3) die Werte auf dem DK wiederholen sich periodisch alle *per* Schritte,
- (4) alle Stellen im Stellen-Vorbereich der Stellen auf dem DK haben Status *po* oder Status *ss*.

Die Begrenzungskreise in einem FRN werden mit Hilfe der in den Stellen berechneten Dominanzpfadausdrücke erkannt. Der Algorithmus überprüft in Schritt 4 zuerst den Status jeder Stelle im FRN. Haben alle Stellen im Stellen-Vorbereich der aktuellen Stelle den Status *ss*<sup>2</sup>, können sich die Fuzzy-Beliefs der aktuellen Stelle nicht mehr ändern und sie bekommt ebenfalls den Status *ss*. Konnte der Status *ss* jedoch nicht zugewiesen werden, testet der Algorithmus nun, ob die Stelle auf einem Dominanzkreis liegt und wieviele Auswertungsschritte dieser bereits existiert. Analog zu Theorem 1 muß ein Dominanzkreis mindestens die Anzahl von Auswertungsschritten bestehen bleiben, die der Periode des Kreises entspricht. Da sich die Periode eines Kreises leicht in Abhängigkeit der Kontrapositionsstellen auf dem Kreis berechnen läßt<sup>3</sup>, erhöht der Algorithmus bei Erkennen eines kompletten DPAs einen Zähler (*DominanceCount*), der die Anzahl an Auswertungsschritten zählt, die ein Dominanzkreis erhalten geblieben ist.

In Schritt 6 testet der Algorithmus jeden Zyklus auf die Begrenzungskreiseigenschaft. Dazu ist es ausreichend, zunächst eine Stelle auf jedem Kreis zu untersuchen. Enthält diese Stelle einen *kompletten* DPA<sup>4</sup>, liegt diese Stelle auf einem Dominanzkreis. Während der Auswertung des FRNs wurde also ein Kreis durchlaufen, der nur aus dominanten Kanten besteht. Nun prüft der Algorithmus, ob der DK bereits *per* Auswertungsschritte existiert und sich die Fuzzy-Beliefs der Stelle nach diesen *per* Auswertungsschritten wiederholen. In diesem Fall weist der DK periodisch oszillierende Werte auf, der Algorithmus ordnet jeder Stelle auf dem DK den Status *po* zu und löscht die kompletten DPAs in den zugehörigen Stellen. Haben als letzte zu testende Bedingung alle Stellen im Stellen-Vorbereich der Stellen auf dem DK analog zu Theorem 2 den Status *po* oder *ss*, erfüllt der Dominanzkreis die oben aufgeführten 4 Eigenschaften eines Begrenzungskreises, und er wird in die Variable *limitcycle*<sup>5</sup> eingetragen.

- 
1. vgl. Kapitel 4.3.3.
  2. *ss* steht für „steady-state“ und bedeutet, daß sich die Fuzzy-Beliefs einer Stelle mit diesem Status nicht mehr ändern können.
  3. vgl. Theorem 1.
  4. s. o.: in einem kompletten DPA entsprechen das erste und das letzte Element der Spezifikation der aktuellen Stelle.
  5. vgl. Abbildung 25.

Liegt eine Stelle hingegen auf mehreren Dominanzkreisen  $K_i$  mit unterschiedlich langer Periode  $per_i$ , muß analog zu Theorem 2 ein kompletter DPA in einer Stelle mindestens  $BRS_{min} = \text{Max}(kgV(per_1, \dots, per_y))$  viele Auswertungsschritte hintereinander erkannt werden. Zusätzlich müssen sich auch hier die Fuzzy-Belief-Werte in den Stellen periodisch mit der Periode  $BRS_{min}$  wiederholen, damit ein Dominanzkreis auch die Begrenzungskreis-Eigenschaft aufweist.

Wurden während eines Auswertungsschrittes ein oder mehrere Begrenzungskreise entdeckt und folglich in *limitcycle* gespeichert, ist damit das Abbruchkriterium für obigen Algorithmus erfüllt. Nun können die Maßnahmen zur Elimination der Begrenzungskreise analog zu Abschnitt 4.3.4 angewendet werden.

### 6.3 Das Laufzeitverhalten der Inferenzmaschine

Das Beispielnetz in Abbildung 35 enthält einen Begrenzungskreis. Die Abbruchbedingung des Algorithmus zur Erkennung von Begrenzungskreisen aus Abbildung 26 wurde für die Untersuchung des Laufzeitverhaltens der Inferenzmaschine geändert und durch die Abfrage einer Variable, die die Anzahl der Auswertungsschritte zählt, ersetzt. Läßt man das Beispielnetz mit diesem geänderten Algorithmus auf einem PC mit einem AMD K6-166 Prozessor 1000 Auswertungsschritte durchführen, benötigt dieser ca. 6 Sekunden. Der Algorithmus benötigt für die Berechnung der Fuzzy-Werte eines Knotens also durchschnittlich ca. 0,0006 sec.

Für ein Netz mit 200 Knoten, das beispielsweise 20 Schritte zum Erreichen des Gleichgewichtszustandes ausgeführt werden muß, benötigt der Algorithmus demnach voraussichtlich

$$200 \text{ Knoten} \cdot 0,0006 \frac{\text{sec}}{\text{Knoten}} \cdot 20 \text{ Schritte} = \underline{2,4 \text{ sec.}}$$

### 6.4 Die Entwicklungsumgebung

Zur Implementierung der in dieser Diplomarbeit vorgestellten Konzepte wurde die Entwicklungsumgebung *VisualAge for Java* von IBM in der Version 1.0 verwendet. *VisualAge for Java* ist eine integrierte Entwicklungsumgebung zur Erstellung von *Java*-Applikationen und *Java*-Applets mit Hilfe interaktiver, visueller Programmierwerkzeuge und *JavaBeans*, die Standardkomponenten von graphischen Benutzeroberflächen repräsentieren. Es werden alle Sprachgemeinschaften von *Java* 1.1 außer verschachtelte Klassen unterstützt.

Die Entwicklung selbst fand unter Windows 95 statt.



## 7 Anwendung der Inferenzmaschine auf das Eingangsszenario

Dieses Kapitel zeigt die Anwendung des Inferenzalgorithmus auf das Eingangsszenario. Dazu stellt Abbildung 28 zur einfacheren Bezugnahme noch einmal die GFRN-Spezifikation für das Szenario dar. Implikation  $i_9$  wurde zur Verkürzung des Beispiels jedoch weggelassen. Anschließend wird die schrittweise Expansion mit den dazwischenliegenden Auswertungsphasen vorgeführt.

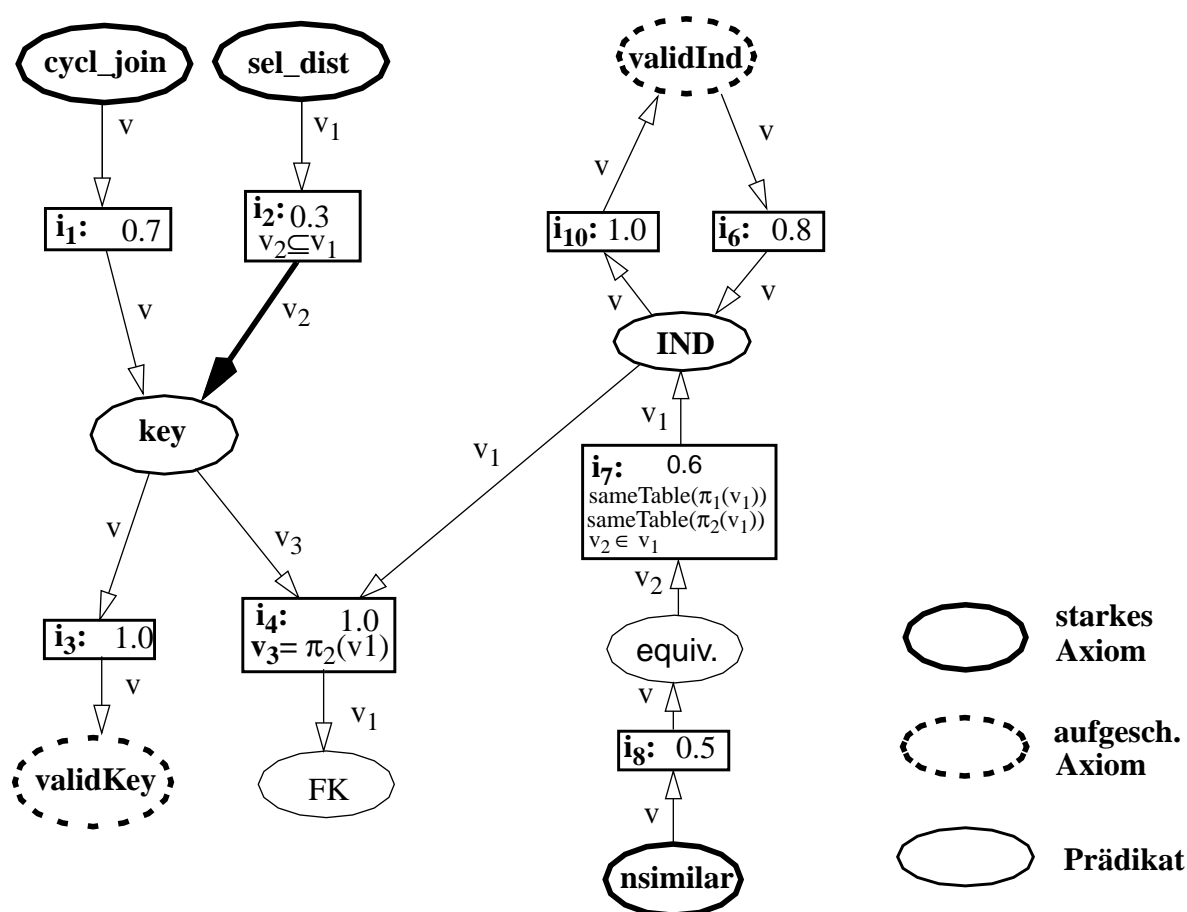


Abb. 28: GFRN-Spezifikation für das Eingangsszenario

Die Indizes an den Stellen in den nachfolgenden Abbildungen geben die jeweilige Phase an, in der eine Stelle erzeugt wurde.

### Phase 1:

Entsprechend dem Inferenzalgorithmus aus Kapitel 5.2 werden zuerst die Analyseoperationen ausgeführt, die an starke Axiome gebunden sind. Als starke Axiome sind das **cycl\_join**

Prädikat, das **sel\_dist** Prädikat und das **nsimilar** Prädikat gekennzeichnet.

Die *cycl\_join* Analyseoperation durchsucht den Applikationscode nach *cyclic\_join* Clichés. Der Ausschnitt des Applikationscodes aus Abbildung 1 enthält nur ein solches Cliché, nämlich die Anfrage

```
select * from Mieter
where (x.Haus = y.Haus)
and not (x.Name = y.Name).
```

Für diesen einen „Treffer“ erzeugt die Analyseoperation in dem zur GFRN-Spezifikation gehörenden FRN eine Stelle (**cycl\_join**(**Mieter.Name**), 1, 0), was bedeutet, daß die Analyseoperation ein *cyclic\_join* Cliché über das Attribut *Name* der Relation **Mieter** gefunden hat. Als positiven Fuzzy-Belief liefert die Analyseoperation den Wert 1, was die Tatsache unterstreicht, daß das Cliché tatsächlich im Applikationscode enthalten ist.

**cycl\_join**  
(**Mieter.Name**)

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} \mathbf{1}$$

Auch die Analyseoperation, die den Applikationscode nach *select-distinct* Clichés durchsucht, findet in dem Beispielausschnitt nur die Anfrage

```
select distinct Haus from Mieter
where Name = #N.
```

Hierfür wird eine Stelle (**sel\_dist**(**Mieter.Name**), 1, 0) im FRN erzeugt. Das Zustandekommen der Fuzzy-Beliefs ist auch hier wiederum mit der unumstößlichen Tatsache verbunden, daß das Cliché tatsächlich im Code gefunden werden konnte.

**sel\_dist**  
(**Mieter.Name**)

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} \mathbf{1}$$

Für das starke Axiom **nsimilar** setzt die Analyse nicht auf dem Applikationscode auf, sondern betrachtet das Schema der relationalen Datenbank. Hier sollen die Attributnamen in den unterschiedlichen Relationen auf ihre Ähnlichkeit überprüft werden. Ein Maß für die Ähnlichkeit zweier Wörter stellt beispielsweise die Levenshtein-Distanz [AWY90] dar, die die Kosten für das Ersetzen, Löschen oder Hinzufügen eines Buchstabens unterschiedlich gewichtet und minimiert.

Die Operation *nsimilar* bewertet die Ähnlichkeit der Attribute **Mieter.Name** und **Mieter.HMName** sowie der Attribute **Mieter.Haus** und **ApHaus.HausID** beispielsweise mit 0.8. Entsprechend werden im FRN eine Stelle (**nsimilar**(**Mieter.HMName**, **Mieter.Name**), 0.8, 0) und eine Stelle (**nsimilar**(**Mieter.Haus**,

**nsimilar**  
(**Mieter.HMName**,  
**Mieter.Name**)

$$\begin{pmatrix} 0.8 \\ 0 \end{pmatrix} \mathbf{1}$$

**nsimilar**  
(**Mieter.Haus**,  
**ApHaus.HausID**)

$$\begin{pmatrix} 0.8 \\ 0 \end{pmatrix} \mathbf{1}$$

**ApHaus.HausID**), 0.8, 0) erzeugt. Die Ähnlichkeit der Attribute besteht natürlich auch in die andere Richtung. Die Stellen (**nsimilar**(**Mieter.Name**, **Mieter.HMName**), 0.8, 0) und (**nsimilar**(**ApHaus.HausID**, **Mieter.Haus**), 0.8, 0) wurden jedoch zur Verkürzung des Beispiels und damit zur besseren Übersichtlichkeit der noch folgenden Abbildungen weggelassen.

Weitere starke Axiome sind in dem GFRN aus Abbildung 28 nicht enthalten. Folglich ist die GFRN-Spezifikation ausgehend von diesen vier Stellen weiter zu expandieren. Hierbei werden zunächst jedoch die aufgeschobenen Axiome vernachlässigt.

### Phase 2:

Der Inferenzalgorithmus legt für Implikation  $i_1$  ausgehend von der Stelle  $(\text{cycl\_join}(\mathbf{Mieter.Name}), 1, 0)$  eine Stelle  $(\mathbf{key}(\mathbf{Mieter.Name}), 0, 0)$  an, da für  $i_1$  alle Stellen im Vorbereich durch aktuelle Parameter vollkommen bestimmt sind. Die Stellen, die nicht als Ergebnis von Analyseoperationen erzeugt werden, erhalten als initiale Fuzzy-Beliefs jeweils Null-Werte. Konkrete Fuzzy-Beliefs ergeben sich später aus der Inferenz.

**key**  
**(Mieter.Name)**

$\begin{pmatrix} 0 \\ 0 \end{pmatrix}_2$

Für Implikation  $i_2$  kann das Prädikat **key** ebenfalls expandiert werden, da die Stelle  $(\text{sel\_dist}(\mathbf{Mieter.Name}), 1, 0)$  bereits in Phase 1 als Instanz des Prädikates **sel\_dist** für den Vorbereich von  $i_2$  erzeugt worden ist. Für die Expansion des Prädikates **key** ist zunächst die boole'sche Funktion<sup>1</sup>  $v_2 \subseteq v_1$  zu berücksichtigen. Diese fordert eine Erzeugung von Instanzen für das Prädikat **key** für alle Teilmengen  $v_2$  von  $v_1$ . Die Attributmeng  $v_1$  hat sich bei Ausführung der Analyseoperation *sel\_dist* zu **Mieter.Name** ergeben, enthält also nur ein Element. Deshalb kann auch nur für eine Teilmenge  $v_2$  von  $v_1$  eine Instanz des Prädikates **key** erzeugt werden, nämlich für die Menge  $v_2 = \{\mathbf{Mieter.Name}\}$ . Da die Stelle  $(\mathbf{key}(\mathbf{Mieter.Name}), 0, 0)$  bereits in dem FRN existiert, wird sie kein zweites Mal angelegt.

Auch für Implikation  $i_8$  kann der Algorithmus Stellen erzeugen, nämlich eine Stelle  $(\mathbf{equiv}(\mathbf{Mieter.HMName}, \mathbf{Mieter.Name}), 0, 0)$  auf Grundlage der Stelle  $(\mathbf{nsimilar}(\mathbf{Mieter.HMName}, \mathbf{Mieter.Name}), 0.8, 0)$  und eine Stelle  $(\mathbf{equiv}(\mathbf{Mieter.Haus}, \mathbf{ApHaus.HausID}), 0, 0)$  unter Berücksichtigung der Stelle  $(\mathbf{nsimilar}(\mathbf{Mieter.Haus}, \mathbf{ApHaus.HausID}), 0.8, 0)$ .

