Universität Paderborn
Die Universität der Informationsgesellschaft

Institut für Informatik
Arbeitsgruppe Softwaretechnik
Warburger Str. 100

33098 Paderborn

Optimierung von Genauigkeitswerten unscharfer Regeln

DIPLOMARBEIT

für den integrierten Studiengang Informatik
im Rahmen des Hauptstudiums II

von

Carsten Reckord
Bergstraße 53
33415 Verl

vorgelegt bei

Prof. Dr. Wilhelm Schäfer
und
Prof. Dr. Gregor Engels

Mai 2004

i.

Eidesstattliche Erklärung

Ich versichere, daß ich die vorliegende Arbeit selbständig und ohne unerlaubte fremde Hilfe sowie ohne Benutzung anderer als der angegebenen Quellen angefertigt habe. Alle Ausführungen, die wörtlich oder sinngemäß übernommen worden sind, sind als solche gekennzeichnet. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen.

Paderborn, den 26. Mai 2004

Carsten Reckord

Inhaltsverzeichnis

KAPITEL 1 EINFÜHRUNG 1

1.1 PROBLEMBESCHREIBUNG 2
1.2 ZIELSETZUNG 3
1.3 AUFBAU DER ARBEIT 4

KAPITEL 2 DER INFERENZPROZESS 5

2.1 DESIGN PATTERN SPEZIFIKATION 5
2.1.1 Der abstrakte Syntaxgraph 5
2.1.2 Die Spezifikationssprache 6
2.2 DAS PATTERN DEPENDENCY NET 12
2.2.1 Struktur des Pattern Dependency Net 12
2.2.2 Der Erkennungsprozess 16
2.3 DIE FUZZY EVALUATION 21
2.3.1 Struktur des Fuzzy Petrinetzes 21
2.3.2 Auswertung des Fuzzy Petrinetzes 23
2.4 BENUTZERINTERAKTION 24

KAPITEL 3 ADAPTION VON WISSEN 29

3.1 MASCHINELLES LERNEN 29
3.2 ADAPTION VON FUZZY-REGELN UND UNSCHARFEN PETRINETZEN 32
3.3 REGRESSION UND GRADIENT DESCENDANT TECHNIKEN 33
3.4 NEURONALE NETZE UND BACKPROPAGATION 34
3.4.1 Das Fuzzy Neural Net 34
3.4.2 Konstruktion der Lernmuster 39
3.4.3 Der Backpropagation-Algorithmus 41
3.5 PROBLEME VON GRADIENT DESCENDANT VERFAHREN 42

KAPITEL 4 DER LÖSUNGSANSATZ: STATISTISCHE ADAPTION IM FPN 45

4.1 ENTWURFSKRITERIEN 45
4.2 STATISTISCHE ADAPTION 46

4.3 ADAPTION DER VERTRAUENSWERTE 48
4.3.1 Senken des Vertrauenswertes 50
4.3.2 Erhöhen des Vertrauenswertes 56
4.3.3 Bestätigen des Vertrauenswertes 58
4.4 STATISTISCHE AUSWERTUNG 58
4.4.1 Sammeln der Änderungsanforderungen 59
4.4.2 Zusammenführen der Lernmuster 59
4.4.3 Adaption der Vertrauenswerte 60
4.5 ERWEITERUNGEN 62
4.5.1 Bereinigung der Datensätze 62
4.5.2 Wahl der Ausgabestellen 64
4.5.3 Situationsspezifisches Wissen und gedächtnisbehaftetes Lernen 65
4.6 ADAPTION DER SCHWELLWERTE 66
4.7 VERALLGEMEINERUNG 69

KAPITEL 5 EVALUATION 71

5.1 VORGEHEN 71
5.2 VERGLEICH MIT FUZZY NEURAL NETS 73
5.3 FAZIT 77

KAPITEL 6 ZUSAMMENFASSUNG UND AUSBLICK 79

6.1 ZUSAMMENFASSUNG 79
6.2 AUSBLICK 81
6.2.1 Statistische Adaption 81
6.2.2 Optimierungsmöglichkeiten im Inferenzprozess 82

ANHANG A LITERATURVERZEICHNIS 83

ANHANG B ABBILDUNGSVERZEICHNIS 85

KAPITEL 1

Einführung

Eines der zentralen Aufgabenfelder in der Softwareentwicklung ist das *Reengineering*. Im Gegensatz zu der vollständigen Neuentwicklung eines Softwaresystems ist das Ziel des Reengineering die Weiterentwicklung existierender Systeme, um sie etwa an geänderte Anforderungen anzupassen oder um neue Funktionalität zu ergänzen. Insbesondere große Softwaresysteme werden so häufig von unterschiedlichen Entwicklern über mehrere Generationen hinweg entwickelt und gepflegt.

Um ein solches sogenanntes *Legacy-System* erfolgreich weiterentwickeln und warten zu können, ist zunächst ein gutes Verständnis von Aufbau und Funktionsweise des Systems erforderlich. Hierzu kann aber in der Praxis häufig nicht auf eine vorhandene Design-Dokumentation zurückgegriffen werden, da diese - sofern überhaupt vorhanden - zumeist nur das initiale Design des Systems reflektiert, während spätere Änderungen und Erweiterungen zumeist undokumentiert bleiben. Auch die ursprünglichen Entwickler stehen insbesondere bei älteren Systemen häufig nicht mehr zur Verfügung, so daß für die Einarbeitung eines neuen Entwicklers in das System zumeist nur der Systemquelltext genutzt werden kann.

Die fehlenden Designinformationen aus dem Quelltext und der verfügbaren Dokumentation zu rekonstruieren und in einer abstrakteren und übersichtlicheren Form dem Entwickler zugänglich zu machen ist Aufgabe des *Reverse Engineering*. Aufgrund der Bedeutung des Reverse Engineering im gesamten Entwicklungsprozess gibt es mittlerweile eine breite Palette an Werkzeugen, die den Entwickler bei dieser Aufgabe unterstützen. Üblicherweise erlauben diese Werkzeuge (wie Borland Together, Rational Rose oder FUJABA) die automatische Erzeugung von *Klassen-* und *Verhaltensdiagrammen* in der *Unified Modelling Language* (UML, [UML]) aus dem gegebenen Quelltext. Eine weitergehende Analyse der Zusammenhänge zwischen einzelnen Systemelementen und ihrer Funktion im Gesamtsystem wird jedoch von den meisten Werkzeugen nicht unterstützt.

Design Pattern bieten hier eine gute Möglichkeit, auch komplexere Zusammenhänge zwischen mehreren Elementen kompakt auszudrücken und erlauben so dem Entwickler einen schnelleren und besseren Überblick über das Gesamtsystem. Design Pattern (oder Entwurfsmuster), die erstmals von Gamma, Helm, Johnson und Vlissides in [GHJV95] vorgestellt worden sind, sind Standard-Lösungsansätze für häufig wiederkehrende Designaufgaben. Ursprünglich als Entwurfselemente auf hohem Abstraktionsniveau für das Forward Engineering konzipiert, abstrahieren sie von einzelnen Elementen hin zu einem übergeordneten Design-Prinzip. Umgekehrt kann aber auch von dem Design Pattern auf die Funktionen der daran beteiligten Systemkom-

ponenten geschlossen werden. Daher ist ein zuverlässiges Erkennungsverfahren für Design Pattern eine wichtige Hilfe im Reengineering-Prozess.

1.1 Problembeschreibung

Design Pattern sind von Gamma et al. ursprünglich für das Forward Engineering vorgestellt worden. Ihre Beschreibung ist in hohem Maße informell und erlaubt dem Entwickler daher viele Freiheiten beim Entwurf und bei der Implementierung. Im Reverse Engineering führt dies zu Problemen, da für eine automatische Mustererkennung eine formale Spezifikation der zu suchenden Muster erforderlich ist. Zudem muß ein automatischer Erkennungsmechanismus nicht nur eine spezifische Patternausprägung erkennen, sondern hat eine Vielzahl möglicher Varianten zu berücksichtigen.
Ein weiteres Problem stellt die Größe der zu untersuchenden Systeme dar. Typische Softwaresysteme liegen häufig in Größenordnungen von einigen hunderttausend bis Millionen Zeilen Quelltext, woraus sich besonders hohe Anforderungen an die Skalierbarkeit des verwendeten Erkennungsmechanismus ergeben.