**equiv.**  
**(Mieter.HMName,**  
**Mieter.Name)**

$\begin{pmatrix} 0 \\ 0 \end{pmatrix}_2$

**equiv.**  
**(Mieter.Haus,**  
**ApHaus.HausID)**

$\begin{pmatrix} 0 \\ 0 \end{pmatrix}_2$

### Phase 3:

Ausgehend von den eben erzeugten Stellen  $(\mathbf{equiv}(\mathbf{Mieter.HMName}, \mathbf{Mieter.Name}), 0, 0)$  und  $(\mathbf{equiv}(\mathbf{Mieter.Haus}, \mathbf{ApHaus.HausID}), 0, 0)$  läßt sich das GFRN für Implikation  $i_7$  weiter expandieren. Dazu bildet der Algorithmus unter Berücksichtigung der in  $i_7$  spezifizierten boole'schen Funktion<sup>2</sup>

**IND**  
**(Mieter.HMName,**  
**Mieter.Name)**

$\begin{pmatrix} 0 \\ 0 \end{pmatrix}_3$

**IND**  
**(Mieter.Haus,**  
**ApHaus.HausID)**

$\begin{pmatrix} 0 \\ 0 \end{pmatrix}_3$

1. vgl. Definition 1.

2. vgl. Definition 1.

„ $v_2 \in v_1$ “ alle möglichen Variablenbindungen für die Variablen  $v_2$  und  $v_1$ . Anschließend berechnet er die Projektionen  $\pi_1$  und  $\pi_2$  auf den möglichen Belegungen für die Menge  $v_1$ . Die relationale Funktion<sup>1</sup>  $\pi_1$  führt eine Projektion auf das erste Element eines Elementepaares in  $v_1$  und  $\pi_2$  auf das zweite Element jedes dieser Paare aus. Nun sind noch die *sameTable*-Bedingungen für die Ergebnismengen der Projektionen auszuwerten. Die Bedingung *sameTable*( $x$ ) liefert für eine Menge von Attributbezeichnern  $x$  den Wert *true*, wenn sich alle Attribute aus  $x$  in derselben Tabelle des Datenbankschemas befinden. Ergeben die Auswertungen dieser Bedingung in beiden Fällen den Wert *true*, werden für die spezielle Variablenbelegung für  $v_1$  und  $v_2$  IND-Stellen erzeugt. Im Beispiel kann der Algorithmus vier verschiedene Variablenbindungen für  $v_1$  und  $v_2$  bestimmen, nämlich

- (1)  $v_1 = \{(HMName, Name), (Haus, HausID)\}$ ,  $v_2 = \{(HMName, Name)\}$ ,
- (2)  $v_1 = \{(HMName, Name), (Haus, HausID)\}$ ,  $v_2 = \{(Haus, HausID)\}$ ,
- (3)  $v_1 = v_2 = \{(HMName, Name)\}$  und
- (4)  $v_1 = v_2 = \{(Haus, HausID)\}$ .

Für die Belegungen (1) und (2) ergibt die *sameTable*-Bedingung nicht den Wert *true*. Die Projektion  $\pi_2$  auf das zweite Element jedes Paares in  $v_1$  enthält nämlich in beiden Fällen die Attribute  $\pi_2(\{(HMName, Name), (Haus, HausID)\}) = \{Name, HausID\}$ . Die Attribute *Name* und *HausID* befinden sich jedoch nicht in derselben Tabelle<sup>2</sup> und die *sameTable*-Bedingung ist damit nicht erfüllt. Die Projektionen auf das zweite Element in  $v_1$  für die Belegungen (3) und (4) enthalten jeweils nur ein Element, nämlich *Name* bzw. *HausID*. Für diese Projektionen ist die *sameTable*-Bedingung erfüllt, und der Algorithmus erzeugt die Stellen (**IND**(**Mieter.HMName**, **Mieter.Name**, 0, 0) und (**IND**(**Mieter.Haus**, **ApHaus.HausID**), 0, 0).

#### Phase 4:

In dieser Phase wird nun noch Implikation  $i_4$  expandiert. In dem Vorwärtsschritt der Expansion<sup>3</sup> betrachtet dieser die Stellen (**IND**(**Mieter.HMName**, **Mieter.Name**), 0, 0) und (**IND**(**Mieter.Haus**, **ApHaus.HausID**), 0, 0) als Instanzen des Prädikats **IND** und die Stelle (**key**(**Mieter.Name**), 0, 0) als Instanz des Prädikats **key**. Bevor der Algorithmus eine Stelle als Instanz des Prädikats **FK** anlegt, führt er die relationale Funktion<sup>4</sup>  $\pi_2(v_1)$  für die Attributmengen der Stellen (**IND**(**Mieter.HMName**, **Mieter.Name**), 0, 0) und (**IND**(**Mieter.Haus**, **ApHaus.HausID**), 0, 0) aus. Diese Funktion bestimmt als Ergebnismenge eine Projektion auf das jeweils zweite Element in jedem Attributpaar aus  $v_1$ . In diesem Beispiel besteht die Attributmenge  $v_1$  für die Stelle (**IND**(**Mieter.HMName**, **Mieter.Name**), 0, 0) nur aus dem Attributpaar (**Mieter.HMName**, **Mieter.Name**) und für die Stelle (**IND**(**Mieter.Haus**, **ApHaus.HausID**), 0, 0) aus dem Attributpaar (**Mieter.Haus**, **ApHaus.HausID**). Das Ergebnis  $v_3$  der Projektion auf das

**FK**  
(**Mieter.HMName**,  
**Mieter.Name**)

(0  
0)<sub>4</sub>

1. vgl. ebenfalls Definition 1.

2. vgl. Abbildung 1: das Attribut *Name* gehört zur Relation **Mieter**, das Attribut *HausID* hingegen zur Relation **ApHaus**.

3. vgl. Kapitel 5.2.

4. vgl. Definition 1.

jeweils zweite Element eines Attributpaares in  $v_1$  enthält für die Stelle (**IND**(**Mieter.HMName**, **Mieter.Name**), 0, 0) nur das Attribut (**Mieter.Name**) und für (**IND**(**Mieter.Haus**, **ApHaus.HausID**), 0, 0) das Attribut (**ApHaus.HausID**). Als nächstes wird nun die boole'sche Funktion<sup>1</sup>  $v_3 = \pi_2(v_1)$  aus Implikation  $i_4$  ausgewertet, die fordert, daß für alle Attribute in der Menge  $v_3$  eine Instanz des Prädikats **key** existiert. Diese Bedingung kann nur für die Menge  $v_3$  als Ergebnis der Projektion auf das zweite Element in der Attributmengung  $v_1 = \{(\mathbf{Mieter.HMName}, \mathbf{Mieter.Name})\}$  mit *true* bewertet werden, da keine Stelle (**key**(**Mieter.HausID**)) im FRN existiert. Deshalb wird das FRN nur um die Stelle (**FK**(**Mieter.HMName**, **Mieter.Name**), 0, 0) erweitert.

Abbildung 29 zeigt das aus den Phasen 1 bis 4 resultierende FRN. Die Namen der Attribute werden aus Gründen der Übersichtlichkeit abgekürzt. Der Buchstabe **n** steht dabei für das Attribut **Mieter.Name**, **hmn** für **Mieter.HMName**, **h** für **Mieter.Haus** und **hID** für **ApHaus.HausID**. Das FRN aus Abbildung 29 wird einer ersten Auswertung unterzogen, bis es nach drei Auswertungsrunden den Gleichgewichtszustand<sup>2</sup> annimmt. An dieser Stelle soll nur das Zustandekommen einiger Analyseergebnisse erläutert werden. Die weiteren Ergebnisse ergeben sich direkt aus der parallelen Anwendung der Minimum- und Maximumfunktion entsprechend der Gleichungen (7) bis (9) an den Stellen und Transitionen des FRNs. Die Zwischenergebnisse dieser Analyse im Gleichgewichtszustand zeigt Abbildung 30.

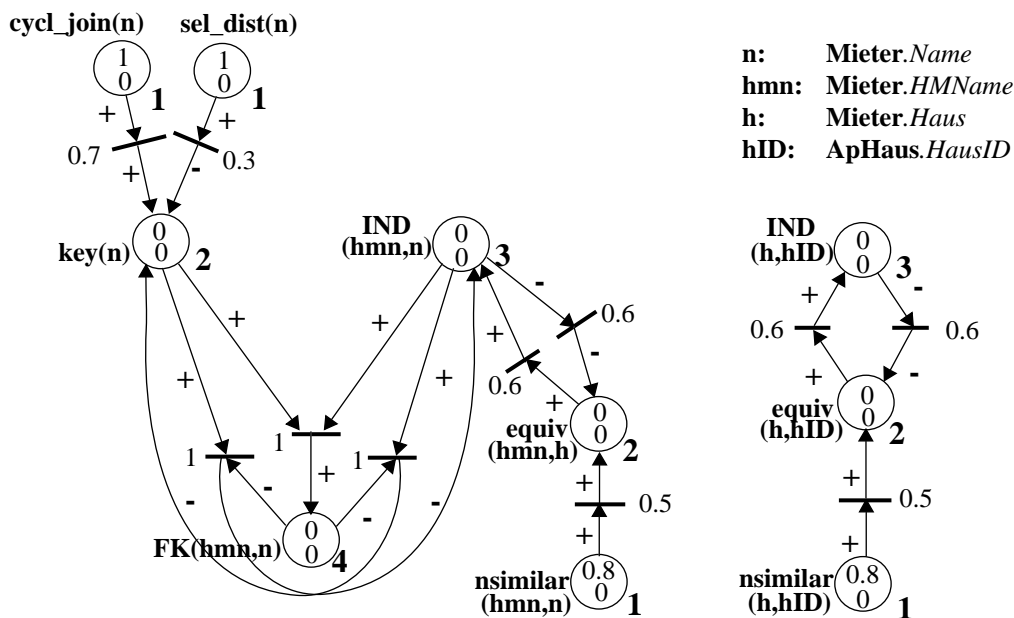


Abb. 29: FRN nach Phase 1 bis 4 mit initialen FB-Werten

Der positive Fuzzy-Belief der Stelle **key**(**n**) von 0.7 ergibt sich wie folgt: zuerst wird an der Transition im Nachbereich der Stelle **cycl\_join**(**n**) das Minimum der absoluten Fuzzy-Beliefs der in diese Transition eingehenden Stellen berechnet. Dieses Minimum ergibt sich zu 1. Das

1. vgl. Definition 1.

2. vgl. Kapitel 4.3: Das FRN befindet sich im Gleichgewichtszustand, wenn sich bei weiteren Auswertungsrunden keine Änderungen der Fuzzy-Belief-Werte in den Stellen mehr ergeben.

an der Transition gebildete Fuzzy-Truth-Token berechnet sich dann aus dem Minimum der Konfidenz der Transition von 0.7 und dem minimalen absoluten Fuzzy-Belief von 1 zu 0.7. Analog erhält die Stelle **key(n)** einen negativen Fuzzy-Belief von 0.3. Auch hier wird an der Transition zwischen **sel\_dist(n)** und **key(n)** das Minimum der absoluten Fuzzy-Beliefs der Stellen im Vorbereich der Transition bestimmt. Dieses ergibt sich zu 1. Nach Vergleich dieses Wertes mit der Konfidenz der Transition von 0.3 und Minimumbildung über diese beiden Werte, wird das Fuzzy-Truth-Token von 0.3 wegen der negativ beschrifteten Ausgangskante der Transition als negativer Fuzzy-Belief in die Stelle **key(n)** eingetragen.

Dem Algorithmus aus Abbildung 24 folgend werden nun die Analyseoperationen ausgeführt, die an die aufgeschobenen Axiome gebunden sind. Dazu sucht er in dem FRN nun die Instanzen von Prädikaten aus den GFRN-Implikationen, die aufgeschobene Axiome beinhalten. Nur wenn diese Instanzen einen absoluten Fuzzy-Belief größer als  $\epsilon^1$  haben, werden die Analyseoperationen der aufgeschobenen Axiome ausgeführt, die in den entsprechenden GFRN-Implikationen enthalten sind. Dies geschieht in dem dargestellten Beispiel für das Prädikat **validKey** in dem Rückwärts-Expansionsschritt, da sich dieses aufgeschobene Prädikat im Vorbereich der zugehörigen Implikation befindet. Das Prädikat **validIND** kann bereits im Vorwärts-Expansionsschritt berücksichtigt werden. Für die Expansion der aufgeschobenen Axiome untersucht der Algorithmus also die Fuzzy-Belief-Werte der Stelle (**key(n)**, 0.7, 0.3), der Stelle (**IND(hmn,n)**, 0.5, 0) und der Stelle (**IND(h,hID)**, 0.5, 0). Alle drei Stellen tragen einen absoluten Fuzzy-Belief größer Null. Deshalb werden für Implikation  $i_3$  die Analyseoperation *validKey(n)* und für Implikation  $i_{10}$  die Analyseoperation *validIND(hmn, n)* und *validIND(h, hID)* aufgerufen und die zugehörigen Stellen im FRN erzeugt.

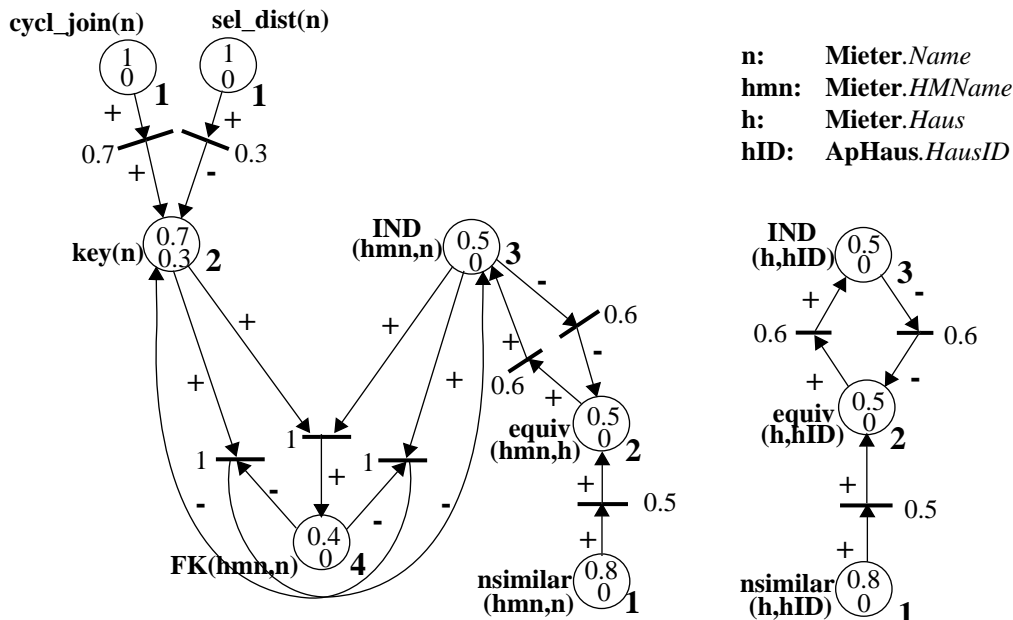


Abb. 30: Zwischenergebnisse der Analyse

1. vgl. Kapitel 5.2: Mit Hilfe der Zahl  $\epsilon$  ist es möglich, die Expansion der Prädikate im GFRN zu steuern.

Die Operation  $validKey(\mathbf{n})$  findet in den Daten der RDBA kein Gegenbeispiel gegen die Schlüsseleigenschaft des Attributs **Mieter.Name**. Deshalb bekommt die Stelle  $validKey(\mathbf{n})$  einen positiven Fuzzy-Belief von 1. Auch die Operation  $validIND$  kann für die Parameter  $(\mathbf{hmn}, \mathbf{h})$  und  $(\mathbf{h}, \mathbf{hID})$  keine Gegenbeispiele gegen eine Teilmengenbeziehung zwischen den genannten Attributen feststellen. Aus diesem Grund erhalten auch die Instanzen für das Prädikat  $validIND$  einen positiven Fuzzy-Belief von 1. Jetzt werden noch die Transitionen für die Implikation  $i_3$  und  $i_{10}$  erzeugt. Abbildung 31 zeigt das nun vollständig expandierte GFRN.

Die Auswertung dieses FRNs bis zum Gleichgewichtszustand ergibt keine zusätzlichen Änderungen der Fuzzy-Beliefs. Da keine weiteren Prädikate oder Implikationen mehr expandiert werden können, zeigt diese Abbildung auch das Endergebnis der Analyse. Dieses besagt, daß eine Schlüsseleigenschaft des Attributs **Mieter.Name** mit einer Notwendigkeit<sup>1</sup> von 0.7 vorliegt, da diese auch durch die Analyseoperation  $validKey(\mathbf{n})$  nicht zu widerlegen war. Eine negative Notwendigkeit von 0.3 gegen diese Aussage konnte nicht ausgeräumt werden, da ja ein *select-distinct* Cliché gegen diese Eigenschaft gesprochen hat. Eine äquivalente Bedeutung der Attribute **Mieter.Name** und **Mieter.HMName** sowie der Attribute **Mieter.Haus** und **ApHaus.HausID** liegt mit einer Notwendigkeit von 0.5 vor, und die Fremdschlüsselbeziehung zwischen den Attributen **Mieter.Name** und **Mieter.HMName** ist mit einer Notwendigkeit von 0.4 feststellbar.

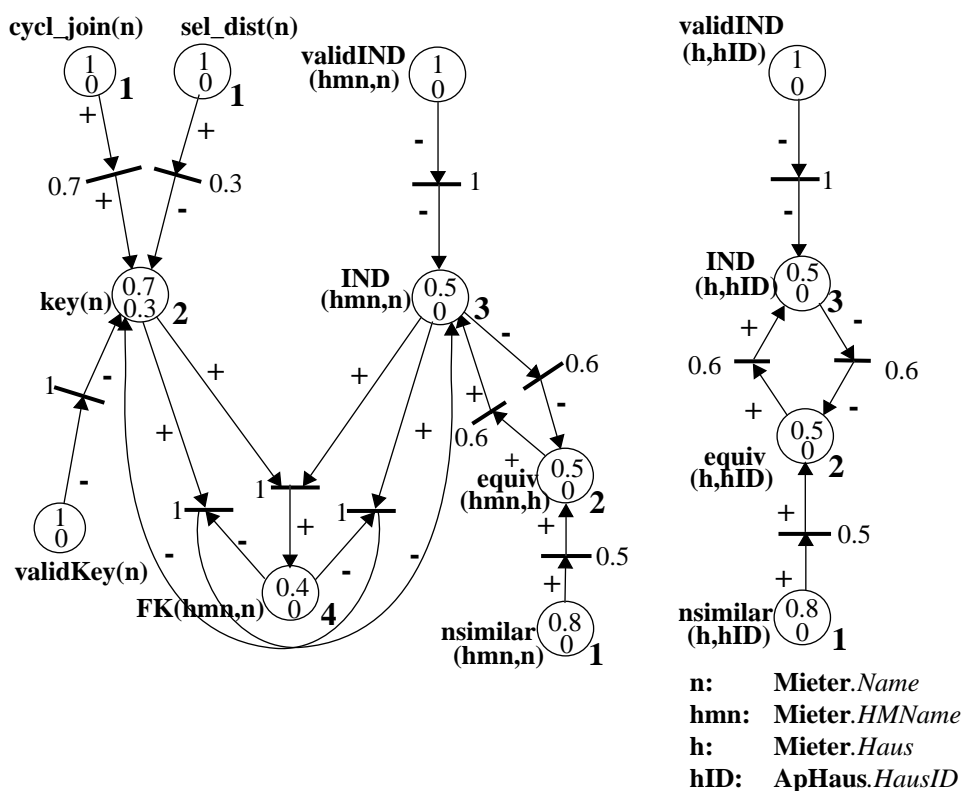


Abb. 31: Vollständig expandiertes GFRN

1. vgl. Kapitel 2.6: die Notwendigkeit einer Aussage gibt eine untere Schranke für deren Sicherheit an.



## 8 Zusammenfassung und Ausblick

### 8.1 Zusammenfassung

Diese Diplomarbeit stellt ein ausführbares Modell für *Generic Fuzzy Reasoning Nets (GFRNs)*, die sogenannten *Fuzzy Reasoning Nets (FRNs)* vor. Diese wurden auf Grundlage eines unscharfen Petrinetzmodells implementiert, das auf einem Ansatz von Konar und Mandal [KM96] basiert. Diese Ausführungsmaschine – auch *Inferenzmaschine* genannt – ist wegen ihrer Verwendung zum logischen Schließen unter Unsicherheit, dem damit verbundenen Schaltverhalten<sup>1</sup> und der verwendeten Logik für die Inferenz, der Possibilistischen Logik, effizient ausführbar.

Die FRNs stellen einen neuen Ansatz zum logischen Schließen unter Unsicherheit mit Hilfe unscharfer Petrinetze dar. Im Gegensatz zu dem in [KM96] vorgestellten unscharfen Petrinetzmodell unterstützen die FRNs einen nichtmonotonen Schlußfolgerungsprozeß. Dies resultiert aus der Möglichkeit der FRNs, Inkonsistenzen direkt verarbeiten zu können<sup>2</sup>, die aus der gleichzeitigen Existenz einer Aussage und deren Negation in demselben FRN entstehen. Außerdem werden die Fuzzy-Beliefs beider Aussagen, d.h. der Aussage an sich und ihrer Negation, bei der Ableitung einer Konsequenz berücksichtigt, was zu einer eindeutigen Interpretation der Analyseergebnisse führt. Die intuitive Semantik der Fuzzy-Beliefs, die einer Aussage im FRN zugeordnet werden, stellt ebenfalls eine Neuerung zu dem Ansatz von Konar und Mandal dar. Der Fuzzy-Wert einer Aussage wird als eine sogenannte *Notwendigkeit* interpretiert. Dieser Begriff stammt aus der Possibilistischen Logik und gibt eine untere Schranke für die Sicherheit einer Aussage an.

Die Possibilistische Logik<sup>3</sup> eignet sich besonders für die Logik der Ausführungsmaschine, da die Inferenz nur über die Minimum- und Maximumfunktion ausgeführt wird und sich die Berechnungen innerhalb der FRNs sehr einfach und performant ausführbar gestalten.

Für die Herleitung semantischer Informationen über eine RDBA wird ein FRN solange ausgewertet, bis sich die Notwendigkeiten der Aussagen nicht mehr ändern. Dieser Zustand des FRNs heißt auch *stabiler Zustand*. Leider ist dieser stabile Zustand nicht für alle FRNs erreichbar. Probleme bereiten zyklische FRNs, bei denen sich Werte in den Stellen auf einem Zyklus unendlich oft und periodisch wiederholen. Solche Zyklen, auch *Begrenzungskreise* genannt, müssen erkannt und eliminiert werden, bevor aussagekräftige Analyseergebnisse resultieren können. Für die Elimination der Begrenzungskreise ist eine Veränderung des zugehörigen FRNs notwendig. Jede Veränderung eines FRNs für eine GFRN-Spezifikation zieht aber auch eine Verfälschung des modellierten Wissens nach sich. Die Eliminierung der Begrenzungskreise verwendet ein Verfahren, das nur minimale Verfälschungen der Analyseergebnisse verursacht.

Die GFRNs ermöglichen eine graphische und abstrakte Spezifikation generischen Reengineering-Wissens über relationale Datenbanken. Sie werden im Rahmen der Reengineering-Umgebung *Varlet* zur Ableitung des konzeptionellen Schemas einer solchen Datenbank herangezogen. Das Wissen kann sehr unsicher sein und modelliert häufig nur Heuristiken. Im

---

1. vgl. Kapitel 4.3.2.

2. vgl. Kapitel 4.2.2.

3. vgl. Kapitel 2.6.

Gegensatz zu existierenden Ansätzen zur Datenbankanalyse [FV95, And94, PB94, PKBT94, JK90, SK90, DA87] wird das Reengineering-Wissen nicht fest in einem Analysewerkzeug kodiert, sondern bleibt wegen seiner expliziten Darstellung in Form eines GFRNs flexibel änderbar und anpaßbar an neue Rahmenbedingungen. Der Reengineer hat damit die Möglichkeit, durch Interaktion mit dem Analysewerkzeug die Datenbankanalyse effizient zu steuern, indem er neu hinzugekommene oder zum bisherigen Wissen abweichende Fakten frühzeitig in den Analysevorgang einbringt.

## 8.2 Ausblick

Die Anwendung der Inferenzmaschine wurde bisher nur anhand kleiner, konstruierter relationaler Datenbanken erprobt. Die Evaluierung der Performanz an realen RDBA steht demzufolge noch aus.

Für einen Einsatz des Analysewerkzeugs im Rahmen der Reengineering-Umgebung *Varlet* ist vorher der Expansionsalgorithmus formal zu spezifizieren und zu implementieren. Dieser könnte bei der Auswertung eines FRNs die hohe Parallelität des zugrundeliegenden unscharfen Petrinetzmodells unterstützen.

Für den Einsatz des Werkzeugs an realen RDBA wäre außerdem eine Möglichkeit zur persistenten Abspeicherung der FRNs sinnvoll. Mit einer optionalen Abspeicherung eines Netzes wäre ein Analyseprozeß unterbrechbar und zu einem späteren Zeitpunkt fortsetzbar.

Als weiterer Punkt ist zu klären, in welcher Form und zu welchen Zeitpunkten eine Präsentation der Zwischen- und Endergebnisse zu erfolgen hat. Gerade die Bereitstellung der Zwischenergebnisse der Analyse ist besonders wichtig, da sie u.U. eine Interaktion seitens des Reengineers initiieren. Durch einen möglichst frühen Eingriff des Reengineers ist die Herleitung semantischer Informationen über eine RDBA effizient steuerbar. Auch wäre es denkbar, Zwischenergebnisse der Analyse weiterzuverarbeiten und zum Training der zu interpretierenden GFRNs heranzuziehen. Ansätze zum Bereitstellen *lernfähiger* GFRNs sind in [CS98] nachzulesen.

Auch der Einsatz der Inferenzmaschine außerhalb einer Reengineering-Umgebung für relationale Datenbanken ist denkbar. Ein möglicher Anwendungsbereich, der ebenfalls einen nichtmonotonen Schlußfolgerungsprozeß mit einer unsicheren und unvollständigen Informationsmenge benötigt, ist das Erkennen von Design-Patterns [JZ97].

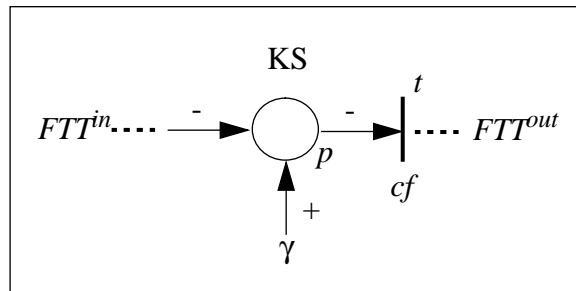


dem Pfad  $pf$  wegen der positiven Ausgangskante aus  $p$  in Analogie zu Gleichung (7) das Fuzzy-Truth-Token.

$$FTT^{out} = \text{Min}((FTT^{in} - \gamma), cf).$$

zu (b)

Wiederum erzeuge die Transition auf dem Pfad  $pf$  im Vorbereich von  $p$  zum Zeitpunkt  $t$  das FTT  $FTT^{in}$ . Diesmal ergeben sich der positive Fuzzy-Belief von  $p$  wegen der Kantenvorzeichen zu  $\gamma$  und der negative zu  $FTT^{in}$ . Zum Zeitpunkt  $t+1$  berechnet sich das FTT an Transition  $t$  wegen der negativen Ausgangskante aus  $p$  ebenfalls zu



$$FTT^{out} = \text{Min}((FTT^{in} - \gamma), cf).$$

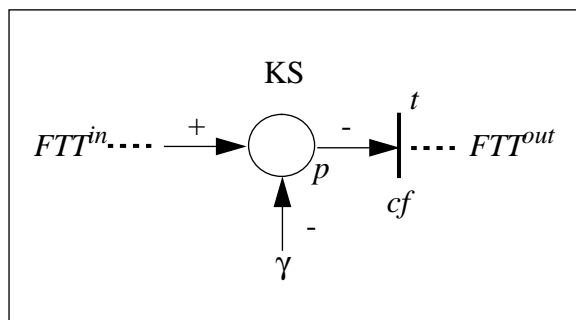
Teil 2:

Für eine Kontrapositionsstelle ergeben sich analog wiederum zwei Fälle:

- (a) Die entlang des Pfades in  $p$  einlaufende Kante ist positiv und die entlang des Pfades auslaufende Kante ist negativ.
- (b) Die entlang des Pfades in  $p$  einlaufende Kante ist negativ und die entlang des Pfades auslaufende Kante ist positiv.

zu (a)

Zum Zeitpunkt  $t$  erzeuge die auf dem Pfad liegende Transition im Vorbereich der Stelle  $p$  das Fuzzy-Truth-Token  $FTT^{in}$ . Der positive Fuzzy-Belief von  $p$  ergibt sich wegen des positiven Vorzeichens der Eingangskante entlang des Pfades zu  $FTT^{in}$ , der negative Fuzzy-Belief zu  $\gamma$ . Im Zeitschritt  $t+1$  berechnet sich wegen der negativen Ausgangskante aus  $p$  entlang des Pfades das Fuzzy-Truth-Token  $FTT^{out}$  zu



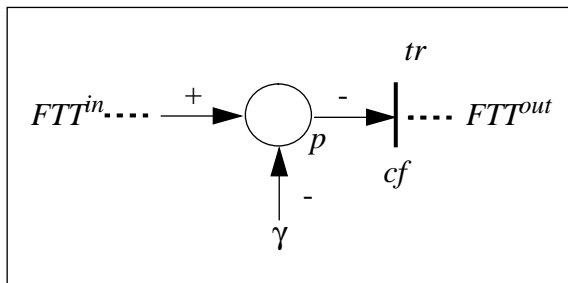
$$FTT^{out} = \text{Min}((\gamma - FTT^{in}), cf).$$

zu (b)

Hier ergibt sich die Markierung der Stelle  $p$  zu einem Zeitpunkt  $t$  zu  $(FTT^{in}, \gamma)$ . Das FTT an Transition  $tr$  berechnet sich anschließend wiederum zu:

$$FTT^{out} = \text{Min}((\gamma - FTT^{in}, cf)),$$

womit die Behauptung bewiesen ist.



**Lemma 2:**

Gegeben sei ein Pfad mit  $k$  Kontrapositionsstellen (KS). Für eine Eingabe von  $FTT_1^{in}$  und  $FTT_2^{in}$  mit  $FTT_1^{in} = FTT_2^{in} + \alpha, \alpha > 0$ , gilt:

$$FTT_1^{out} = FTT_2^{out} + \varepsilon; \varepsilon \in [0, \alpha], \text{ für } k \text{ gerade und}$$

$$FTT_1^{out} + \varepsilon = FTT_2^{out}; \varepsilon \in [0, \alpha], \text{ für } k \text{ ungerade.}$$

**Beweis:**

**Teil 1: Betrachte eine Nicht-Kontrapositionsstelle:**

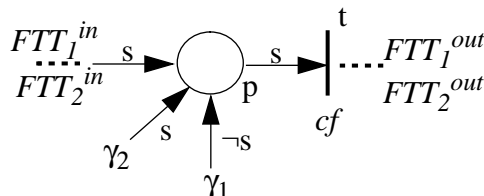


Abb. 33: Eine Nicht-KS

Für die Eingabe  $FTT_1^{in}$  und  $FTT_2^{in}$  in die Stelle  $p$  berechnen sich die Fuzzy-Truth-Token  $FTT_1^{out}$  und  $FTT_2^{out}$  an Transition  $t$  zu:

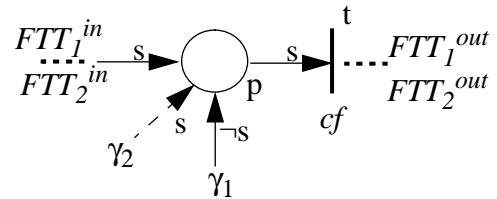
$$FTT_1^{out} = \begin{cases} \text{Min}(cf, FTT_1^{in} - \gamma_1), & \text{falls } FTT_1^{in} > \gamma_2 \\ \text{Min}(cf, \gamma_2 - \gamma_1), & \text{falls } FTT_1^{in} < \gamma_2 \end{cases}$$

$$FTT_2^{out} = \begin{cases} \text{Min}(cf, FTT_1^{in} - \alpha - \gamma_1), & \text{falls } FTT_2^{in} > \gamma_2 \\ \text{Min}(cf, \gamma_2 - \gamma_1), & \text{falls } FTT_2^{in} < \gamma_2 \end{cases}$$

Man betrachte nun die folgenden Fälle:

a)  $\neg FTT_1^{in} > \gamma_2 \Rightarrow FTT_1^{out} = \text{Min}(cf, FTT_1^{in} - \gamma_1)$

Dazu ergeben sich die folgenden Fälle:



1)