Ein Mustererkennungssystem, das diese beiden Probleme addressiert, ist von Lothar Wendehals in seiner Diplomarbeit [Wen01] vorgestellt worden. Dieses System beinhaltet eine objekt-orientierte Musterspezifikationssprache, die eine formale Spezifikation der Design Pattern auf Basis von Graphersetzungsregeln erlaubt.
Das Variantenproblem wird durch die Verwendung von Unschärfe in der Spezifikation gelöst. So ist es möglich, durch eine einzelne, möglichst allgemein gehaltene Musterspezifikation mehrere Varianten eines Pattern abzudecken.

Aufgrund der mangelnden Präzision bei der ursprünglichen Patternbeschreibung und des daraus resultierenden Variantenproblems können nicht immer exakte Erkennungsergebnisse gewährleistet werden. Insbesondere entsteht durch eine verhältnismäßig weit gefaßte unscharfe Patternspezifikation die Gefahr, falsche Patternvorkommen, sogenannte *False Positives*, zu finden. Um dem zu begegnen erfolgt eine Gütebewertung der gefundenen Analyseergebnisse auf Basis fuzzy-logischer Abschätzungen, die jeder gefundenen Patternausprägung einen *Vertrauenswert* (*Fuzzy Belief*) zuordnet. Diese Vertrauenswerte können von dem Benutzer bei der Auswertung der Analyseergebnisse herangezogen werden, um etwa besonders unsichere Ergebnisse zu kontrollieren. Die Mustererkennung ist daher als halbautomatischer Prozess konzipiert, in den der Benutzer zu jeder Zeit eingreifen kann, um auf Basis seines eigenen Wissens und seiner Erfahrung die von dem Prozess gelieferten Zwischenergebnisse zu korrigieren und so die Präzision der Erkennung zu erhöhen.

Da die Vertrauenswerte der gefundenen Patternausprägungen dem Benutzer als Grundlage für die weitere Auswertung und mögliche Kontrollen der Ergebnisse dienen, haben sie entscheidenden Anteil an der Qualität der Patternerkennung. Die Festlegung der für die Fuzzy-Auswertung entscheidenden Parameter und damit der Qualität der Vertrauenswerte ist zur Zeit jedoch dem Benutzer überlassen. Die optimalen Werte für diese Parameter hängen direkt von der Qualität

der Patternspezifikation ab und haben sich in der Praxis als nur schwer abschätzbar erwiesen. Sie werden zudem in hohem Maße von der Einsatzumgebung beeinflußt - etwa durch Styleguides und Programmierkonventionen für das zu analysierenden Softwaresystem. Daher ist hier eine automatische Adaption der Fuzzy-Parameter auf Basis der Benutzereingaben während der Mustererkennung einer manuellen Anpassung mittels Abschätzen durch den Benutzer vorzuziehen.

Abbildung 1.1 zeigt einen schematischen Überblick über den Inferenzprozess in seiner bisherigen Form. Insbesondere ist zu erkennen, daß die Fuzzy Beliefs der gefundenen Musterinstanzen zwar vom Benutzer angepaßt werden kann, jedoch keine Rückkopplung von den Benutzerkorrekturen zu den Fuzzy-Parametern der Musterspezifikationen erfolgt.

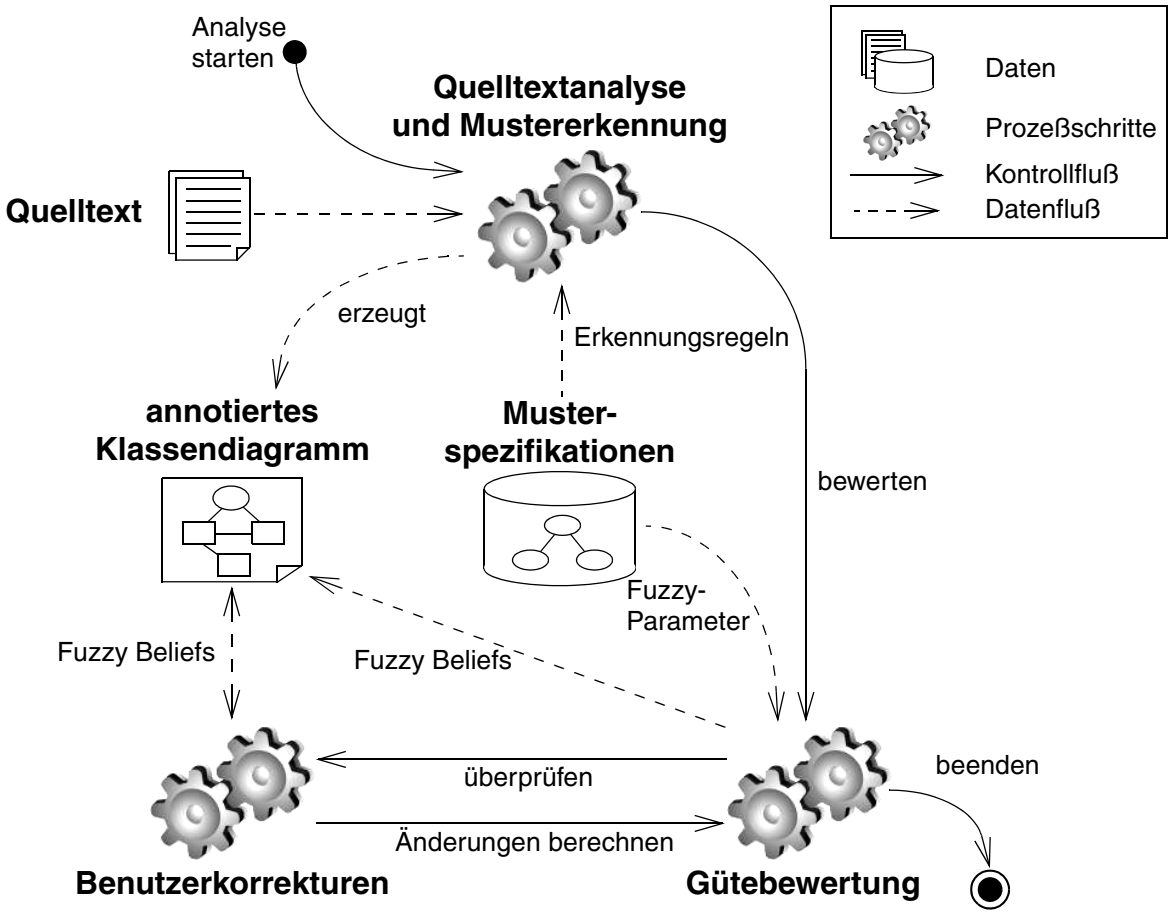


Abbildung 1.1: Überblick über den Mustererkennungsprozess

1.2 Zielsetzung

Im Rahmen dieser Diplomarbeit wird ein Adaptionsverfahren entwickelt, das die Anpassung der Fuzzy-Parameter für die Design Pattern Erkennung nach [Wen01] ermöglicht. Grundlage für die Adaption sollen die manuellen Anpassungen der Fuzzy-Werte durch den Benutzer während eines Erkennungsprozesses sein. Das Ziel ist somit die Realisierung der in Abbildung 1.1

fehlenden Rückkopplung von den Benutzerkorrekturen zu den Patternspezifikationen mit Hilfe dieses Adaptionsverfahren.

Eine Adaption der Fuzzy-Parameter auf Basis konnektionistischer Methoden (*Fuzzy Neural Nets, FNN*, [Str99]) hat sich in einem ähnlichen Projekt ([Jah99]) aufgrund der Laufzeitkomplexität des Lernenden-Netzes als nicht anwendbar herausgestellt, da die Auswertung des Netzes ein Vielfaches der übrigen Analysezeit benötigt hat. Ähnliches läßt sich für andere klassische iterative Lernansätze herleiten. Daher wird bei dem zu entwickelnden Adaptionsverfahren besonderes Augenmerk auf die Skalierbarkeit hinsichtlich Laufzeit und Speicherbedarf gelegt, damit das Verfahren auch für Analysen von realistischen Projektgrößen im Umfang von mehreren hunderttausend LOC (Lines Of Code) noch anwendbar bleibt.

1.3 Aufbau der Arbeit

In Kapitel 2 wird zunächst das Mustererkennungsverfahren aus [Wen01] als Grundlage des zu entwickelnden Adaptionsalgorithmus zusammenfassend beschrieben. Zudem werden einige Veränderungen in dem Verfahren gegenüber [Wen01] erläutert. Hierunter fällt insbesondere die Ausführungsreihenfolge der Patternanalysen, die entsprechend [NSW+02] angepaßt worden ist, sowie einige strukturelle Vereinfachungen, die sich auf die Realisierung des Adaptionsansatzes auswirken.