$\neg FTT_1^{in} - \gamma_1 > cf \Rightarrow FTT_1^{out} = FTT_1^{in} - \gamma_1$

1.1)

$\neg FTT_2^{in} > \gamma_2 \Rightarrow FTT_2^{out} = \text{Min}(cf, FTT_1^{in} - \alpha - \gamma_1) \mid \text{da } FTT_1^{in} = FTT_2^{in} + \alpha$

1.1.1)  $\neg FTT_2^{in} - \gamma_1 < cf \Rightarrow FTT_2^{out} = FTT_1^{in} - \alpha - \gamma_1$

1.2)  $\neg FTT_2^{in} < \gamma_2 \Rightarrow FTT_2^{out} = \text{Min}(cf, \gamma_2 - \gamma_1)$

1.2.1)  $\gamma_2 - \gamma_1 < cf \Rightarrow FTT_2^{out} = \gamma_2 - \gamma_1$

2)  $\neg FTT_1^{in} - \gamma_1 < cf \Rightarrow FTT_1^{out} = cf$

2.1)  $\neg FTT_2^{in} > \gamma_2 \Rightarrow FTT_2^{out} = \text{Min}(cf, FTT_1^{in} - \alpha - \gamma_1)$

2.1.1)  $\neg FTT_2^{in} - \gamma_1 < cf \Rightarrow FTT_2^{out} = FTT_1^{in} - \alpha - \gamma_1$

2.1.2)  $\neg FTT_2^{in} - \gamma_1 > cf \Rightarrow FTT_2^{out} = cf$

2.2)  $\neg FTT_2^{in} < \gamma_2 \Rightarrow FTT_2^{out} = \text{Min}(cf, \gamma_2 - \gamma_1)$

2.2.1)  $\neg FTT_2^{in} - \gamma_1 < cf \Rightarrow FTT_2^{out} = FTT_1^{in} - \alpha - \gamma_1$

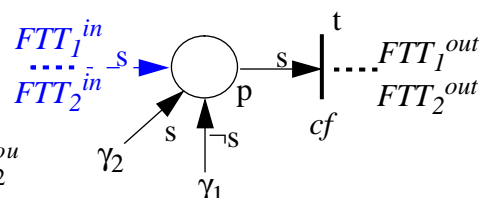
2.2.2)  $\neg FTT_2^{in} - \gamma_1 > cf \Rightarrow FTT_2^{out} = cf$

b)  $\neg FTT_1^{in} < \gamma_2 \Rightarrow FTT_1^{out} = \text{Min}(cf, \gamma_2 - \gamma_1)$

Daraus ergeben sich die Fälle

1)

$\gamma_2 - \gamma_1 < cf \Rightarrow FTT_1^{out} = \gamma_2 - \gamma_1 = FTT_2^{out}$



und

$$2)'_2 - \gamma_1 > cf \quad \Rightarrow FTT_1^{out} = cf = FTT_2^{ou}.$$

**zu a 1.1.1)**

$$\text{Es gilt: } \gamma_1 > \gamma_2 \text{ und } FTT_1^{in} - \gamma_1 < cf \quad \Rightarrow FTT_1^{out} = FTT_1^{in} - \gamma.$$

$$\text{Außerdem sei } \gamma_2 > \gamma_1 \text{ und } FTT_2^{in} - \gamma_1 < cf \Rightarrow FTT_2^{out} = FTT_1^{in} - \alpha - \gamma.$$

$$\text{Also ist } FTT_1^{out} = FTT_2^{out} + \alpha \quad \Rightarrow \varepsilon = \alpha.$$

**zu a 1.2.1)**

$$\text{Auch hier gilt: } \gamma_1 > \gamma_2 \text{ und } FTT_1^{in} - \gamma_1 < cf \quad \Rightarrow FTT_1^{out} = FTT_1^{in} - \gamma.$$

$$\text{Nun sei aber } \gamma_2 < \gamma_1 \text{ und } \gamma_2 - \gamma_1 < cf \Rightarrow FTT_2^{out} = \gamma_2 - \gamma.$$

$$\text{Da } FTT_1^{in} > \gamma_2, \text{ existiert ein } \beta > 0, \text{ so daß } FTT_1^{in} - \beta = \gamma_2.$$

$$\text{Daraus folgt, daß } FTT_2^{out} = \underbrace{FTT_1^{in} - \gamma_1 - \beta}_{FTT_1^{out}}$$

$$\text{Somit gilt: } FTT_1^{out} = FTT_2^{out} + \beta \Rightarrow \varepsilon = \beta.$$

**zu a 2.1.1)**

$$\text{Hier gilt: } \gamma_1 > \gamma_2 \text{ und } FTT_1^{in} - \gamma_1 > cf \quad \Rightarrow FTT_1^{out} = cf.$$

$$\text{Des weiteren ist } \gamma_2 > \gamma_1 \text{ und } FTT_2^{in} - \gamma_1 < cf \quad \Rightarrow FTT_2^{out} = FTT_2^{in} - \gamma.$$

$$\text{Da } FTT_2^{in} - \gamma_1 < cf, \text{ existiert ein } \beta > 0, \text{ so daß } FTT_2^{in} - \gamma_1 + \beta = cf.$$

$$\text{Somit läßt sich } FTT_1^{out} \text{ schreiben als } FTT_1^{out} = \underbrace{FTT_2^{in} - \gamma_1 + \beta}_{FTT_2^{out}} \text{ und } \varepsilon = \beta.$$

**zu a 2.1.2)**

$$\text{Auch hier gilt wieder } \gamma_1 > \gamma_2 \text{ und } FTT_1^{in} - \gamma_1 > cf \quad \Rightarrow FTT_1^{out} = cf.$$

$$\text{Außerdem gilt: } \gamma_2 > \gamma_1 \text{ und } FTT_2^{in} - \gamma_1 > cf \quad \Rightarrow FTT_2^{out} = cf.$$

$$\text{Also ist } FTT_1^{out} = FTT_2^{out} \text{ und somit } \varepsilon = 0.$$

**zu a 2.2.1)**

$$\text{Es sei wieder } \gamma_1 > \gamma_2 \text{ und } FTT_1^{in} - \gamma_1 > cf \quad \Rightarrow FTT_1^{out} = cf$$

$$\text{und } \gamma_2 < \gamma_1 \text{ und } \gamma_2 - \gamma_1 < cf \quad \Rightarrow FTT_2^{out} = \gamma_2 - \gamma.$$

$$\text{Da } \gamma_2 - \gamma_1 < cf \text{ existiert ein } \beta > 0, \text{ so daß } \gamma_2 - \gamma_1 + \beta = cf.$$

Somit läßt sich  $FTT_1^{out}$  auch schreiben als  $FTT_1^{out} = \underbrace{\gamma_2 - \gamma_1 + \beta}_{FTT_2^{out}}$

Also gilt damit:  $FTT_1^{out} = FTT_2^{out} + \beta$  und  $\varepsilon = \beta$ .

**zu a 2.2.2)**

Auch hier gilt wieder:  $FTT_1^{in} > \gamma_2$  und  $FTT_1^{in} - \gamma_1 > cf \Rightarrow FTT_1^{out} = cf$ .

Diesmal sei aber  $FTT_2^{in} < \gamma_2$  und  $\gamma_2 - \gamma_1 > cf \Rightarrow FTT_2^{out} = cf$ .

Damit gilt  $FTT_1^{out} = FTT_2^{out}$  und  $\varepsilon = 0$ .

**zu b)**

In beiden Fällen gilt:  $FTT_1^{out} = FTT_2^{out}$  und damit  $\varepsilon = 0$ .

**Teil 2: Man betrachte eine Kontrapositionsstelle:**

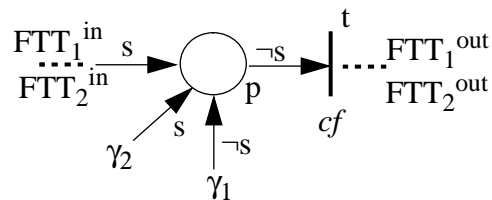


Abb. 34: Eine KS

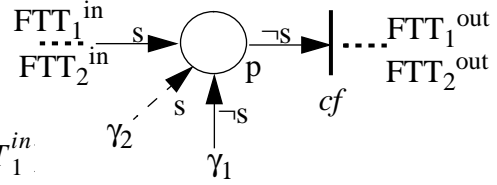
Wiederum gelte:  $FTT_1^{in} = FTT_2^{in} + \alpha$ ,  $\alpha \geq 0$ .

Für eine Kontrapositionsstelle berechnen sich  $FTT_1^{out}$  und  $FTT_2^{out}$  nach den folgenden Gleichungen:

$$FTT_1^{out} = \begin{cases} \text{Min}(cf, \gamma_1 - FTT_1^{in}), & \text{falls } FTT_1^{in} > \gamma_2 \\ \text{Min}(cf, \gamma_1 - \gamma_2), & \text{falls } FTT_1^{in} < \gamma_2 \end{cases}$$

$$FTT_2^{out} = \begin{cases} \text{Min}(cf, \gamma_1 - FTT_1^{in} + \alpha), & \text{falls } FTT_2^{in} > \gamma_2 \\ \text{Min}(cf, \gamma_1 - \gamma_2), & \text{falls } FTT_2^{in} < \gamma_2 \end{cases}$$

Hieraus ergeben sich nun die folgenden Fälle:



a)

$$FTT_1^{in} > \gamma_2 \Rightarrow FTT_1^{out} = \text{Min}(cf, \gamma_1 - FTT_1^{in})$$

1)

$$\gamma_1 - FTT_1^{in} < cf \Rightarrow FTT_1^{out} = \gamma_1 - FTT_1^{in}$$

$$1.1) FTT_2^{in} > \gamma_2 \Rightarrow FTT_2^{out} = \text{Min}(cf, \gamma_1 - FTT_2^{in})$$

$$1.1.1) \gamma_1 - FTT_2^{in} < cf \Rightarrow FTT_2^{out} = \gamma_1 - FTT_2^{in}$$

$$1.1.2) \gamma_1 - FTT_2^{in} > cf \Rightarrow FTT_2^{out} = cf$$

$$1.2) FTT_2^{in} < \gamma_2 \Rightarrow FTT_2^{out} = \text{Min}(cf, \gamma_1 - \gamma_2)$$

$$1.2.1) \gamma_1 - \gamma_2 < cf \Rightarrow FTT_2^{out} = \gamma_1 - \gamma_2$$

$$1.2.2) \gamma_1 - \gamma_2 > cf \Rightarrow FTT_2^{out} = cf$$

$$2) \gamma_1 - FTT_1^{in} > cf \Rightarrow FTT_1^{out} = cf$$

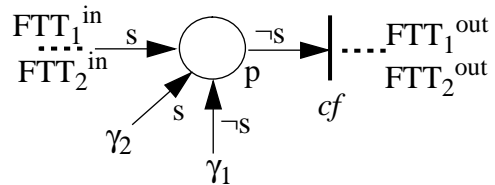
$$2.1) FTT_2^{in} > \gamma_2 \Rightarrow FTT_2^{out} = \text{Min}(cf, \gamma_1 - FTT_2^{in})$$

$$2.1.1) \gamma_1 - FTT_2^{in} > cf \Rightarrow FTT_2^{out} = cf$$

$$2.2) FTT_2^{in} < \gamma_2 \Rightarrow FTT_2^{out} = \text{Min}(cf, \gamma_1 - \gamma_2)$$

$$2.2.1) \gamma_1 - \gamma_2 > cf \Rightarrow FTT_2^{out} = cf$$

$$b) FTT_1^{in} < \gamma_2 \Rightarrow FTT_1^{out} = \text{Min}(cf, \gamma_1 - \gamma_2) \wedge FTT_2^{out} < \gamma_2$$



zu a 1.1.1)

Es gilt:

$$FTT_1^{in} > \gamma_2 \wedge (\gamma_1 - FTT_1^{in} < cf) \Rightarrow FTT_1^{out} = \gamma_1 - FTT_1^{in}$$

Weiterhin ist  $(FTT_2^{in} > \gamma_2) \wedge (\gamma_1 - FTT_2^{in} < cf)$

$$\Rightarrow FTT_2^{out} = \gamma_1 - FTT_2^{in} = \gamma_1 - FTT_1^{in} + \alpha$$

$$\Rightarrow FTT_1^{out} + \alpha = FTT_2^{out}, \alpha \geq 0$$

**zu a 1.1.2)**

Es gilt:  $(FTT_1^{in} > \gamma_2) \wedge (\gamma_1 - FTT_1^{in} < cf) \Rightarrow FTT_1^{out} = \gamma_1 - FTT_1^{in}$ .

Außerdem gilt:  $(FTT_2^{in} > \gamma_2) \wedge (\gamma_1 - FTT_2^{in} > cf) \Rightarrow FTT_2^{out} = cf$ .

Da  $(\gamma_1 - FTT_1^{in} < cf)$  existiert ein  $\beta > 0$ ,

so daß  $\underbrace{\gamma_1 - FTT_1^{in}}_{FTT_1^{out}} + \beta = cf$

$$\Rightarrow FTT_1^{out} + \beta = FTT_2^{out}$$

**zu a 1.2.1)**

Es gilt:  $(FTT_1^{in} > \gamma_2) \wedge (\gamma_1 - FTT_1^{in} < cf) \Rightarrow FTT_1^{out} = \gamma_1 - FTT_1^{in}$ .

Außerdem ist:  $(FTT_2^{in} < \gamma_2 \text{ und } \gamma_1 - \gamma_2 < cf) \Rightarrow FTT_2^{out} = \gamma_1 - \gamma_2$ .

Da  $(FTT_1^{in} > \gamma_2)$  existiert ein  $\beta > 0$ , so daß  $FTT_1^{in} - \beta = \gamma_2$ .

Also ist  $FTT_2^{out} = \gamma_1 - (FTT_1^{in} - \beta) = \gamma_1 - FTT_1^{in} + \beta$

$$\Rightarrow FTT_1^{out} + \beta = FTT_2^{out}.$$

**zu a 1.2.2)**

Wiederum gilt:  $(FTT_1^{in} > \gamma_2) \wedge (\gamma_1 - FTT_1^{in} < cf) \Rightarrow FTT_1^{out} = \gamma_1 - FTT_1^{in}$

und  $(FTT_2^{in} < \gamma_2 \text{ und } \gamma_1 - \gamma_2 > cf) \Rightarrow FTT_2^{out} = cf$ .

Da  $(\gamma_1 - FTT_1^{in} < cf)$  existiert ein  $\beta > 0$ , so daß

$$FTT_1^{out} + \beta = \gamma_1 - FTT_1^{in} + \beta = cf = FTT_2^{out}.$$

**zu 2.1.1)**

Hier ist jetzt  $(FTT_1^{in} > \gamma_2) \wedge (\gamma_1 - FTT_1^{in} > cf) \Rightarrow FTT_1^{out} = cf$ .

Weiterhin gilt:  $(FTT_2^{in} > \gamma_2 \text{ und } \gamma_1 - FTT_2^{in} > cf) \Rightarrow FTT_2^{out} = cf$

$$\Rightarrow FTT_1^{out} = cf = FTT_2^{out} \Rightarrow \varepsilon = 0$$

**zu 2.2.1)**

Auch hier gilt:  $(FTT_1^{in} > \gamma_2) \wedge (\gamma_1 - FTT_1^{in} > cf) \Rightarrow FTT_1^{out} = cf$ .

Außerdem ist:  $FTT_2^{in} < \gamma_2$  und  $\gamma_1 - FTT_2^{in} > cf \Rightarrow FTT_2^{out} = cf$   
 $\Rightarrow FTT_1^{out} = FTT_2^{out} \Rightarrow \varepsilon = 0$

**zu Teil 2b)**

Sei nun

$$FTT_1^{in} < \gamma_2 \Rightarrow (FTT_1^{out} = \text{Min}(cf, \gamma_1 - \gamma_2))$$

$$\wedge (FTT_2^{in} < \gamma_2 \Rightarrow (FTT_2^{out} = \text{Min}(cf, \gamma_1 - \gamma_2)))$$

**zu 2b.1)**

Sei  $\gamma_1 - \gamma_2 < cf \Rightarrow FTT_1^{out} = \gamma_1 - \gamma_2 = FTT_2^{out} \Rightarrow \varepsilon = 0$ .

**zu 2b.2)**

Sei  $\gamma_1 - \gamma_2 > cf \Rightarrow FTT_1^{out} = cf = FTT_2^{out} \Rightarrow \varepsilon = 0$ .

### Teil 3: Vollständige Induktion

Die Fälle 1 und 2 dieses Beweises zeigen bisher nur, wie sich zwei Fuzzy-Truth-Token  $FTT_1^{in}$  und  $FTT_2^{in}$  auf einem einelementigen Pfad beim Passieren einer Kontrapositionsstelle bzw. einer Nicht-Kontrapositionsstelle verhalten: falls nämlich  $FTT_1^{in} = FTT_2^{in} + \alpha$  mit  $\alpha \geq 0$  vor dem Erreichen der Stelle gilt, so gilt nach Passieren einer Kontrapositionsstelle, daß  $FTT_1^{out} + \beta = FTT_2^{in}$  mit  $\beta \geq 0$  und nach Passieren einer Nicht-Kontrapositionsstelle, daß  $FTT_1^{out} = FTT_2^{in} + \beta$  mit  $\beta \geq 0$ . Der dritte Teil des Beweises zeigt nun die eigentliche Behauptung von Lemma 2 per Induktion über die Anzahl  $k$  der Kontrapositionsstellen auf einem Pfad.

#### Induktionsanfang (IA):

Sei  $k = 0$ , d.h. der Pfad besteht aus beliebig vielen Nicht-KS.

$FTT_1^{in}$  und  $FTT_2^{in}$  seien die Eingabe für den Pfad und es gilt nach Voraussetzung, daß  $FTT_1^{in} = FTT_2^{in} + \alpha$  mit  $\alpha \geq 0$ . Nach Teil 1 des Beweises gilt nach jeder Stelle:  $FTT_1^{out} = FTT_2^{out} + \beta$  mit  $\beta \geq 0$ . Damit gilt die Behauptung für einen Pfad ohne KS.

Sei  $k = 1$ , d.h. der Pfad läßt sich in einen Teilpfad a, der bei der Startstelle des Pfades beginnt und direkt vor der KS endet, die KS als Teilpfad b und einen Teil-

pfad c, der direkt nach der KS beginnt und bis zur letzten Stelle des Pfades reicht, aufteilen.

$FTT_i^{d,k}$  mit  $i \in \{1,2\}$ ,  $d \in \{in, out\}$  und  $k \in \mathbb{N}$  bezeichne den Ausgabewert, der sich aus  $FTT_i^{in}$  nach Passieren der k-ten KS ergibt. Der Parameter  $d$  gibt an, ob das entsprechende  $FTT$  Eingabe oder Ausgabe des zugehörigen Pfades ist. Für Pfad (a) gilt nach Teil 1 dieses Beweises:

$$FTT_1^{in} > FTT_2^{in} \Rightarrow FTT_1^{out,0} > FTT_2^{out,0}.$$

Für Pfad (b) gilt nach Teil 2 dieses Beweises:

$$FTT_1^{in,0} > FTT_2^{in,0} \Rightarrow FTT_1^{out,1} < FTT_2^{out,1},$$

und für Pfad (c) gilt nach Teil 1:

$$FTT_1^{in,1} < FTT_2^{in,1} \Rightarrow FTT_1^{out,1} < FTT_2^{out,1}$$

Damit gilt die Behauptung für eine KS:

$$FTT_1^{in} > FTT_2^{in} \Rightarrow FTT_1^{out} < FTT_2^{out}.$$

### Induktionsbehauptung (IB):

Die Behauptung aus Lemma 2 gelte für  $z$  KS, d.h.

$$FTT_1^{in} > FTT_2^{in} \Rightarrow FTT_1^{out,z} > FTT_2^{out,z}, \text{ falls } z \text{ gerade}$$

$$FTT_1^{in} > FTT_2^{in} \Rightarrow FTT_1^{out,z} < FTT_2^{out,z}, \text{ falls } z \text{ ungerade.}$$

### Induktionsschritt: $z \rightarrow z + 1$ .

a) Sei  $z$  gerade. Nach IB gilt für einen Pfad mit  $z$  KS:

$$FTT_1^{in} > FTT_2^{in} \Rightarrow FTT_1^{out,z} > FTT_2^{out,z}.$$

Verlängere diesen Pfad um einen Teilpfad  $pf$  bestehend aus beliebig vielen Stellen und genau einer KS. Dieser Pfad  $pf$  läßt sich analog zu IA mit  $k = 1$  in drei Teilpfade (a), (b) und (c) aufteilen.

Setze als Eingabe  $FTT_1^{in,z}$  und  $FTT_2^{in,z}$  für Teilpfad (a) die Werte

$$FTT_1^{out,z} \text{ und } FTT_2^{out,z}.$$

Der Teilpfad (a) hat nach Teil 1 dieses Beweises die Ausgabe:

$$FTT_1^{out,z^*} \text{ und } FTT_2^{out,z^*} \text{ mit}$$

$$FTT_1^{out,z^*} > FTT_2^{out,z^*}, \text{ da } FTT_1^{in,z} > FTT_2^{in,z}.$$

Die Ausdrücke  $FTT_1^{out,z^*}$  und  $FTT_2^{out,z^*}$  haben ebenfalls die oben beschrie-

bene Semantik. Das hochgestellte Sternchen „\*“ wird zur Unterscheidung verwendet, wenn zwei Ausdrücke mit gleichem  $z$ -Wert existieren, da die numerischen Werte dieser beiden Ausdrücke unterschiedlich sein können, z.B. bei  $FTT_1^{out, z}$  und  $FTT_1^{out, z^*}$ . Entsprechendes gilt auch für die Ausdrücke  $FTT_1^{in, z^*}$  und  $FTT_2^{in, z^*}$ .

Setze als Eingabe  $FTT_1^{in, z^*}$  und  $FTT_2^{in, z^*}$  wiederum die Werte  $FTT_1^{out, z^*}$  und  $FTT_2^{out, z^*}$ . Nach Teil 2 gilt für Teilpfad (b):

$$\vartriangleleft FTT_1^{out, z+1} < FTT_2^{out, z+1}, \text{ da } FTT_1^{in, z^*} > FTT_2^{in, z^*}$$

und für Teilpfad (c) nach Teil 1 des Beweises:

$$\vartriangleleft FTT_1^{out, z+1^*} < FTT_2^{out, z+1^*}, \text{ da } FTT_1^{in, z+1} < FTT_2^{in, z+1}$$

Also gilt zusammenfassend:

$$\vartriangleleft FTT_1^{in} > FTT_2^{in} \Rightarrow FTT_1^{out} < FTT_2^{out}$$

Damit gilt die Behauptung für eine ungerade Anzahl von KS.

b) Sei  $z$  ungerade. Nach IB gilt für einen Pfad mit  $z$  KS:

$$\vartriangleleft FTT_1^{in} > FTT_2^{in} \Rightarrow FTT_1^{out, z} < FTT_2^{out, z}$$

Verlängere diesen Pfad um einen Teilpfad  $pf$  bestehend aus beliebig vielen Stellen und genau einer KS. Dieser Pfad  $pf$  läßt sich wiederum analog zu IA mit  $k = 1$  in drei Teilpfade (a), (b) und (c) aufteilen.

Setze als Eingabe  $FTT_1^{in, z}$  und  $FTT_2^{in, z}$  für Teilpfad (a) die Werte  $FTT_1^{out, z}$  und  $FTT_2^{out, z}$ .

Der Teilpfad (a) hat nach Teil 1 dieses Beweises die Ausgabe:

$$\vartriangleleft FTT_1^{out, z^*} \text{ und } FTT_2^{out, z^*} \text{ mit} \\ \vartriangleleft FTT_1^{out, z^*} < FTT_2^{out, z^*}, \text{ da } FTT_1^{in, z} < FTT_2^{in, z}$$

Setze als Eingabe  $FTT_1^{in, z^*}$  und  $FTT_2^{in, z^*}$  wiederum die Werte  $FTT_1^{out, z^*}$  und  $FTT_2^{out, z^*}$ . Nach Teil 2 gilt für Teilpfad (b):

$$\vartriangleleft FTT_1^{out, z+1} > FTT_2^{out, z+1}, \text{ da } FTT_1^{in, z^*} < FTT_2^{in, z^*}$$

und für Teilpfad (c) nach Teil 1 des Beweises:

$$\vartriangleleft FTT_1^{out, z+1^*} > FTT_2^{out, z+1^*}, \text{ da } FTT_1^{in, z+1} > FTT_2^{in, z+1}$$

Also gilt zusammenfassend:

$$FTT_1^{in} > FTT_2^{in} \Rightarrow FTT_1^{out} > FTT_2^{out}.$$

Damit gilt die Behauptung für eine gerade Anzahl von KS.

## Anhang B Klassendefinitionen

Dieser Anhang zeigt die Klassendefinitionen und Schnittstellenmethoden der Klassen, die für die Realisierung der Ausführungsmaschine implementiert worden sind. Die Klasse **Evaluator** muß zur Auswertung eines Fuzzy Reasoning Nets in ihrer main-Methode die Definition dieses Netzes und den Aufruf *evaluate()* enthalten. Nach der Beschreibung der Klasse **Class Evaluator** folgt ein Beispiel für den Inhalt einer solchen main-Methode.

### Class Evaluator

#### Hierarchy

```
java.lang.Object
    Evaluator
```

#### Definition

```
import java.util.Vector;
```

```
class Evaluator {
```

```
/**
```

```
    Diese Klasse enthält Methoden zur Auswertung eines unscharfen Petrinetzes bis zum Gleichgewichtszustand. Bei der Auswertung gefundene Begrenzungskreise werden ggf. eliminiert.
```

```
*/
```

```
}
```

#### Methods

Konstruktor:

```
    Evaluator()
```

main-Methode

```
    public void Evaluator.main(java.lang.String [])
```

```
    void Evaluator.evaluate(FPN fpn)    //Auswertung des übergebenen FPNs bis zum
                                        //steady-state
```

### Beispiel für eine main-Methode in der Klasse Evaluator

```
public static void main(String args[]){
```

```
    //Netz aus 5 Stellen und 5 Transitionen mit Begrenzungskreis
```

```
    /* Achtung: Steuerung der Ausgabe des Auswerters
```

```
    Die Ausgabe von Zwischenergebnissen der Analyse wird über die Parameter gesteuert, die der Methode main in dem Parameterstring args übergeben werden. Diese Parameterliste sollte für die Ausgabe der Zwischenergebnisse den String „true“ an erster Stelle enthalten, ansonsten „false“. Wird eine Ausgabe der Analyseergebnisse gewünscht, sollte die Parameterliste an zweiter Stelle den Wert „true“ enthalten, ansonsten „false“.
```

Ein Aufruf des Auswerters ohne Ausgabe der Zwischenergebnisse, aber mit Ausgabe der Analyseergebnisse hat somit die Form:

```

java Evaluator „false“ „true“

*/

FPN fpn = new FPN();           //Erzeugen eines leeren FPNs

Evaluator eval = new Evaluator(); //Erzeugen eines leeren Evaluators
Vector sortedList = new Vector(); //Wird zum Eliminieren von Begrenzungskreisen
                                //benötigt

//Anlegen der Stellen. Die erste Position muß eine von 0 an nummerierte eindeutige Zahl als
//ID für die Stelle sein. Der nachfolgende String steht für die Aussage, die die Stelle repräsen-
//tiert. Die Positionen 3 und 4 der Parameterliste enthalten den positiven und den negativen
//Fuzzy-Belief der Stelle.

fpn.createNewPlace(0,"A",0,0);
fpn.createNewPlace(1,"AI",50,0);
fpn.createNewPlace(2,"B",0,0);
fpn.createNewPlace(3,"BI",50,0);
fpn.createNewPlace(4,"C",0,0);

//Anlegen der Transitionen des FPNs. Die erste Position muß wieder eine eindeutige ID für die
//Transition enthalten, die von 0 an nummeriert sein muß. Position 2 enthält das initiale Fuzzy-
//Truth-Token, Position 3 enthält den initialen Schwellwert und Position 4 die Konfidenz der
//Transition.

fpn.createNewTransition(0,0, 0, 100);
fpn.createNewTransition(1,0, 0, 100);
fpn.createNewTransition(2,0, 0, 100);
fpn.createNewTransition(3,0, 0, 100);
fpn.createNewTransition(4,0, 0, 100);

//Anlegen der Kanten mit createNewEdge(int PlaceID,
                                int TransitionID,
                                char direction,
                                char sign)
//direction: +: Stelle ---> Transition, -: Transition ---> Stelle)
//Vorzeichen der Kante.

fpn.createNewEdge(0,1,'+','-');
fpn.createNewEdge(2,1,'-','-');
fpn.createNewEdge(2,2,'+', '+');
fpn.createNewEdge(4,2,'-', '+');
fpn.createNewEdge(4,4,'+', '+');
fpn.createNewEdge(0,4,'-', '+');

```

```

fpn.createNewEdge(1,0,'+', '+');
fpn.createNewEdge(0,0,'-', '-');
fpn.createNewEdge(3,3,'+', '+');
fpn.createNewEdge(2,3,'-', '+');

//Auswertung des übergebenen FPNs.
if ((args != null) && (args.length > 0)) {
    eval.evaluate(fpn, args[0]);
}
else {
    limit.evaluate(fpn, "false");
}
if ((args != null) && (args.length > 1)) {
    if ((args[1].equals(",true")) {
        fpn.showResults();
    }
}
}
}

```

Das Netz, das in dieser main-Methode definiert worden ist, hat das folgende Aussehen:

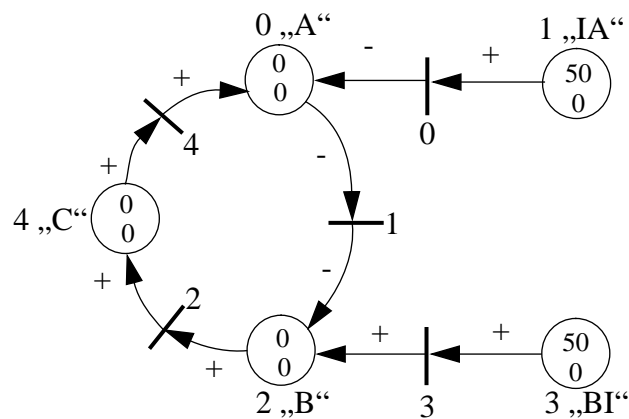


Abb. 35 Das FRN der main-Methode

## Class FPN

### Hierarchy

```

java.lang.Object
  FPN

```

**Definition**

```
import java.io.*;
import java.util.*;
```

```
/**
```

Diese Klasse definiert die Struktur und das Verhalten der unscharfen Petrinetze. Sie enthält Methoden zum Erzeugen, zur Manipulation und zur Auswertung dieser Netze. Die Stellen und Transitionen werden in indizierten Feldern vom Typ *indexedArray* unter dem Namen *places* bzw. *Transitions* gespeichert. Die Kanten des FPNs sind direkt in den Stellen und den Transitionen gespeichert.

```
*/
```

```
class FPN {
    public IndexedArray places;    //Stellen des FPNs
        IndexedArray transitions; //Transitionen des FPNs
        int steadyCount;          //Anzahl der Stellen mit Status „ss“
        int changingCount;       //Anzahl der Stellen mit Status „ch“
        int poCount;             //Anzahl der Stellen mit Status „po“
        Vector paths;            //Pfade im FPN
        Vector cycles;           //Zyklen im FPN
        Vector limitcycle;       //Zyklen mit Begrenzungskreis-Eigenschaft
    }
```

**Methods**

Konstruktor:

```
    FPN()
```

```
    void FPN.main(java.lang.String [])
```

get-Methoden:

```
    java.util.Vector FPN.getPaths()
    Place FPN.getPlaceByID(int)
    Place FPN.getPlaceByIndex(int)
    IndexedArray FPN.getPlaces()
    void FPN.getStatus()
    Transition FPN.getTransitionByID(int)
    Transition FPN.getTransitionByIndex(int)
    IndexedArray FPN.getTransitions()
```

set-Methoden

```
    void FPN.setCH(java.util.Vector)
    void FPN.setInitialDpe()
    void FPN.setInitialStatus()
    void FPN.setPaths(java.util.Vector)
    void FPN.setPO(java.util.Vector)
```

Sonstige public-Methoden:

```

void FPN.checkLC() //Testet alle Zyklen auf Begrenzungskreis-
//Eigenschaft

int FPN.computeLongestPathSize() //Berechnet die längste Pfadlänge im FPN
void FPN.computePaths() //Berechnet die Pfade im FPN
void FPN.createNewEdge(int place, int transition, char dir, char sig)
//Erzeugt eine neue Kante zwischen place und
//transition mit entsprechender Richtung dir und
//Kantenvorzeichen sig. Diese Kante wird in der
//spezifizierten Stelle place und in der angege-
//benen Transition transition eingetragen.

void FPN.createNewPlace(int pID, java.lang.String proposition, int posFBM, int negFBM)
//Erzeugt neue Stelle mit den in der Parameterliste
//angegebenen Parametern und trägt sie in places
//ein.

void FPN.createNewTransition(int, int, int, int)//Erzeugt neue Transition mit den in der
//Parameterliste angegebenen Parametern und
//trägt sie in transitions ein.

java.util.Vector FPN.evaluate() //Auswertung des FPNs bis zum steady-state, bzw.
//bis zum Erkennen eines Begrenzungskreises

void FPN.printCycles() //Ausgabe der Zyklen auf dem Bildschirm
void FPN.printLimitcycle() //Ausgabe der gefundenen Begrenzungskreise
void FPN.printPaths() //Ausgabe der gefundenen Pfade
void FPN.showResults() //Ausgabe der Fuzzy-Beliefs der Stellen und der
//Fuzzy-Truth-Token der Transitionen

java lang.String FPN.toString() //Schreibt FPN in einen String und liefert diesen
//zurück

```

## Class IndexedFpnObject

/\*\*

Diese Klasse definiert Objekte, die mit einer eindeutigen ID versehen sind. Diese Objekte werden in einem Array vom Typ *IndexedArray* gespeichert.