Kapitel 3 gibt einen Überblick über die Adaption unsicheren Wissens und diskutiert Probleme existierender Lösungsansätze. Insbesondere wird in Abschnitt 3.4 der Ansatz des Fuzzy Neural Nets nach [Str99] noch einmal ausführlich erläutert, da es im späteren Verlauf der Arbeit als Referenzimplementierung zum Vergleich mit dem neu zu erstellenden Verfahren dienen soll. Hierbei wird speziell auf strukturelle Änderungen des FNN zur Anpassung an die verwendete Form der Patternspezifikationen und die Konstruktion der Lernaufgabe für das FNN anhand der gefundenen Patternausprägungen eingegangen, die sich von dem Verfahren in [Str99] unterscheiden.

Thema von Kapitel 4 ist der Lösungsansatz dieser Arbeit. Hier wird zunächst beispielhaft die Änderung einzelner Fuzzy-Parameter untersucht, bevor anschließend die Auswirkungen auf Teilpattern und darauf basierend das entgültige Lernverfahren präsentiert werden.

Kapitel 5 widmet sich der Evaluierung der Lernverfahren. Hierzu wird zunächst der Versuchsaufbau zum Testen der Lernansätze beschrieben. Anschließend folgt ein Vergleich des hier entwickelten Ansatzes mit dem in [Str99] vorgestellten FNN-Ansatz. Hierbei wird neben der Qualität der Lernergebnisse besonders die Skalierbarkeit der Ansätze überprüft. Anhand der so gesammelten empirischen Daten können Vor- und Nachteile beider Ansätze miteinander verglichen werden.

In Kapitel 6 werden schließlich die Ergebnisse dieser Arbeit zusammengefaßt und im Ausblick offene Fragen und weitere lohnenswerte Ansätze für die rechnergestützte Optimierung des Inferenzprozesses aufgegriffen.

KAPITEL 2

Der Inferenzprozess

Die vorliegende Arbeit baut in besonderem Maße auf dem Inferenzprozess zur Design Pattern Erkennung auf, der in der Diplomarbeit „Cliché- und Mustererkennung auf Basis von Generic Fuzzy Reasoning Nets“ von Lothar Wendehals [Wen01] beschrieben ist. Zum besseren Verständnis der Adaptionsalgorithmen, die Gegenstand der folgenden Kapitel sind, wird daher in diesem Kapitel zunächst dieser Inferenzprozess zusammenfassend dargestellt. Insbesondere wird dabei auch auf Änderungen und Ergänzungen gegenüber der ursprünglichen Arbeit eingegangen, die für die zu entwickelnden Adaptionsalgorithmen von Bedeutung sind.

2.1 Design Pattern Spezifikation

Gamma et al. haben, wie bereits in Kapitel 1 erwähnt, Design Pattern ursprünglich für das Forward Engineering auf hohem Abstraktionsniveau entwickelt [GHJV95]. Die von ihnen festgelegte Patternbeschreibung ist in weiten Teilen rein textuell und informal. Zumeist sind die einzigen formalen Bestandteile in der Strukturbeschreibung zu findende Beispielausprägungen des Patterns in Form von Klassendiagrammen. Für das Forward Engineering ist eine solche informale Beschreibung vollkommen ausreichend und läßt dem Entwickler zudem hinreichend Freiraum, das Pattern seinen Bedürfnissen entsprechend anzupassen. Für einen automatischen Erkennungsprozess wird hingegen eine klare formale Spezifikation benötigt, die einen algorithmischen Abgleich mit dem zu analysierenden Quelltext erlaubt. Daher ist es erforderlich, eine formale Repräsentation der Patternbeschreibung zu finden, die möglichst alle relevanten Strukturinformationen der informalen Beschreibung umfaßt.

2.1.1 Der abstrakte Syntaxgraph

Eine wichtige Voraussetzung für die Patterndefinition ist die Möglichkeit, Strukturen und Zusammenhänge im Quelltext einfach und präzise zu beschreiben und auf sie zu verweisen. Hierzu wird in [Wen01] der Quelltext des zu analysierenden Systems nach dem Einparsen durch einen abstrakten Syntaxgraph (ASG) repräsentiert, der auf einem an das UML-Metamodell [UML] angelehnten Metamodell basiert. Neben strukturellen Informationen wie Klassendefinitionen, enthaltenen Attributen und Methoden sowie Vererbungs- und Benutzt-Beziehungen enthält der ASG auch Teilgraphen, die die Methodenrümpfe repräsentieren.

Die Verwendung eines abstrakten Syntaxgraphen zur Repräsentation von Quelltext hat zahlreiche Vorteile gegenüber der direkten Verwendung des Quelltextes. So wird im ASG von der For-

matierung des Quelltextes abstrahiert. Einfache syntaktische Variationen im Quellcode, wie etwa i++ und i=i+1 oder unterschiedliche Schleifenkonstrukte, werden im ASG einheitlich dargestellt. So ist etwa eine Unterscheidung zwischen for- und while-Schleifen im ASG nicht mehr erforderlich. Neben der Vereinfachung der Darstellung bietet die ASG-Darstellung aber auch zusätzliche Informationen über Variablendeklarationen, Namensauflösung und mehr, die während des Parsens aus dem Quelltext gewonnen werden.

Abbildung 2.1 zeigt beispielhaft Auszüge eines Java-Quelltextes und des entsprechenden abstrakten Syntaxgraphen in Form eines Objektdiagrammes. Der Syntaxgraph des Methodenrumpfes ist hier zur besseren Darstellung etwas vereinfacht abgebildet. Detaillierte Informationen über das verwendete Metamodell und den abstrakten Syntaxgraph sind in [FNT98] zu finden.

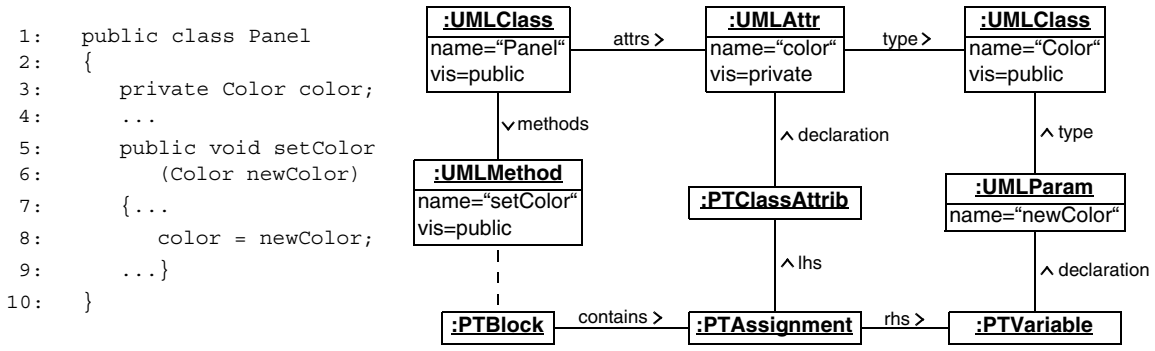


Abbildung 2.1: ASG-Repräsentation von Quellcode

2.1.2 Die Spezifikationssprache

Eine grafische Spezifikationssprache für die formale Patternbeschreibung auf Basis des abstrakten Syntaxgraphen ist von Marcus Palasdies in [Pal01] vorgestellt und von Lothar Wendehals in [Wen01] in das hier verwendete Mustererkennungsverfahren integriert worden. Diese Sprache ermöglicht die Spezifikation von Beispielausprägungen eines Musters, die alle Charakteristika einer gültigen Musterausprägung beschreibt. Entspricht eine potentielle Patterninstanz allen in einer Beispielausprägung festgelegten Charakteristika, so wird sie positiv erkannt. Die Menge der Beispielausprägungen zu einem Muster definiert somit eine Äquivalenzrelation deren zugehörige Äquivalenzklasse alle gemäß Spezifikation gültigen Ausprägungen des Musters umfaßt. Die Spezifikationssprache ist objektorientiert und ihre Syntax orientiert sich an den Standards der UML. Sie ist unterteilt in Objektdiagramme, in denen die Beispielausprägungen definiert werden, und ein Klassendiagramm, das die Beziehungen zwischen den Musterspezifikationen beschreibt.