\*/

### Hierarchy

```

java.lang.Object
  IndexedFpnObject

```

### Definition

```

class IndexedFpnObject {
    public int objectID;
}

```

**Methods**

Konstruktoren:

```
IndexedFpnObject()
IndexedFpnObject(int)
```

main-Methode:

```
void IndexedFpnObject.main(java.lang.String [])
```

get-Methoden:

```
int IndexedFpnObject.getObjectID()
```

set-Methoden:

```
void IndexedFpnObject.setObjectID(int)
```

Sonstige public-Methoden:

```
void IndexedFpnObject.print()
java.lang.String IndexedFpnObject.toString()
```

**Class Place****Hierarchy**

```
java.lang.Object
IndexedFpnObject
Place
```

**Definition**

```
import java.util.Vector;
```

```
class Place extends IndexedFpnObject{
/**
```

```
    Diese Klasse definiert Struktur und Verhalten der Stellen-Knoten eines FPNs.
*/
```

```
private String proposition;           //Aussage der Stelle
    int posfbm;                       //positiver Fuzzy-Belief [1,100]
    int negfbm;                       //negativer Fuzzy-Belief [1,100]
    Vector InAntecedentOf;            //Liste der Eingangskanten in Transitionen, in
                                        //denen die aktuelle Stelle enthalten ist.
    Vector InConsequentOf;           //Liste der Ausgangskanten aus Transitionen, in
                                        //deren Nachbereich die aktuelle Stelle enthalten
                                        //ist
    String status;                   //Die Variable status kann drei Werte annehmen:
                                        //ss = steady-state, ch = changing, po = periodic
                                        //oscillation
```

```

    Vector dpe; //dpe = dominant path expression: wird zum
                //Erkennen von Begrenzungskreisen benötigt
    Vector dpeOld; //Wird in der aktuellen Stelle ein kompletter dpe
                //erkannt, wird dieser mit dem vorherigen
                //kompletten dpe verglichen und anschließend
                //selber in dpeOld gespeichert, wenn dpe und
                //dpeOld unterschiedlich waren
    int dominanceCount; //Zählt, wie viele Auswertungsschritte ein Zyklus
                //dominant geblieben ist.
    int domPfbm; //Speichert den positiven Fuzzy-Belief zu Beginn
                //jeder neuen Periode, in der der dpe dominant
                //bleibt. Länge der Periode ist abhängig von der
                //Anzahl der Kontrapositionsstellen auf dem
                //Dominanzkreis
    int domNfbm; //Speichert den negativen Fuzzy-Belief zu Beginn
                //jeder neuen Periode.
}

```

## Methods

Konstruktoren:

```

    Place()
    Place(int, java.lang.String, int, int, java.lang.String)

```

main-Methode:

```

    void Place.main(java.lang.String [])

```

get-Methoden:

```

    java.util.Vector Place.getCompleteDpe()
    int Place.getCPCount()
    int Place.getDominanceCount()
    int Place.getDomNFBM()
    int Place.getDomPFBM()
    java.util.Vector Place.getDpe()
    java.util.Vector Place.getDpeOld()
    Edge Place.getEdgeInConsequentOf(int)
    java.util.Vector Place.getInAntecedentOf()
    java.util.Vector Place.getInConsequentOf()
    int Place.getNegFBM()
    int Place.getPosFBM()
    java.lang.String Place.getProposition()
    java.lang.String Place.getStatus()

```

set-Methoden:

```

    void Place.setDominanceCount(int)
    void Place.setInAntecedentOf(Edge)
    void Place.setInConsequentOf(Edge)
    void Place.setInitialDpe(FPN)

```

```

void Place.setInitialStatus(FPN)
void Place.setNegFBM(int)
void Place.setPosFBM(int)
void Place.setProposition(java.lang.String)
void Place.setStatus(java.lang.String)

```

Sonstige public-Methoden:

```

boolean Place.checkAntecedentForPO(FPN)//Testet, ob alle Stellen im Vorbereich der
//aktuellen Stelle den Status po oder ss haben und
//liefert in diesem Falle true zurück
void Place.checkStatus(FPN) //Aktualisiert den Status der aktuellen Stelle
int Place.computeIncons() //Berechnet das Inkonsistenzmaß
void Place.evaluate(FPN,java.util.Vector) //Berechnet einen neuen positiven und negativen
//Fuzzy-Belief. Ändert sich mindestens einer
//dieser Werte, werden die IDs der Stellen in den
//Java-Vector changeArray aufgenommen.
void Place.print() //Ausgabe der Stelle auf dem Bildschirm
java.lang.String Place.toString() //Schreibt die Stelle in einen String und liefert
//diesen zurück.

```

## Class Transition

### Hierarchy

```

java.lang.Object
  IndexedFpnObject
    Transition

```

### Definition

```

import java.util.Vector;
/**

```

Diese Klasse definiert Struktur und Verhalten der Transition-Knoten der FPNe.

```

*/

```

```

class Transition extends IndexedFpnObject{
  public int ftt; //Fuzzy-Truth-Token der Transition
    int th; //Schwellwert
    int c; //Konfidenz
    Vector Antecedent; //Liste der Ausgangskanten aus Stellen, die im
//Vorbereich der aktuellen Transition sind.
    Vector Consequent; //Liste der Eingangskanten in Stellen, die im
//Nachbereich der aktuellen Transition sind.
    Vector dpe; //dpe = dominance path expression; wird zum
//Erkennen von Begrenzungskreisen benötigt.
}

```

**Methods**

Konstruktoren:

```
Transition()
Transition(int TransitionID, int FTT, int TH, int conf)
```

main-Methode:

```
void Transition.main(java.lang.String [])
```

get-Methoden:

```
java.util.Vector Transition.getAntecedent()
int Transition.getConfidence()
java.util.Vector Transition.getConsequent()
java.util.Vector Transition.getDpe()
Edge Transition.getEdgeInAntecedent(int)
int Transition.getFTT()
int Transition.getThreshold()
```

set-Methoden:

```
void Transition.setAntecedent(Edge)
void Transition.setConfidence(int)
void Transition.setConsequent(Edge)
void Transition.setFTT(int)
void Transition.setThreshold(int)
```

Sonstige public-Methoden:

```
void Transition.evaluate(FPN) //Berechnung eines neuen Fuzzy-Truth-Token
void Transition.print() //Ausgabe der Transition auf dem Bildschirm
java.lang.String Transition.toString() //Schreibt Transition in einen String und liefert
//diesen zurück.
```

**Class IndexedArray****Hierarchy**

```
java.lang.Object
IndexedArray
```

**Definition**

```
class IndexedArray {
    int blocksize = 5; //Verlängerung des Arrays jeweils um 5 leere
                       //Elemente
    public IndexedFpnObject array[]; //ein IndexedArray enthält nur Elemente vom Typ
                                     //IndexedFpnObject
    int arraysize; //Größe des Arrays
```

```

    int elemcount;                //Anzahl der Elemente
}

```

## Methods

Konstruktoren:

```

IndexedArray()
IndexedArray(int)

```

main-Methode

```

void IndexedArray.main(java.lang.String [])

```

get-Methoden:

```

int IndexedArray.getArraySize()
IndexedFpnObject IndexedArray.getElem(int)
int IndexedArray.getElemCount()
IndexedFpnObject IndexedArray.getObjectByID(int)

```

Sonstige public-Methoden:

```

void IndexedArray.insertObject(IndexedFpnObject)
                                //Einfügen eines IndexedFpnObjects in
                                //das Array
void IndexedArray.print()       //Ausgabe des IndexedArray auf dem Bildschirm
java.lang.String IndexedArray.toString() //Schreibt das IndexedArray in einen String und
                                //liefert diesen zurück

```

## Class ArrayObject

### Hierarchy

```

java.lang.Object
  ArrayObject

```

### Definition

```

class ArrayObject {
    //Superklasse der Klassen DPElem, Edge, Path und PathElem
}

```

### Methods

Konstruktor:

```

ArrayObject()

```

Sonstige public-Methoden:

```

java.lang.Object ArrayObject.clone()
void ArrayObject.print()

```

```
java.lang.String ArrayObject.toString()
```

## Class DPElem

### Hierarchy

```
java.lang.Object
  ArrayObject
    DPElem
```

### Definition

```
class DPElem extends ArrayObject {
/**
```

Diese Klasse definiert Elemente, die Stellen und Transitionen in den sogenannten *dominance path expressions* repräsentieren. Die Member-Variable *sign* enthält das Vorzeichen der Kante auf dem dpe, die das Vorgängerelement mit dem aktuellen verbindet. Die Variable *type* kann die Werte *p* für Place oder *t* für Transition tragen, *typeID* dient zur eindeutigen Identifikation der entsprechenden Stelle, bzw. der Transition.

```
*/
```

```
public char  sign;                //Vorzeichen der Kante zwischen vorherigem und
                                   //aktuellem Element
    String type;                  //Handelt es sich um eine Stelle (p) oder Transition
                                   //(t)
    int  typeID;                  //ID des IndexedFpnObject
}
```

### Methods

Konstruktor:

```
DPElem()
DPElem(char sig, java.lang.String type , int typeID)
```

main-Methode:

```
void DPElem.main(java.lang.String [])
```

get-Methoden:

```
char DPElem.getSign()
java.lang.String DPElem.getType()
int DPElem.getTypeID()
```

set-Methoden:

```
void DPElem.setSign(char)
void DPElem.setType(java.lang.String)
void DPElem.setTypeID(int)
```

Sonstige public-Methoden:

```

java.lang.Object DPElem.clone()           //Erzeugt identische Kopie des Objekts und liefert
                                           //diese zurück
boolean DPElem.equals(DPElem)           //Vergleich zweier DPElems. Liefert true zurück,
                                           //wenn beide Elemente gleich sind.
void DPElem.print()                     //Ausgabe des DPElem auf dem Bildschirm
java.lang.String DPElem.toString()      //Schreibt das DPElem in einen String und liefert
                                           //diesen zurück.

```

## Class Edge

### Hierarchy

```

java.lang.Object
  ArrayObject
    Edge

```

### Definition

```

class Edge extends ArrayObject{
/**

```

Diese Klasse definiert die Kanten-Objekte, die die Stellen und Transitionen im FPN verbinden. Kanten werden in JAVA-Vektoren in den zugehörigen Stellen und Transitionen gespeichert. Jede Kante ist damit zweimal abgelegt.

```

*/

```

```

public int place;           //ID der Stelle
    int transition;        //ID der Transition
    char direction;        //Richtung der Kante:
                           //+: Stelle ---> Transition
                           //-: Transition ---> Stelle
    char sign;             //Vorzeichen der Kante
}

```

### Methods

Konstruktoren:

```

Edge()
Edge(int placeID, int transitionID, char direction, char sign)

```

main-Methode:

```

void Edge.main(java.lang.String [])

```

get-Methoden:

```

char Edge.getDirection()
int Edge.getPlace()
char Edge.getSign()

```

```
int Edge.getTransition()
```

set-Methoden:

```
void Edge.setDirection(char)
void Edge.setPlace(int)
void Edge.setSign(char)
void setTransition(int)
```

Sonstige public-Methoden:

```
java.lang.Object Edge.clone()           //Erzeugt identische Kopie des Kanten-Objekts
boolean Edge.equals(Edge)              //Vergleich zweier Kanten-Objekte; liefert true
                                        //bei Gleichheit

void Edge.print()                       //Ausgabe des Kanten-Objekts auf dem
                                        //Bildschirm

void Edge.printEdgeInOutPlace()         //Formatierte Ausgabe der Kante
void Edge.printEdgeInOutTransition()    //Formatierte Ausgabe der Kante
java.lang.String Edge.toString()       //Schreibt das Kanten-Objekt in einen String und
                                        //liefert diesen zurück
```

## Class PathElem

### Hierarchy

```
java.lang.Object
  ArrayObject
    PathElem
```

### Definition

```
class PathElem extends ArrayObject {
```

```
/**
```

Diese Klasse definiert Elemente zur Speicherung azyklischer Schlußfolgerungspfade oder von Zyklen im FPN. Die Variable *type* kann die Werte *p* oder *t* für Stelle bzw. Transition annehmen. *typeID* dient der eindeutigen Identifizierung dieses IndexedFpnObjects in den Feldern FPN.places, bzw. FPN.transitions. Der *Fuzzy-Gain* wird bei Erkennen eines Begrenzungskreises benötigt, um die passende Transition für die Erhöhung des Schwellwertes zur Eliminierung dieses Begrenzungskreises auswählen zu können.

```
*/
```

```
public String type;           //p: Place, t: Transition
    int typeID;              //ID des IndexedFpnObject
    int posFuzzyGain;
    int negFuzzyGain;
    int allOverPosFG;
    int allOverNegFG;
    int allOverFG;
}
```

**Methods**

Konstruktoren:

```
PathElem()
PathElem(java.lang.String type, int typeID, int pfg, int nfg, int apfg, int anfg, int afg)
```

main-Methode:

```
void PathElem.main(java.lang.String [])
```

get-Methoden:

```
int PathElem.getAllOverFuzzyGain()
int PathElem.getAllOverNegFuzzyGain()
int PathElem.getAllOverPosFuzzyGain()
int PathElem.getNegFuzzyGain()
int PathElem.getPosFuzzyGain()
java.lang.String PathElem.getType()
int PathElem.getTypeID()
```

set-Methoden:

```
void PathElem.setAllOverFuzzyGain(int)
void PathElem.setAllOverNegFuzzyGain(int)
void PathElem.setAllOverPosFuzzyGain(int)
void PathElem.setNegFuzzyGain(int)
void PathElem.setPosFuzzyGain(int)
void PathElem.setType(java.lang.String)
void PathElem.setTypeID(int)
```

Sonstige public-Methoden:

```
java.lang.Object PathElem.clone() //Erzeugt identische Kopie des Pfadelementes
boolean PathElem.equals(PathElem) //Vergleich zweier Pfadelemente. Dabei werden
//die Fuzzy-Gains nicht berücksichtigt. Es wird
//true geliefert, wenn die type und typeID-Werte
//beider Pfadelemente übereinstimmen

void PathElem.print() //Ausgabe des Pfadelementes auf dem Bildschirm
java.lang.String PathElem.toString() //Schreibt das Pfadelement in einen String und
//liefert diesen zurück.
```

**Class Path****Hierarchy**

```
java.lang.Object
  ArrayObject
    Path
```

**Definition**

```
import java.util.Vector;
```

```
class Path extends ArrayObject{
```

```
/**
```

Diese Klasse dient der Speicherung einzelner azyklischer Schlußfolgerungspfade oder einzelner Zyklen im FPN. Jeder Pfad besteht aus einem *Java*-Vector, der Instanzen der Klasse *PathElem* enthält, und startet in einer Stelle, die keine Eingangskanten besitzt. Der Fuzzy-Gain, der in jedem Pfadelement gespeichert ist, gibt den Fuzzy-Gain von der Startstelle des Pfades bis zum aktuellen Pfadobjekt an. Falls das letzte Pfadelement eines kompletten Pfades zu mehreren parallelen Pfaden gehört, wird ein „allover“ Fuzzy-Gain aus dem Maximum der einzelnen Pfad-Gains berechnet.

```
*/
```

```
public Vector    array;                //JAVA-Vektor der Pfadelemente des Pfades
    boolean    containsCircle;        //true, falls der Pfad einen Zyklus enthält (wird
                                        //bei der Berechnung der Pfade zum Erkennen der
                                        //Zyklen benötigt)
    PathElem    doubleElem;          //bei zyklischen Pfaden
    boolean    extendable;           //false, falls der Pfad vollständig ist
    boolean    deleted;              //ist der Pfad Teil eines anderen, wird deleted auf
                                        //true gesetzt. Anschließend wird der Pfad
                                        //gelöscht.
}
```

**Methods**

Konstruktoren:

```
Path()
```

```
Path(java.util.Vector array, boolean cyclic, PathElem doubleElem, boolean extendable)
```

main-Methode:

```
void Path.main(java.lang.String [])
```

get-Methoden:

```
java.util.Vector Path.toArray()
```

```
boolean Path.getContainsCircle()
```

```
boolean Path.getDeleted()
```

```
PathElem Path.getDoubleElem()
```

```
boolean Path.getExtendable()
```

set-Methoden:

```
void Path.setArray(java.util.Vector)
```

```
void Path.setContainsCircle(boolean)
```

```
void Path.setDeleted(boolean)
```

```
void Path.setDoubleElem(PathElem)
```

```
void Path.setExtendable(boolean)
```

Sonstige public-Methoden:

```
java.lang.Object Path.clone() //Erzeugt identische Kopie des Pfad-Objekts
ArrayObject Path.containsCircle() //Prüft, ob der aktuelle Pfad zyklisch ist
int Path.containsElement(PathElem) //Prüft, ob das übergebene PathElem im Pfad
//enthalten ist und liefert die gefundene Position im
//Pfad.

Path Path.detectCycle() //Sucht den Zyklus in einem zyklischen Pfad und
//liefert diesen als Teilpfad zurück

boolean Path.equals(Path) //Vergleicht zwei Pfade und liefert bei Gleichheit
//true. Zwei Pfade sind gleich, wenn sie die
//gleichen Elemente unabhängig von ihrer Position
//enthalten

void Path.insertPathElem(java.lang.String, int, int, int, int, int, int)
//Erzeugt aus den in der Parameterliste
//angegebenen Werten ein Pfadelement und fügt
//dieses in den Pfad ein.

void Path.insertPathElem(PathElem) //Fügt ein Pfadelement in einen Pfad ein.
boolean Path.isSubPath(Path path) //Liefert true, falls der aktuelle Pfad ein Teilpfad
//von path ist.

void Path.print() //Ausgabe des Pfades auf dem Bildschirm
java.lang.String Path.toString() //Schreibt den Pfad in einen String und liefert
//diesen zurück
```

## Abbildungsverzeichnis

Abbildung 1) Ausschnitt aus einer relationalen Datenbankanwendung (RDBA) . . . . .	3
Abbildung 2) Konzeptionelles Schema in EER-Notation . . . . .	4
Abbildung 3) Datenbankanalyse in Varlet . . . . .	6
Abbildung 4) Einfache Regel in GFRN-Notation . . . . .	25
Abbildung 5) Ausschnitt einer GFRN-Spezifikation für das Eingangsbeispiel . . . . .	28
Abbildung 6) Beispiel einer GFRN-Spezifikation . . . . .	31
Abbildung 7) Ausschnitt einer GFRN-Spezifikation für das Eingangsbeispiel . . . . .	34
Abbildung 8) Ein S/T-Netz . . . . .	40
Abbildung 9) Schaltverhalten eines S/T-Netzes . . . . .	42
Abbildung 10) Transition mit $v$ Eingangsstellen . . . . .	44
Abbildung 11) Ausgangsstelle von $u$ Transitionen . . . . .	45
Abbildung 12) Ausschnitt aus einer GFRN-Spezifikation . . . . .	46
Abbildung 13) Ausschnitt aus einem FPN nach Konar und Mandal . . . . .	47
Abbildung 14) Berechnung des FTT . . . . .	51
Abbildung 15) Berechnung der Fuzzy-Beliefs . . . . .	52
Abbildung 16) Dominante und nicht dominante Kanten . . . . .	55
Abbildung 17) (a) Eine Nicht-KS und (b) eine KS . . . . .	57
Abbildung 18) Eine Nicht-KS . . . . .	58
Abbildung 19) Eine KS . . . . .	59
Abbildung 20) Berechnung der Fuzzy-Gains eines Pfades . . . . .	65
Abbildung 21) Berechnung des neuen Schwellwertes einer Transition . . . . .	66
Abbildung 22) Induktion von Begrenzungskreis-Verhalten in Nachbarzyklen . . . . .	67
Abbildung 23) Expansion einer einfachen GFRN-Regel mit 3 Prädikaten . . . . .	71

Abbildung 24) Der Inferenzalgorithmus . . . . .	72
Abbildung 25) Grobdesign der Ausführungsmaschine . . . . .	75
Abbildung 26) Der Auswertungsalgorithmus aus Kapitel 4.3.3 mit Verfeinerungen . .	78
Abbildung 27) Berechnung der Dominanzpfadausdrücke . . . . .	79
Abbildung 28) GFRN-Spezifikation für das Eingangsszenario . . . . .	83
Abbildung 29) FRN nach Phase 1 bis 4 mit initialen FB-Werten . . . . .	87
Abbildung 30) Zwischenergebnisse der Analyse . . . . .	88
Abbildung 31) Vollständig expandiertes GFRN . . . . .	89
Abbildung 32) (a) Eine Nicht-KS und (b) eine KS . . . . .	93
Abbildung 33) Eine Nicht-KS . . . . .	95
Abbildung 34) Eine KS . . . . .	98
Abbildung 35) Das FRN der main-Methode . . . . .	107

## Literatur

- [And94] M. Anderson. Extracting an Entity Relationship Schema from a Relational Database through Reverse Engineering. In *Proc. of the 13th Int. Conference of the Entity Relationship Approach*, Manchester, pages 403-419. Springer, 1994.
- [Arn93] R. S. Arnold. *Software Reengineering*. IEEE Computer Society PR., Los Alamitos, Calif., 1993.
- [AWY90] L. Allison, C.S. Wallace, C.N. Yee. *When is a String like a String?* In *AI & Maths*. 1990.
- [BB94a] A. Bugarin, S. Barro. *Goal-Driven Reasoning for Fuzzy Knowledge-Based Systems using a Petri Net Formalism*, IEEE, 1994.
- [BB94b] A. J. Bugarin, S. Barro. Fuzzy Reasoning Supported by Petri Nets. In: P. Smets, E. H. Mamdani, D. Dubois, H. Prade (editors) *IEEE Transactions on Fuzzy Systems*, Vol. 2, No. 2, pages 135-159, Academic Press, London, May 1994.
- [BB98] Barbara Bewermeyer. *Cliché-Erkennung in relationalen Datenbankanwendungen*. Diplomarbeit, Universität GH Paderborn, Fachbereich Mathematik/Informatik, 1998. (Fertigstellung im Herbst 1998)
- [BCN92] C. Batini, S. Ceri, S.B. Navathe. *Conceptual Database Design - An Entity-Relationship Approach*. Benjamin Cummings, Redwood City, CA. 1992.
- [CS98] Christoph Strebin. *Adaption unsicheren Reverse Engineering-Wissens auf Basis konnektionistischer Methoden*. Diplomarbeit, Universität GH Paderborn, Fachbereich Mathematik/Informatik, 1998. (Fertigstellung im Herbst 1998)
- [DA87] K. H. Davis, A. K. Arora. Converting a Relational Database Model into an Entity-Relationship-Model. In *Proc. of the 6th Int. Conference of the Entity Relationship Approach*, New York, pages 271- 285. North-Holland, November 1987.
- [DLP94] D. Dubois, J. Lang, H. Prade. Possibilistic Logic. In Dov M. Gabbay, C. J. Hogger, J. A. Robinson, D. Nute. (editors) *Handbook of Logic in AI and Logic Programming*, Clarendon Press, Oxford, 1994.
- [DP88] D. Dubois, H. Prade. An Introduction to Possibilistic and Fuzzy Logics. In P. Smets, E. H. Mamdani, D. Dubois (editors) *Non-Standard Logics for Automated Reasoning*. Academic Press, London, 1988.
- [DP88b] W. Dörfler, W. Peschek. *Einführung in die Mathematik für Informatiker*. Hanser-Verlag. 1988.
- [DRSW86] R. D. Dowsing, V.J. Rayward-Smith, C. D. Walter. *A First Course in Formal Logic and its Applications in Computer Science*. Blackwell Scientific Publications, 1986.
- [FV95] C. Fahrner, G. Vossen. Transforming Relational Database Schemas into Object-Oriented Schemas according to ODMG-93. In *Proc. of the 4th Int. Conf. on Deductive and Object-Oriented Databases 1995*, 1995.

- [FP92] H. Farreny, H. Prade. Positive and negative explanations of uncertain reasoning in the framework of possibility theory. In L. A. Zadeh, J. Kacprzyk (editors), *Fuzzy-Logic for the Management of Uncertainty*, John Wiley & Sons, Inc. 1992.
- [Gen87] H. J. Genrich. Predicate/Transition Nets. In K. Jensen, G. Rozenberg (editors), *High-Level Petri Nets*. SpringerVerlag, 1987.
- [Gin94] M. L. Ginsberg. AI and Nonmonotonic Reasoning. In Dov M. Gabbay/C. J. Hogger/J. A. Robinson/D. Nute. (editors), *Handbook of Logic in AI and Logic Programming*. Clarendon Press, Oxford, 1994.
- [Gr95] A. Grauel. *Fuzzy-Logik: Einführung in die Grundlagen mit Anwendungen*. BI-Wiss.-Verlag, 1995.
- [Haj94] P. Hajek. On Logics of Approximate Reasoning. In: *Lecture Notes in Computer Science* (808), pages 17-29, 1994.
- [JK90] P. Johannesson, K. Kalman. A Method for translating relational schemas into conceptual schemas. In F. H. Lochovsky, (editors), *Entity-Relationship Approach to Database Design and Querying*. ERI, 1990.
- [JH98] J. H. Jahnke, M. Heitbreder. Design Recovery of Legacy Database Applications based on Possibilistic Reasoning. In *Proceedings of 7th IEEE Int. Conf. of Fuzzy-Systems (FUZZ'98)*, May 1998, Anchorage, US.
- [JSZ97] J. Jahnke, W. Schäfer, A. Zündorf. Generic fuzzy reasoning nets as a basis for reverse engineering relational database applications. In *Proceedings of European Software Engineering Conference (ESEC/FSE)*, September 1997, Zürich. Springer (LNCS 1301).
- [JZ97] J. Jahnke, A. Zündorf. Rewriting poor Design Patterns by good Design Patterns. In S. Demeyer, H. Gall (editors) *Proceedings ESEC/FSE'97 Workshop on Object-Oriented Reengineering*. Technical Report TUV-1841-97-10, Technical University of Vienna, Information System Institute, Argentinierstraße 8/184-1, A-1040 Wien, Austria.
- [KL97] H. Kleine-Büning, T. Lettmann. *Skriptum zur Vorlesung Wissensbasierte Systeme*. Universität-GH Paderborn, Fachbereich Mathematik/Informatik, 1997.
- [KE96] A. Kemper, A. Eickler. *Datenbanksysteme: eine Einführung*. Oldenbourg Verlag, München, 1996.
- [KM96] A. Konar, A. K. Mandal. Uncertainty Management in Expert Systems Using Fuzzy Petri Nets. In *IEEE Transactions on Knowledge and Data Engineering*, Vol. 8, No. 1, Februar 1996.
- [Loo88] C.G. Looney. Fuzzy Petri Nets for Rule-Based Decisionmaking. In *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 18, No. 1, Januar/Februar 1988.

- [Nute94] D. Nute. Defeasible Logic. In Dov M. Gabbay, C.J. Hogger, J. A. Robinson, D. Nute (editors) *Handbook of Logic in AI and Logic Programming*. Clarendon Press, Oxford, 1994.
- [Paa88] G. Paass. Probabilistic Logic. In P. Smets, E. H. Mamdani, D. Dubois (editors) *Non-Standard Logics for Automated Reasoning*. Academic Press, London, 1988.
- [PAA94] L. M. Pereira, J. N. Aparicio, J. J. Alferes. Logic Programming for Non-Monotonic Reasoning. In: *Lecture Notes in Computer Science*, ISSN 0302 - 9743, 1994, S. 107-121.
- [PB94] W. L. Premerlani, M. R. Blaha. An approach for reverse engineering of relational databases. In *Communications of the ACM*, 37 (5): 42-49, Mai 1994.
- [PKBT94] J.-M. Petit, J. Kouloumdjian, J.-F. Boulicout, F. Toumani. Using queries to improve database reverse engineering. In *Proc. of the 13th Int. Conference of ERA*, Manchester, pages 369-386, Springer, 1994.
- [Poo94] D. Poole. Default Logic. In Dov M. Gabbay, C. J. Hogger, J. A. Robinson, D. Nute (editors), *Handbook of Logic in AI and Logic Programming*, Clarendon Press, Oxford, 1994.
- [Rei82] W. Reisig. *Petrinetze - Eine Einführung*. Springer, Berlin, Heidelberg, New York 1982.
- [Rol89] David W. Rolston. *Principles of artificial intelligence and expert system development - 3* [Dr.], New York [u.a.]. McGraw - Hill, 1989, Kapitel 6: Dealing with Uncertainty.
- [Schö92] U. Schöning. *Logik für Informatiker*. BI-Wiss.-Verlag, 1992 (Reihe Informatik; Bd. 56).
- [She92] P. P. Shenoy. Using possibility theory in expert systems. In *Proc. of the 1st Int. Conf. on Deductive and Object-Oriented Databases (52)*, Kyoto, Japan, pages 129-142. Elsevier Science Publishers B.V. (North-Holland), 1992.
- [SG93] P. Srinivasan, D. Gracânin. Approximate Reasoning with Fuzzy Petri Nets. In: *Second IEEE Int. Conf. on Fuzzy Systems*, San Francisco, California, 1993.
- [SK90] F. N. Springsteel, C. Kou. Reverse Data Engineering of E-R Designed Relational Schemas. In *Proc. of Databases, Parallel Architectures and their Applications*, pages 438-440. Springer, März 1990.
- [SLGC94] O. Signore, M. Loffredo, M. Gregori, M. Cima. Reconstruction of er schema from database applications: a cognitive approach. In *Proc. of the 13th Int. Conference of ERA*, Manchester, pages 387-402. Springer, 1994.
- [Sme88] P. Smets. Belief Functions. In P. Smets, E. H. Mamdani, D. Dubois, H. PRade (editors) *Non-Standard Logics for automated Reasoning*. pages 153-286. Academic Press, London, 1988.

- [Zad75] L. Zadeh. Fuzzy Sets. In *Proc. of the 1st Int. Conf. on Deductive and Object-Oriented Databases* (8), Kyoto, Japan, pages 338-353, Elsevier Science Publishers B.V. (North-Holland), 1975.
- [Zad78] L. Zadeh. Fuzzy Sets as a Basis for a Theory of Possibility. In *Fuzzy Sets and Systems* 1 (1978) 3-28.