Die Spezifikation der Beispielausprägungen soll im Folgenden am Beispiel des Composite Pattern erläutert werden. Eine vollständige Beschreibung der verwendeten Spezifikationssyntax ist in [Wen01] und [Pal01] zu finden.

Design Pattern nach Gamma

Abbildung 2.2 zeigt das Klassendiagramm des Composite Pattern nach Gamma et al. Das Composite Pattern beschreibt eine Baumstruktur, in der Baum- und Blattknoten auf gleiche Weise behandelt werden können.

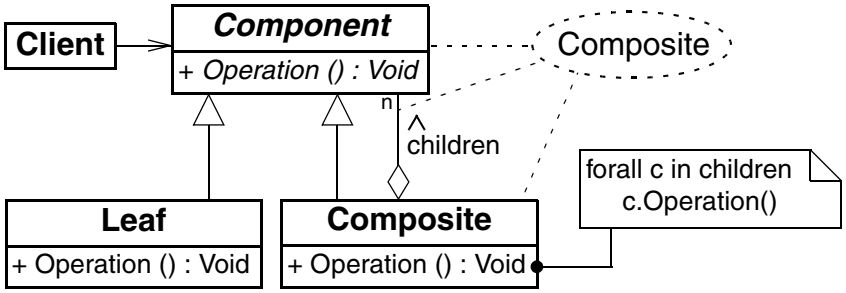


Abbildung 2.2: Das Composite-Pattern

Die Hauptelemente des Pattern sind die Klassen Component und Composite, die die eigentliche Baumstruktur beschreiben. Die Klasse Component ist die Basisklasse für alle Klassen der Baumstruktur. Sie deklariert eine einheitliche Schnittstelle für alle Operationen auf den Elementen des Baumes. Die Klasse Composite erbt von Component und modelliert über die zu-n-Assoziation children eine Vater-Kind-Beziehung zur Klasse Component, wodurch die Baumstruktur realisiert wird. Operationen der Composite-Klasse sind in der Regel als Delegationen auf alle Kind-Objekte realisiert, so daß Aufrufe auf einem Composite-Objekt durch den darunterliegenden Teilbaum an alle Nachfolger propagiert werden. Die Klasse Leaf steht in diesem Beispiel stellvertretend für alle Blattklassen. Blattklassen erben ebenfalls von Component. Im Gegensatz zur Composite-Klasse werden Aufrufe auf einem Blattobjekt jedoch üblicherweise direkt lokal abgearbeitet.
Die Interaktion eines Clients mit Elementen der Composite-Struktur erfolgt durch die in Component deklarierte Schnittstelle. Da alle an der Composite-Struktur beteiligten Klassen von Component erben, ist sichergestellt, daß diese Schnittstelle von allen Elementen zur Verfügung gestellt wird. Für den Client ist daher eine Unterscheidung zwischen Blatt- und Composite-Elementen nicht erforderlich.

Zur Kennzeichnung des Patterns erlaubt die UML das Erstellen von Annotationen im Klassendiagramm. Dazu wird das Pattern als gestricheltes Oval dargestellt, das den Patternnamen enthält. Es ist durch gestrichelte Kanten mit den annotierten Diagrammelementen verbunden, die die Kernelemente des Pattern darstellen.

Das Variantenproblem

Eine formale Spezifikation von Design Pattern und eine darauf aufbauende Mustererkennung werden erheblich erschwert durch die Vielzahl möglicher Varianten eines Patterns.

Durch die informale Patternbeschreibung nach Gamma et al. sind dem Entwickler für die Umsetzung eines Patterns viele Freiräume gelassen, um das Pattern an die eigenen Anforderun-

gen anzupassen. Hierdurch entstehen viele struktuelle Variationen desselben Patterns, die im Erkennungsprozess und damit auch bereits bei der formalen Spezifikation des Patterns zu berücksichtigen sind.

So ist etwa auch das Klassendiagramm aus Abbildung 2.2 lediglich als eine Beispielausprägung zu betrachten. Eine häufig gefundene struktuelle Variation des Composite-Patterns, bei der die Rollen der ursprünglichen Klassen Component und Composite in der Klasse Composite vereinigt sind, ist in Abbildung 2.3 dargestellt.

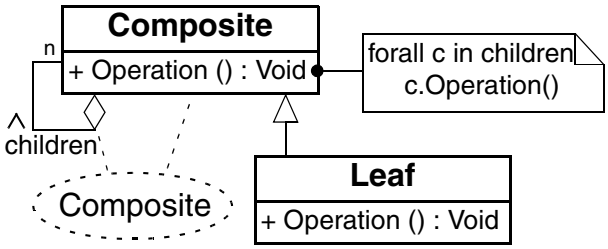


Abbildung 2.3: Eine Variante des Composite-Patterns

Eine weitere Schwierigkeit für die Spezifikation und Mustererkennung stellt die Umsetzung höherer objektorientierter Konstrukte bei der Implementierung dar. Insbesondere strukturelle Beziehungen zwischen den Klassen eines Patterns sind in diesem Zusammenhang problematisch, da die meisten objektorientierten Programmiersprachen etwa für bidirektionale Assoziationen oder zu-n-Beziehungen keine Sprachelemente besitzen. Solche Beziehungen werden zumeist durch Referenzattribute beziehungsweise Containerklassen und entsprechende Zugriffsmethoden abgebildet. Die konkreten Implementierungen können jedoch stark voneinander abweichen. Abbildung 2.4 zeigt beispielsweise zwei unterschiedliche Möglichkeiten, die zu-n-Assoziation children aus Abbildung 2.3 zu implementieren. In Abbildung 2.4a wird hierzu ein Standardcontainer-Objekt verwendet (Zeile 3), auf das über Zugriffsmethoden addToChildren, removeFromChildren und iteratorOfChildren elementweise zugegriffen werden kann (Zeilen 4-12). Die Implementierung aus Abbildung 2.4b verwendet dagegen ein Array (Zeile 3), das dem Benutzer über die beiden Zugriffsmethoden setChildren und getChildren (Zeilen 4-8) zur Verfügung gestellt wird. Die Organisation der Elemente in dem Array bleibt dabei dem Benutzer überlassen.

Aufgrund der hohen Anzahl möglicher Design- und Implementierungsvarianten ist die separate Spezifikation aller möglichen Variationen eines Design Patterns kaum zu realisieren. Daher wird eine Spezifikationssprache benötigt, die einen möglichst großen Teil dieser Variationen in wenigen Spezifikationen auszudrücken vermag.

Patternspezifikation

Die formale Spezifikation eines Patterns erfolgt mit Hilfe von Graphtransformationsregeln[1] auf dem abstrakten Syntaxgraph. Die verwendete Notation orientiert sich an der Notation von UML

1. Eine formale Definition für die hier verwendete Form von Graphtransformationsregeln ist in [Zün01] zu finden.

```
1:  public class Composite
2:  {
3:     private Set children;
4:     public void addToChildren
5:        (Composite child)
6:     {...}
7:     public void removeFromChildren
8:        (Composite child)
9:     {...}
10:    public Iterator iteratorOfChildren
11:       (Composite child)
12:    {...}
13: }
```

(a) Containerklassen

```
1:  public class Composite
2:  {
3:     private Composite[] children;
4:     public void setChildren
5:        (Composite[] children)
6:     {...}
7:     public Composite[] getChildren ()
8:     {...}
9:  }
```

(b) Arrays

Abbildung 2.4: Implementierung von zu-n-Assoziationen

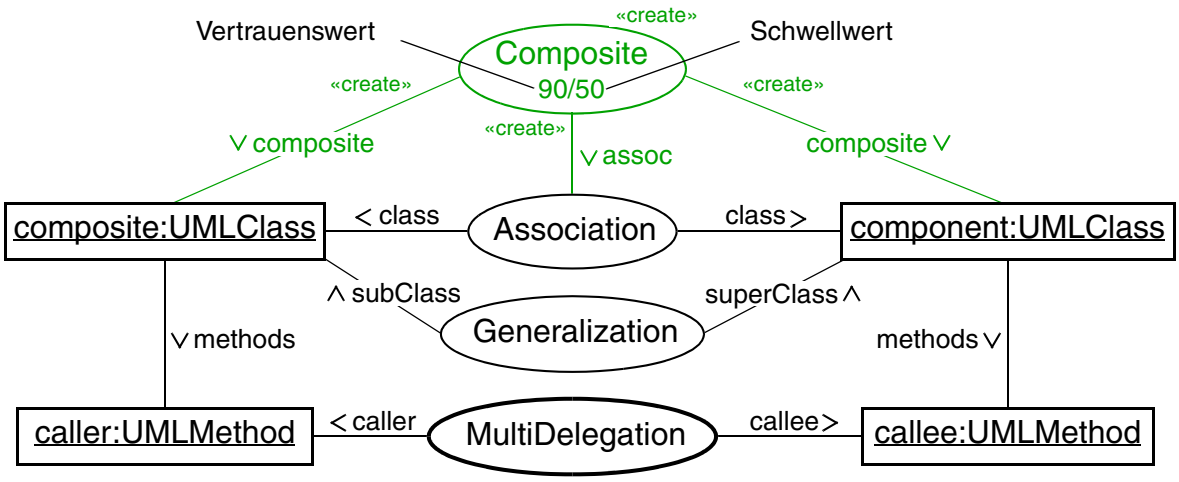


Abbildung 2.5: Formale Spezifikation des Composite-Patterns

Kollaborationsdiagrammen. Eine mögliche Spezifikation des Composite Patterns ist in Abbildung 2.5 dargestellt. Es ist zu erkennen, daß sich die Spezifikation in zwei Teile gliedert.

Die schwarz dargestellten Knoten und Kanten beschreiben die Struktur des Patterns, die im ASG zu suchen ist. Sie repräsentiert also die Vorbedingung der Graphtransformationsregel. In diesem Teil der Spezifikation sind alle wesentlichen Elemente des Patterns wiederzufinden. Die beiden UMLClass-Objekte repräsentieren die Composite- und die Component-Klasse.
Die Assoziation und die Vererbung zwischen den beiden Klassen sind hier durch Subpattern ausgedrückt, die ähnlich der Annotation im Klassendiagramm als Oval mit dem Patternnamen dargestellt werden. Auch die Delegation von Methodenaufrufen im Composite wird durch ein Subpattern ausgedrückt. Die beteiligten Methoden werden durch entsprechende Objekte des ASG repräsentiert.
Der fett dargestellte Rahmen des MultiDelegation Patterns weist es als einen Trigger für das Composite Pattern aus. Trigger werden in dem im nächsten Abschnitt beschriebenen Inferenzalgorithmus zur Steuerung der Patternsuche verwendet. Wird während der Inferenz ein Objekt vom Typ des Triggers gefunden, so wird die Analyse der betreffenden Pattern ausgelöst, die die-

sen Trigger haben. Daher ist es erforderlich, daß in jeder Musterspezifikation ein Element als Trigger markiert ist.
Es ist zu erkennen, daß in der Patternspezifikation die Klassen Leaf und Client nicht repräsentiert sind. Um eine Vielzahl von Mustervarianten abdecken zu können, sollte die Spezifikation stets nur die minimal erforderlichen Teile eines Pattern umfassen. Da die Klasse Client in der Patternbeschreibung nur der Veranschaulichung einer Einbindung des Patterns dient, kann sie bei der Spezifikation weggelassen werden. Die Klasse Leaf ist ebenfalls nicht entscheidend für ein Vorkommen des Musters, da die entscheidende Baumstruktur bereits durch die Component- und Composite-Klasse definiert wird. Die Existenz konkreter Blattklassen ist hingegen für das Composite-Pattern nicht entscheidend. So sind etwa häufig in Softwarebibliotheken Instanzen des Composite-Patterns zu finden, zu denen erst der Benutzer der Bibliothek die konkreten Blattklassen implementiert. Einer Musterspezifikation, die die Leaf-Klasse umfaßt, würde solche Instanzen jedoch nicht mit einschließen.
Neben dieser Minimierung der Spezifikation auf die wesentlichen Elemente existieren noch weitere Sprachkonstrukte wie Pfadausdrücke oder optionale Knoten, mit denen die Zahl der benötigten Patternspezifikationen minimiert werden kann. Für eine genauere Beschreibung hierzu sei auf [Pal01] verwiesen.

Der grüne Teil der Patternspezifikation beschreibt die Annotation des Syntaxgraphen für den Fall, daß die schwarz dargestellten Strukturen im ASG gefunden werden.Alle grün dargestellten und mit dem Stereotyp «create» versehenen Teile der Spezifikation werden dem ASG als neue Elemente hinzugefügt. Es handelt sich hierbei stets um genau einen Musterknoten für das gefundene Pattern und Kanten von diesem Knoten zu den annotierten Elementen. Die neu angelegten Annotationen werden anschließend bei der deduktiven Suche nach Pattern verwendet, in denen das gefundene Pattern als Subpattern enthalten ist.

Vertrauens- und Schwellwerte

Durch eine möglichst allgemein gehaltene Patternspezifikation und die Beschränkung auf die Hauptbestandteile eines Pattern ist es möglich, mit einer Spezifikation eine Vielzahl von Patternvarianten abzudecken. Andererseits wächst hierdurch aber auch die Chance, die spezifizierte Struktur im ASG zu finden, ohne daß es sich tatsächlich um ein Patternvorkommen handelt. Um dem Reverse Engineer die Identifikation solcher False Positives zu erleichtern, ist es sinnvoll, ihm statt einer absoluten Aussage über ein Patternvorkommen eine Gütebewertung über die Analyseergebnisse zu geben, auf deren Basis er ausgewählte Ergebnisse genauer untersuchen kann. Diese Gütebewertung geschieht durch eine Fuzzy Evaluation, die jeder gefundenen Musterausprägung einen Fuzzy-Wert zuweist und in Abschnitt 2.3 noch beschrieben wird. Die Parameter, die als Grundlage für diese Evaluation dienen, sind die Vertrauens- und Schwellwerte, die allen Musterspezifikationen zugeordnet sein müssen.

Der Vertrauenswert ist ein prozentualer Wert zwischen 0 und 100, der das Vertrauen des Reengineers in die Genauigkeit der Musterspezifikation ausdrückt. Eine exakte Spezifikation, die nie False Positives produziert, erhält demnach einen Wert von 100.

Der Schwellwert, der ebenfalls Werte zwischen 0 und 100 annehmen kann, gibt das Minimum der Güte gefundener Musterausprägungen an, die bei der Suche nach dem spezifizierten Muster Verwendung finden. Musterausprägungen mit niedrigerem Fuzzy Belief werden für eine positive Identifikation des spezifizierten Patterns nicht verwendet. Hierdurch kann die Zahl verbleibender False Positives bei geeigneter Wahl des Schwellwertes reduziert werden. Allerdings sollte der Schwellwert auch nicht zu hoch angesetzt werden, da sonst unter Umständen auch tatsächliche Musterausprägungen ignoriert werden.

Die Festlegung geeigneter Werte für Vertrauens- und Schwellwert zu einer Patternspezifikation obliegt dem spezifizierenden Softwaretechniker und sollte sich an seinen Erfahrungswerten mit dem spezifizierten Pattern orientieren. Die Darstellung des Vertrauens- und Schwellwertpaares erfolgt in dem Musterknoten für das spezifizierte Pattern, wie in Abbildung 2.5 dargestellt.

Das Pattern-Klassendiagramm

Das Pattern-Klassendiagramm bildet das Domänenmodell zu den spezifizierten Mustern. Es enthält Klassen für alle spezifizierten Muster, sowie für alle ASG-Objekte, die in den Spezifikationen verwendet worden sind. Klassen zu Musterobjekten sind durch den Stereotyp «Pattern» gekennzeichnet. Zu den in den Spezifikationen verwendeten Links enthält das Klassendiagramm gleichnamige Assoziationen zwischen den Klassen der beteiligten Objekte.