## Index

### A

Abbruchbedingung 33, 34, 35, 56  
 Abbruchkriterium 81  
 Ablehnung 16  
 Ableitungsregel 11, 42, 47  
 AFB 50, 55, 64  
 aktiviert 44  
 Aktivierung 40, 41  
 Aktualisierungsparameter 29, 33  
 All-Quantor 31  
 Analyse, statische 54  
 Analyseergebnis 6, 11, 76, 87, 91  
 Analyseergebnis, aussagekräftiges 7, 49  
 Analyseergebnis, Verfälschung 54  
 Analyseoperation 7, 26, 30, 52, 69, 72, 73, 83, 85, 88, 89  
 Analyseoperation, aufwendige 26  
 Analyseoperation, teure 7  
 Analyseoperation, verspätete 71  
 Analysewerkzeug 5, 7, 92  
 Analysewerkzeug, wissensbasiertes 3  
 Anfangsmarkierung 40  
 Anfechtbare Logik 11, 12, 21  
 Anforderungen 6, 7  
 Anforderungsanalyse 1  
 Anfragemuster 4  
 Anwendungsprogrammierer 27  
 Applikationscode 1, 2, 5, 48, 69  
 Äquivalenzrelation 12  
 Assoziativität 17  
 Attributnamen, ähnliche 27, 73, 84  
 Ausführung eines FRNs 51, 53  
 Ausführungsmaschine 8, 39, 75, 91  
 Ausführungsmechanismus 7, 25  
 Ausgangsfunktion 43  
 Aussage 43, 50, 76  
 Aussagenlogik 11, 25  
 Auswertungsschritt 53, 63, 78, 81  
 Axiom 26  
 Axiom, aufgeschobenes 26, 27, 30, 31, 46, 52, 70, 71, 72, 73, 74, 88  
 Axiom, schwaches 26, 46, 52, 72  
 Axiom, starkes 26, 27, 30, 31, 46, 52, 69, 70, 71, 72, 73, 83, 84  
 Axiome rationalen Verhaltens 14

### Axiomstypen 27

### B

Begrenzungskreis 53, 63, 65, 76, 80, 81, 91  
 Begrenzungskreis, Elimination 63, 64, 67, 75, 81, 91  
 Begrenzungskreis, Erkennung 75, 76  
 Begrenzungskreis, Lokalisierung 76  
 Begrenzungskreis-Eigenschaft 62  
 Beziehung, is\_a 5  
 Beziehungen, Ableitung von 3

### C

changing 56  
 Cliché 21, 73  
 Cliché, cyclic-exclusion 46, 47  
 Cliché, cyclic-join 4, 5, 7, 21, 27, 69  
 Cliché, join 47  
 Cliché, select-distinct 4, 5, 7, 21, 27, 72, 84  
 Clichés 4  
 Clichés, Aufbau von 4  
 Clichés, Finden von 4  
 Constraint 29, 33, 35  
**cycl\_join** 27

### D

Daten 2  
 Datenaustausch 1  
 Datenbankadministrator 2  
 Datenbankanalyse 5, 69  
 Datenbankausprägung 27  
 Datenbankentwurf 1  
 Datenbankmigration 1, 2  
 Datenbankschema 2  
 Datenbankwissen, generisches 6  
 Datenhaltung, konsistente 1  
 Deduktion 20, 21  
 Deduktionsregel 12  
 DeMorgan 70  
 Denkweise, menschliche 22  
 Denormalisierung 2  
 distinct 4, 5, 27  
 Distributivität 17  
 DK 56  
 Dokumentation 1  
 DominanceCount 78  
 Dominanzkreis 56, 60, 77, 80  
 Dominanzpfadausdruck 77

- DPA 79  
DPA, kompletter 78, 80, 81  
DPE 77  
dpe 76  
Duplikateliminierung 27
- E  
EER-Notation 5  
Effizienz 7, 23  
Eigenschaften, semantische 4  
Eingangsfunktion 43  
Eingangsstelle 44  
Eintrittswahrscheinlichkeit 13, 14  
Elemente, passive 39  
Elimination von Inkonsistenzen 48  
Empfindung, subjektive 16  
**equiv.** 27  
Erdung 48  
Ereignis 13  
Ereignisraum 13  
Erfahrung 15, 22  
Ergebnisinterpretation 39  
Erhebung, statistische 13  
ES 13  
evaluate() 75  
Evaluator 75  
Existierende Ansätze 8  
Exklusivität elementarer Aussagen 14  
Expansion 69, 75, 83  
Expansionsalgorithmus 39, 70  
Expertenschätzung 22  
Expertensystem 13, 42  
Expertenwissen 15
- F  
Falsifikation von Analyseergebnissen 7  
FBM 50  
Fehlerabschätzung 22  
Feuern einer Transition 41  
Flußrelation 39, 40, 50  
Föderierung von Datenbanken 1  
Folgemarkierung 40, 41  
formale Syntax der FRNs 50  
Formel erster Ordnung, notwendigkeitsbe-  
wertete 20  
Formel, Interpretation 11  
Formel, notwendigkeitsbewertete 20  
Fragwürdigkeitsmaß 15, 16
- Fremdschlüsselbeziehung 2, 3, 5, 27, 29, 46,  
47, 48, 89  
FRN 39, 49, 50, 69, 91  
FTT 44  
Funktion, boole'sche 85  
Funktion, boole'sche 30, 87  
Funktion, rekursive 33  
Funktion, relationale 30, 86  
Funktionssymbol, boole'sches 34  
Funktionssymbol, relationales 33  
Fuzzy Logik 16  
Fuzzy Petrinetz 43  
Fuzzy Reasoning Net 39, 49, 50, 91  
Fuzzy-AND-Operation 43, 44, 45  
Fuzzy-Belief 43, 44, 45, 48, 50, 51, 52, 53,  
55, 61, 62, 77, 79  
Fuzzy-Belief, absoluter 50, 63, 64  
Fuzzy-Belief, negativer 50, 76  
Fuzzy-Belief, positiver 50, 76, 84  
Fuzzy-Belief-Marking 50  
Fuzzy-Gain 63, 65, 76  
Fuzzy-Gain, negativer 64  
Fuzzy-Gain, positiver 64  
Fuzzy-Inferenz 17  
Fuzzy-Logik 8, 11, 22, 43, 44  
Fuzzy-Menge 16, 17, 18, 19, 20  
Fuzzy-OR-Operation 43, 45, 52  
Fuzzy-Regel 18  
Fuzzy-Relation 18  
Fuzzy-Truth-Token 44, 51, 52, 53, 55, 58,  
62, 76, 77, 79  
Fuzzy-Variable 19  
Fuzzy-Verkettung 18  
Fuzzy-Wissen 18
- G  
Generic Fuzzy Reasoning Net 5, 8, 25, 29,  
49, 69, 91  
Gesamt-Fuzzy-Gain 63  
Gesetz von DeMorgan 17  
Gesetze, mengentheoretische 17  
Gewichtsfunktion 40  
Gewichtung einer Formel 33  
Gewichtung einer Implikation 21  
Gewichtung von Indikatoren 5  
Gewichtungsfaktor 42  
GFRN 91  
GFRN, expandiertes 89

- GFRN-Implikation 69
- GFRN-Regel 69
- GFRN-Spezifikation 6, 8, 27, 34, 39, 46, 47, 49, 69, 83, 85
- GFRN-Spezifikation, Ausführung einer 6
- GFRNToFormular(G) 33
- Glaubwürdigkeitsmaß 15, 16
- Gleichgewichtszeit 53
- Gleichgewichtszustand 45, 49, 53, 54, 63, 67, 72, 74, 81, 87, 89
- Graph, bipartiter 39
- Graph, gerichteter 39
- Grundlage, mathematische 13
  
- H
- hat-Beziehung 5
- Heuristik 2, 3, 7, 11, 15, 21, 22, 27, 69, 91
  
- I
- Implementierungsdetail 75
- Implikation 14, 29, 49
- Implikation, unscharfe 17
- Implikationen in GFRNs 25
- incons 20
- IND 27, 28
- Index, eindeutiger 76
- Indikator 2, 21
- Indikatoren, widersprüchliche 4
- Inferenz 8, 23, 26
- Inferenzalgorithmus 83
- Inferenzmaschine 6, 39, 49, 53, 64, 69, 71, 91
- Inferenzmaschine, Logik der 21
- Inferenzprozeß, nichtmonotoner 72
- Inferenzregeln 8
- Information, semantische 6, 7, 91
- Inkonsistenz 50
- Inkonsistenzen 42, 48, 49, 91
- Inkonsistenzen, Ortung von 49
- Inkonsistenzmaß 20
- Instanz einer GFRN-Regel 69
- Integration von Datenbanken 1
- Integritätsbedingungen 2
- Interaktion 7, 11, 49, 92
- Interpretation 19, 20, 26
- Interpretation des Sicherheitsfaktors 16
- Interpretation von Analyseergebnissen 48
- Interpretation, eindeutige 91
  
- J
- Java-Applets 81
- Java-Applikation 81
- JavaBeans 81
  
- K
- K/M-Axiom 46, 78
- K/M-Netz 39, 42, 43, 46, 47
- Kante 29, 76
- Kante, dominante 54, 77, 80
- Kante, gerichtete 25
- Kantenrichtung 76
- Kantenvorzeichen 33, 49, 50, 55, 65, 70, 76, 79
- Kapazitätsfunktion 40
- key** 27
- Klasseneinteilung 17
- KM-Axiom 46
- Knoten, aktiver 39
- Kolmogoroff, Axiomensystem von 14
- Kommutativität 17
- Kommutativität, UND-Verknüpfung 70
- Komplement 17
- Komplexität 5, 7, 16
- Konfidenz 15, 31, 33, 42, 44, 50, 70, 76
- Konfidenz, Gewichtung einer 27
- Konfidenz-Assoziation 30
- Konfidenzen in GFRNs 25
- Konfidenzfaktor 44
- Konsequenz 42
- Konsequenz, logische 21
- Kontraposition 26, 47, 48, 69, 73
- Kontrapositionsregel 70
- Kontrapositionsregel, Ausnahme 71
- Kontrapositionsstelle 55, 56, 58, 59, 80
- Kripke-Modell 11
- KS 55
  
- L
- Laufzeitminimierung 71
- Laufzeitverhalten 7
- Lebensphasen einer Datenbank 1
- Legacy-Datenbank 1, 6, 8
- Levenshtein-Distanz 84
- Limitcycle 55
- Logik erster Ordnung 18
- Logik, mehrwertige 21
- Logikkalküle, numerische 11

Logikkalküle, symbolische 11  
 Lösungsansatz 8

## M

main-Methode 75  
 M-aktiviert 41  
 Management von Inkonsistenzen 48  
 manuelle Herleitung semantischer Information 2  
 Marke 39, 40, 43, 45  
 Markenanzahl 53  
 Markenfluß 40, 43  
 Markenpaar, initiales 52  
 Markenposition 53  
 Markierung 40  
 Markierung, initiale 44  
 Markierungsfunktion 50  
 mathematische Grundlage 13  
 Maximumoperation 23  
 Mehrwertigkeit der Probabilistischen Logik 14  
 Menge der wahren Aussagen 17  
 Messung 13  
 Minimumoperation 23  
 Modale Logik 11, 21  
 Modell 20  
 Modell, ausführbares 6, 8, 91  
 Modell, formales 49  
 Modellierungskonstrukte, höhere 4  
 Modellierungskonstrukte, objektorientierte 4  
 Modus Ponens 12, 17, 18  
 Möglichkeit einer Aussage 11  
 Möglichkeitsmaß 18, 19  
 Möglichkeitsverteilung 18  
 Möglichkeitsverteilungsfunktion 19, 20

## N

Nachbereich 29, 40, 41, 42, 52  
 Nebenläufigkeit, modellinhärente 8  
 Necessitation-Regel 12  
 Netz 39  
 Nichtmonotonie 7  
 Normalisierungsbedingung 20  
 Notwendigkeit 23, 25, 30, 71, 73, 89, 91  
 Notwendigkeit einer Aussage 11  
 Notwendigkeiten, abgestufte 23  
 Notwendigkeitsmaß 18, 19

nsimilar 27

## O

Optimierung, manuelle 27  
 Optimierungsstrukturen 2  
 Oszillation, periodische 55

## P

Parallelität 42  
 Parameter, aktuelle 34, 72, 73, 85  
 Parameter, formale 25, 72  
 Parameter, nichttechnische 5  
 Parameter, technische 5  
 periodic oszillation 56  
 Petrinetz, unscharfes 6, 8, 42, 47, 50, 69, 91  
 Petrinetzmodell, unscharfes 49, 50  
 Pfad, azyklischer 66  
 Position 29  
 Possibilistische Logik 8, 11, 18, 23, 91  
 Prädikat 27, 29  
 Prädikat, negiertes 70  
 Prädikate in GFRNs 25  
 Prädikatenlogik erster Ordnung 25, 31, 69  
 Präferenz-Ordnung 20  
 Präzision 16  
 Priorität 12  
 Priorität einer (inferenz-) Regel 13  
 Probabilistische Logik 11, 13, 14, 15, 22, 23  
 Problem der Abwärtskompatibilität 2  
 Problemlösung 16  
 Programmierstil 5  
 Projektion 29, 46, 86, 87

## R

Randbedingungen, Unschärfe von 22  
 RDBA 2  
 Redundanzen 2  
 Reengineering 1  
 Reengineering-Wissen 5, 11  
 Reengineering-Wissen, generisches 22, 91  
 Rekursion 33, 35  
 Rekursion in Ableitungsregel 22  
 Relation, binäre 11  
 relationalen Datenbankanwendung 2  
 Reliabilität 22  
 Reverse Engineering 1, 2, 5  
 Rückgabeformel 33, 35  
 Rückwärts-Expansion 72, 73, 88

## S

S/T-Netz 39  
 sameTable-Bedingung 29, 86  
 Satz von Bayes 14, 22  
 Schalten einer Transition 43, 45  
 Schaltverhalten 40, 41, 49, 50, 91  
 Schaltverhalten eines K/M-Netzes 44  
 Schaltverhalten, Steuerung 50  
 Schätzung 13, 15, 22  
 Schema 84  
 Schema, konzeptionelles 1, 3, 11, 91  
 Schließen, approximatives 17  
 Schließen, logisches 42  
 Schließen, nichtmonotones 39, 48, 71  
 Schlüsseleigenschaft 3, 27, 46, 48, 71, 89  
 Schlüsselkandidat 3, 25, 29  
 Schlußfolgern, menschliches 16, 43  
 Schlußfolgerungspfad 54, 65  
 Schlußfolgerungspfad, azyklischer 63, 76  
 Schlußfolgerungsprozeß, nichtmonotoner  
     11, 91  
 Schnittbildung 17  
 Schranke, obere 22  
 Schranke, untere 19, 22, 23, 42, 49, 73, 91  
 Schreibvereinfachung 70  
 Schwellwert 42, 44, 50, 67, 76  
 Schwellwert, Erhöhung 65  
 Schwellwert, initialer 48  
**sel\_dist** 27  
 select\_distinct Statement 25  
 Semantik der GFRNs 31  
 Semantik der Wahrscheinlichkeitswerte 13  
 Semantik einer RDBA 21  
 Semantik einer GFRN-Regel 25  
 Semantik, statische 1  
 Sicherheitsfaktor 23  
 Sicherheitsfaktoren 11, 15, 22  
 Spezifikation, abstrakte 91  
 Sprachstandard 2  
 SQL'92 2, 3  
 SQL-Anfrage 2  
 Stabilitätsanalyse 49  
 Stabilitätsprobleme 54  
 Statistik 13  
 Status 62, 76, 78, 80  
 status 76  
 Status einer Stelle 56  
 Status, initialer 62, 77

steady-state 45, 56, 63, 78  
 Stelle 39, 40, 43, 50, 76  
 Stellen/Transitions-Netz 39  
 Stellen-Vorbereich 54, 61, 78, 80  
 Struktur der Arbeit 8  
 Szenario 83

## T

Tautologie 33, 34  
**tcomp** 27  
 Teilmengenbeziehung 28, 29, 46, 48  
 Transition 39, 40, 43, 50, 76  
 Typkompatibilität 28

## U

Übersetzungsmechanismus 8  
 Unabhängigkeit von Aussagen 22  
 Unsicherheit 7, 13, 42

## V

**validInd** 27  
**validKey** 27  
 Variablen, linguistische 17  
 Variablenbindung 86  
 Variante 4  
 Varlet 5, 6, 7, 91  
 Verbund 46  
 Verbund über Attribute 4  
 Vereinigung 17  
 Vererbung 4  
 Verfälschung, minimale 65  
 Verknüpfung, additive 23  
 Verknüpfung, disjunktive 15  
 Verknüpfung, konjunktive 15  
 Verknüpfungen auf Fuzzy-Mengen 17  
 VisualAge for Java 81  
 Vollständige Induktion 59  
 Vorbedingung 33, 42  
 Vorbereich 29, 40, 41, 42, 44, 51, 60, 61, 79,  
     85  
 Vorwärts-Expansion 72, 73, 86, 88  
 Vorzeichen 29  
 Vorzeichenfunktion 50

## W

Wahrheitsgrad 42  
 Wahrheitswert 42, 43  
 Wahrscheinlichkeit 13, 14, 16, 22

- Wahrscheinlichkeit, bedingte 15, 22
  - Wahrscheinlichkeiten, bedingte 14
  - Wahrscheinlichkeiten, geschätzte 14
  - Wahrscheinlichkeiten, objektive 14
  - Wahrscheinlichkeiten, subjektive 14
  - Wahrscheinlichkeitsfunktion 13
  - Wahrscheinlichkeitsmaß 13, 14
  - Wahrscheinlichkeitstheorie 13
  - Wahrscheinlichkeitswert 13
  - Werte-Beschränkung 19
  - Wertetabelle 17
  - Widerspruch 5, 7, 11, 21
  - Widerspruch, Auflösung 7
  - Widersprüche 7
  - Windows 95 81
  - Wissen 5
  - Wissen, generisches 3, 5, 69
  - Wissen, unsicheres 7, 42
  - Wissen, widersprüchliches 7
  - wissensbasierter Ansatz 5
  - Wissensbasis 42
  - Wissensspezifikation, generische 25
  - Wissensspezifikation 27
  - wohnt\_in-Beziehung 5
- Z
- Zielkonflikt 16
  - Zufälligkeit 22
  - Zufallsexperiment 13
  - Zugehörigkeitsfunktion 17, 18, 19
  - Zugehörigkeitsgrad 16, 17, 43
  - Zugehörigkeitsgrad, kontinuierlicher 16
  - Zustand, stabiler 53, 63, 91
  - Zustimmung 16
  - Zweiwertigkeit 16
  - Zyklen, Elimination von 54
  - Zyklus 49, 53, 76, 78, 80