Das Klassendiagramm wird weitestgehend automatisch aus den Musterspezifikationen erzeugt. Die Klassen und Assoziationen, die automatisch auf Basis der Spezifikationen erstellt werden, können vom Benutzer zusätzlich um Vererbungsbeziehungen zwischen den Musterklassen ergänzt werden. Abbildung 2.6 zeigt einen Ausschnitt eines Pattern-Klassendiagramms mit Vererbung. Durch die Verwendung von Vererbung ist es möglich, in einer einzigen Spezifikation

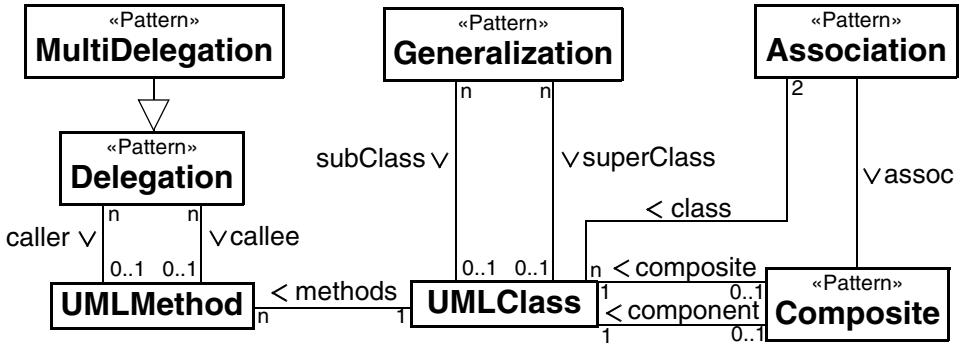


Abbildung 2.6: Das Pattern-Klassendiagramm

eine große Vielfalt unterschiedlicher Musterausprägungen zu beschreiben. Erreicht wird dies durch die polymorphe Bindung gefundener Musterausprägungen an die Musterobjekte in den Spezifikationen während der Mustererkennung. So können etwa im Beispiel des Composite Patterns unterschiedliche Arten von Assoziationen spezifiziert und als Unterklassen der Association Musterklasse definiert werden. Sie finden dann bei der Suche nach einem Composite Pattern mittels Polymorphie Verwendung, ohne daß an der Spezifikation des Composite Änderungen nötig sind.

Auch die Verwendung von Subpattern, die im Klassendiagramm durch Benutztbeziehungen dargestellt ist, ermöglicht eine flexiblere Spezifikation von Pattern. Durch die Komposition von Subpattern ist es möglich, schnell und einfach komplexere Patternspezifikationen neu zu erzeugen. Insbesondere sogenannte Code-Pattern, die Standardelemente objektorientierten Entwurfs repräsentieren wie Reference, Association oder Generalization sind hierfür hilfreich. Mit ihnen können neue Pattern abstrakt formuliert werden ohne die konkrete Umsetzung im ursprünglichen Code kennen zu müssen. Ferner können die Patternspezifikationen so nachträglich leicht an spezielle Programmierkonventionen der Entwickler angepaßt werden, indem nur die Pattern, die in direktem Zusammenhang mit dem Code stehen, entsprechend verändert werden. Alle übrigen Spezifikationen können unberührt bleiben.

Der modulare Aufbau der Spezifikationen unter Ausnutzung von Vererbung und Subpattern ermöglicht zudem eine einfache Zusammenstellung von sogenannten Patternkatalogen. Ein Patternkatalog besteht aus einer Sammlung von ausgewählten Spezifikationen von Design Pattern und den darin verwandten Subpattern sowie dem zugehörigen Klassendiagramm. Durch die Auswahl entsprechender Design Pattern können etwa spezielle Kataloge für unterschiedliche Domänen der zu analysierenden Systeme zusammengestellt werden. Auch eine Anpassung von Code-Pattern wie Reference oder Association an unterschiedliche Programmierkonventionen kann einfach durch Auswahl eines entsprechenden Patternkataloges erfolgen.

2.2 Das Pattern Dependency Net

Nachdem im letzten Abschnitt die Musterspezifikationssprache vorgestellt worden ist, befaßt sich dieser Abschnitt mit dem Algorithmus zur Erkennung der spezifizierten Muster.

2.2.1 Struktur des Pattern Dependency Net

Aufgrund der Verwendung einer Hierarchie aus Pattern und Subpattern für die Musterspezifikation haben die Beziehungen zwischen den Spezifikationen entscheidenden Einfluß auf die Suchreihenfolge. Ein Pattern kann erst dann erfolgreich erkannt werden, wenn vorher alle erforderlichen Subpattern erkannt worden sind. Aus diesem Grund müssen die Informationen über die Patternbeziehungen in die Steuerung des Inferenzprozesses einfließen.
Den Kern des Inferenzalgorithmus bildet daher ein gerichteter Abhängigkeitsgraph, der die Beziehungen zwischen den Patternspezifikationen darstellt. Dieser Graph wird als Pattern Depencdency Net (PDN) bezeichnet. Das PDN kann aus dem vom Benutzer für den Erkennungsprozess ausgewählten Patternkatalog automatisch erstellt werden.

Aufbau des Pattern Dependency Net

Das PDN besteht aus Prädikaten und Implikationen, sowie gerichteten Kanten zwischen diesen beiden Knotentypen. Die Prädikate sind noch einmal unterteilt in einfache Prädikate und Axiome. Jedes Pattern des Patternkataloges wird im PDN durch ein einfaches Prädikat mit gleichem Namen repräsentiert. Die Darstellung im PDN erfolgt in Form eines Ovals mit dem

Namen des Prädikates. Axiome stellen hingegen Basisfakten dar, die unmittelbar aus dem geparsten Quelltext beziehungsweise dem ASG hervorgehen. Sie repräsentieren ausgewählte ASG-Elemente, die als Ausgangspunkte für die Patternsuche dienen. Im PDN werden sie durch ein Rechteck mit fett gezeichnetem Rand dargestellt.
Implikationen bestehen aus einem Vertrauens-/Schwellwertpaar und einer Bedingung, unter der von den Elementen aus dem Vorbereich der Implikation auf den Nachbereich geschlossen werden kann. Jede Spezifikation im Patternkatalog wird durch eine Implikation im PDN repräsentiert. Der Vertrauens- und der Schwellwert der Implikation sind dabei mit den Werten der Spezifikation identisch. Sie werden, wie nachfolgend in Abschnitt 2.3 beschrieben, für den Aufbau des Fuzzy Petrinetzes zur späteren Fuzzy-Bewertung der gefundenen Musterinstanzen benötigt. Die Implikationsbedingung ist gegeben durch den schwarzen Teilgraph der Patternspezifikation, repräsentiert also die Vorbedingung der spezifizierten Graphtransformationsregel. Eine Implikation wird im PDN durch ein Rechteck dargestellt, das den Vertrauens- und Schwellwert enthält. Zusätzlich kann sie optional die Implikationsregel in der Notation der Patternspezifikation enthalten. Aus Platzgründen wird hierauf jedoch in der Regel verzichtet.

Sowohl einfache Prädikate als auch Axiome können durch gerichtete Implikationskanten mit nachfolgenden Implikationen verbunden sein. Die Richtung der Implikationskanten wird durch eine offene Pfeilspitze angegeben. Der Vorbereich einer Implikation enthält die Prädikate zu allen Pattern, die in der zugehörigen Musterspezifikation als Teilpattern enthalten sind. Außerdem werden alle ASG-Objekte, die als Trigger markiert sind, durch gleichnamige Axiome im Vorbereich repräsentiert. Ist ein Axiom oder Prädikat im Vorbereich der Implikation in der Spezifikation als Trigger festgelegt, so ist die zugehörige Implikationskante im PDN eine Triggerkante, die fett dargestellt wird. Analog werden Prädikate zu optionalen Elementen der Spezifikation durch gestrichelt dargestellte optionale Kanten mit der Implikation verbunden.
Der Nachbereich einer Implikation besteht aus genau einem Prädikat, das das in der zugehörigen Spezifikation beschriebene Pattern repräsentiert. Umgekehrt hat jedes einfache Prädikat genau eine Implikation in seinem Vorbereich. Axiome haben hingegen keine Vorgängerknoten, da sie Basisfakten repräsentieren.

Neben den Implikationskanten gibt es außerdem noch Vererbungskanten, die in der UML-üblichen Notation die im Musterklassendiagramm definierte Vererbungshierarchie zwischen den Pattern reflektieren.

Abbildung 2.7 zeigt einen Ausschnitt aus einem Pattern Dependency Net. Es ist zu erkennen, daß die Muster eine Abhängigkeitshierarchie bilden, an deren Spitze in diesem Fall das in Abschnitt 2.1 vorgestellte Composite-Pattern steht. Das Pattern wird repräsentiert durch ein gleichnamiges Prädikat, dessen eingehende Implikation die Prädikate Association, MultiDelegation und Generalization im Vorbereich hat. Diese Prädikate entsprechen gerade den in der Spezifikation des Composite-Pattern verwendeten Subpattern. Desweiteren ist die Kante von dem Prädikat MultiDelegation zur Implikation als Trigger markiert. Dies stimmt mit der Markierung des MultiDelegation-Pattern als Trigger in der Patternspezifikation überein. Neben den in Abschnitt 2.1 für das Composite verwendeten Pattern zeigt der Ausschnitt zudem noch Prädikate zu weiteren Pattern, die die Spezifikation der Subpattern des Composite verfeinern bis

auf die Ebene der ASG-Elemente, die hier durch die Axiome UMLMethod, UMLAttr und UMLGeneralization vertreten sind.

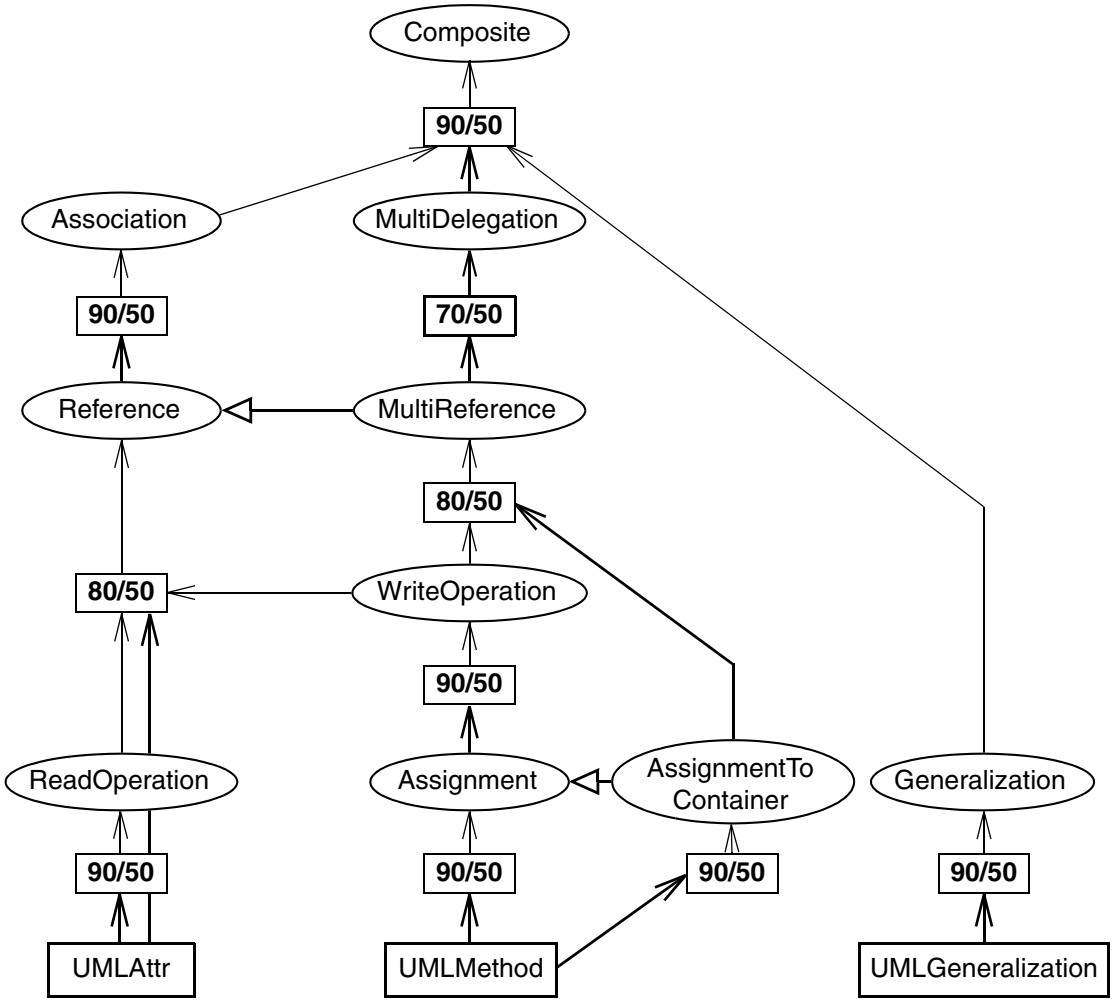


Abbildung 2.7: Ausschnitt eines Pattern Dependency Nets

Anpassung der Syntax

Das in dieser Arbeit als Pattern Dependency Net vorgestellte Abhängigkeitsnetz ist in [Wen01] ursprünglich als Generic Fuzzy Reasoning Net (GFRN) bezeichnet worden. Diese Bezeichnung ist aus einem anderen Projekt namens Varlet [Jah99] übernommen worden, in dem das GFRN ursprünglich als Regelnetz für die Rekonstruktion relationaler Datenbankschemata eingeführt worden ist.
Das GFRN, wie es in [Jah99] beschrieben ist, enthält einige Elemente, die sich im praktischen Einsatz bei der Design Pattern Erkennung als nicht erforderlich oder unpraktisch erwiesen haben. Da diese Elemente sowohl die Musterspezifikation als auch den Erkennungsprozess unnötig komplizieren, sind sie mittlerweile aus dem Netz entfernt worden. Das resultierende Netz stellt durch diese Einschränkungen somit eine spezielle Variante des ursprünglichen GFRN dar. Um diesen Umstand zu verdeutlichen, ist daher der Name Patten Dependency Net

für das hier verwendete Netz eingeführt worden. Auf die Änderungen des PDN gegenüber dem ursprünglichen GFRN soll im Folgenden kurz eingegangen werden.

Die gravierendste Vereinfachung gegenüber dem ursprünglichen GFRN stellt der Verzicht auf negative Implikationskanten dar. Negative Implikationskanten drücken einen gegenseitigen Ausschluß zwischen den beteiligten Prädikaten aus. Im Kontext des Datenbank-Reengineering sind sie verwendet worden, um die Auswirkungen widersprüchlichen Wissens zu modellieren. Es hat sich bereits im Varlet-Projekt erwiesen, daß solche Ausschlußbedingungen dort in der Praxis nur äußerst selten benötigt worden sind. Darüber hinaus basiert die Spezifikation von Design Pattern auf einem konstruktiven Prinzip durch die Beschreibung von Beispielausprägungen, bei dem, im Gegensatz zu dem in [Jah99] verwendeten eher analytischen Regelprinzip, widersprüchliches Wissen keine relevante Rolle spielt. Dementsprechend sind bisher in der Praxis auch keine Fälle aufgetreten, die die Verwendung negativer Implikationen bei der Mustererkennung erfordert hätten.
Der Verzicht auf negative Implikationskanten hat neben einer Vereinfachung des PDN vor allem Auswirkungen auf das in Abschnitt 2.3 beschriebene Fuzzy Petrinetz. Die Verwendung negativer Implikationskanten hat sich dort in negativen Rückkopplungskanten niedergeschlagen, durch die Kreise im Petrinetz erzeugt werden. Durch den Verzicht auf negative Kanten sind diese Kreise eliminiert und das Petrinetz ist nun kreisfrei und strikt vorwärtsgerichtet. Dies vereinfacht sowohl die Fuzzy Evaluation in Abschnitt 2.3 als auch die in den folgenden Kapiteln vorgestellten Adaptionsansätze.

Wie schon erwähnt können Prädikate im PDN nur eine Implikation in ihrem Vorbereich haben. Die ursprünglichen GFRNs aus [Jah99] lassen hingegen mehrere Implikationen im Vorbereich eines Prädikates zu. Hierdurch ist es möglich, unterschiedliche Schlußregeln für ein Prädikat zu spezifizieren. Mit der Einführung von Vererbung ist in [Wen01] durch die Verwendung von Polymorphie eine weitere Möglichkeit geschaffen worden, unterschiedliche Spezifikationen für ein Pattern zu formulieren. Hierzu müssen lediglich die verschiedenen Varianten im Pattern-Klassendiagramm von einer gemeinsamen Pattern-Klasse abgeleitet werden. Die Verwendung von Vererbung unterstützt damit alle Variationen, die auch mit mehreren Implikationen möglich gewesen sind. Daher ist zur Vereinfachung des PDN und zur Forcierung einer durchgängigen Verwendung objektorientierter Entwurfsparadigmen auf die Verwendung mehrerer Implikationen pro Prädikat zugunsten der Vererbung verzichtet worden.

Formale Beschreibung

Bisher ist der Aufbau des Pattern Dependency Net informal vorgestellt worden. Diese Vorstellung soll nun noch durch eine formale Beschreibung abgeschlossen und durch einige nützliche Relationen ergänzt werden.

Definition 2.1: Pattern Dependency Net

Ein Pattern Dependency Net ist ein Tupel (P, I, E_I, E_G, cf, th), für das gilt:

- $P = P_A \cup P_P$ ist die endliche Menge der Prädikate, aufgeteilt in Axiome und einfache Prädikate.
- I ist die endliche Menge der Implikationen.

- $E_I \subset (P \times I) \cup (I \times P_P)$ ist die Menge der gerichteten Implikationskanten von Prädikaten zu Implikationen und von Implikationen zu einfachen Prädikaten. Es gilt :
  $$\forall p \in P_P \exists i \in I : (i, p) \in E_I$$
  $$\forall i \in I \exists q, r \in P : (q, i) \in E_I \wedge (i, r) \in E_I$$
  $$\forall (i, p) \in E_I : (j, p) \notin E_I \wedge (i, s) \notin E_I \forall j \in I \backslash \{i\}, s \in P_P \backslash \{p\}$$
- $E_G \subset (P \times P)$ ist die Menge der Vererbungskanten. Es gilt für $p_1, p_2 \in P$:
  $$(p_1, p_2) \in E_G \Rightarrow (p_1, q) \notin E_G \forall q \in E_P \backslash \{p_2\}$$
- $cf: I \rightarrow [0, 100]$ ordnet jeder Implikation einen Vertrauenswert (engl. Confidence) zu.
- $th: I \rightarrow [0, 100]$ ordnet jeder Implikation einen Schwellwert (engl. Threshold) zu.

Definition 2.2: Vor- und Nachbereich

Für $i \in I$ und $p \in P$ gilt:

- $pre: P_P \rightarrow I$ mit $pre(p) = j \Leftrightarrow (j, p) \in E_I$ ordnet jedem einfachen Prädikat die eindeutig bestimmte Implikation in seinen Vorbereich zu.
- $pre: I \rightarrow P(P)$ mit $pre(i) = \{q \in P | (q, i) \in E_I\}$ ordnet jeder Implikation ihren Vorbereich zu. Es gilt $pre(i) \neq \emptyset \forall i \in I$.
- $pre_P: P \rightarrow P(P)$ mit $pre_P(p) = \{q \in P | (q, j) \in E_I \wedge j = pre(p)\}$ ordnet jedem Prädikat alle direkten Vorgänger-Prädikate zu.
- $post: P \rightarrow P(I)$ mit $post(p) = \{j \in I | (p, j) \in E_I\}$ ordnet jedem Prädikat seinen Nachbereich zu.
- $post: I \rightarrow P$ mit $post(i) = q \Leftrightarrow (i, q) \in E_I$ ordnet jeder Implikation das eindeutig bestimmte Prädikat in ihrem Nachbereich zu.
- $post_P: P \rightarrow P(P)$ mit $post_P(p) = \{q \in P | (j, q) \in E_I \wedge j \in post(p)\}$ ordnet jedem Prädikat alle direkten Nachfolger-Prädikate zu.

Definition 2.3: Transitiver Abschluß und Kreise

- $pre_P^*: P \rightarrow P(P)$ ist der transitive Abschluß von P bezüglich pre_P mit $pre_P^*(p) = \{q \in P | q \in pre(p) \vee \exists r \in pre_P(p) : q \in pre_P^*(r)\}$. Er enthält alle Prädikate aus dem Teilgraph unterhalb von p.
- $post_P^*: P \rightarrow P(P)$ ist der transitive Abschluß von P bezüglich $post_P$ mit $post_P^*(p) = \{q \in P | q \in post(p) \vee \exists r \in post_P(p) : q \in post_P^*(r)\}$. Er enthält alle Prädikate aus dem Teilgraph oberhalb von p.
- $cycl: P_P \rightarrow P(P_P)$ mit $cycl(p) = \{q \in P_P | q \in post_P^*(p) \wedge p \in post_P^*(q)\}$ ordnet jedem Prädikat p die Menge der Prädikate zu, die mit p auf einem Kreis im PDN liegen.

2.2.2 Der Erkennungsprozess

Mit Hilfe des Pattern Dependency Net kann nun der Erkennungsprozess definiert werden. Grundlage für den Erkennungsprozess sind ein Katalog mit Musterspezifikationen und das daraus generierte PDN. Der Musterkatalog legt fest, nach welchen Mustern zu suchen ist und gibt mit den Graphtransformationsregeln der Musterspezifikationen die Suchkriterien für das Pat-

tern Matching vor. Das PDN enthält alle für den Erkennungsprozess wichtigen Abhängigkeiten zwischen den Spezifikationen und steuert damit die Reihenfolge, in der die Anwendung der Spezifikationen erfolgt.

Ein wichtiges Kriterium für die Gestaltung des Erkennungsprozesses ist die Interaktion mit dem Reverse Engineer. Er ist in der Lage, auf der Basis von Hintergrundwissen und eigenen Analysen False Positives zu identifizieren und die Analyseergebnisse um zusätzliche Informationen zu ergänzen. In der Regel interessiert der Reverse Engineer sich vor allem für komplexere Muster, die weit oben in der Pattern-Hierarchie angesiedelt sind. Häufig treten Fehler bei der Mustererkennung jedoch bereits bei einfacheren Mustern weiter unten in der Hierarchie auf, die erst auffallen, wenn sie zu falschen Ergebnissen bei den für den Reverse Engineer relevanten Mustern führen. Damit diese Fehler schnellstmöglich erkannt und so weitere fehlerhafte Ergebnisse auf Basis falscher Annahmen vermieden werden können, muß der Erkennungsprozess gewährleisten, daß dem Reverse Engineer so früh wie möglich für ihn relevante Zwischenergebnisse zur Verfügung stehen. Nur so ist es möglich, ihn von Anfang an effektiv in den Erkennungsprozess zu integrieren.

Um die Integration des Reverse Engineers zu unterstützen, ist daher ein halbautomatischer, inkrementeller Erkennungsprozess realisiert worden, der schnell zu guten Zwischenergebnissen führt und jederzeit vom Benutzer unterbrochen und nach eventuellen Korrekturen ohne den Verlust von Zwischenergebnissen fortgesetzt werden kann. Änderungen durch den Benutzer, auf die in Abschnitt 2.4 eingegangen wird, gehen dabei unmittelbar in die weiteren Analysen ein.

Die Inferenzmaschine

Der Mustererkennungsprozess wird gesteuert durch die Inferenzmaschine. Diese sucht nach Instanzen der im Musterkatalog spezifizierten Pattern.

Da die Musterspezifikationen bereits als Graphtransformationsregeln vorliegen, ist es naheliegend, die Patternsuche durch Anwendung dieser Regeln auf die ASG-Darstellung des eingeparsten Quelltextes zu realisieren. Jede erfolgreiche Anwendung einer Regel resultiert dabei in der Erzeugung einer Annotation im ASG, die dem Benutzer im Klassendiagramm des zu analysierenden Systems angezeigt wird. Bei Graphtransformationssystemen wie Progres[Zün96] oder AGG[AGG] kann die Anwendung von Transformationsregeln auf große Graphen zu Problemen mit der Skalierbarkeit führen. Um diese Probleme zu vermeiden, wird in dem in [Wen01] verwendeten Graphtransformationssystem für jede Regelanwendung ein Kontext, typischerweise ein Element des ASG, angegeben. Dieser Kontext muß von der Inferenzmaschine zur Verfügung gestellt werden.

Für eine erfolgreiche Anwendung der Graphtransformationsregeln ist zudem die Reihenfolge entscheidend. Diese wird von der Inferenzmaschine mit Hilfe des PDN und der darin festgehaltenen Abhängigkeiten bestimmt. Die hierarchische Struktur des PDN legt ein deduktives Vorgehen nahe. Tatsächlich handelt es sich bei der Design Pattern Erkennung um ein deduktives Analyseproblem, bei dem sukzessive von kleineren Subpattern auf komplexere Pattern

geschlossen wird, um schließlich ein vollständiges Bild aller Musterausprägungen zu erhalten. Ein rein deduktives, auch als Bottom-Up bezeichnetes Vorgehen ist trotzdem für den Erkennungsprozess nicht geeignet. Denn bei einem klassischen deduktiven Ansatz werden zunächst alle Analysen einer Hierarchieebene abgeschlossen, bevor Analysen der nächsten Ebene beginnen. Daher stehen die für den Reverse Engineer relevanten Ergebnisse erst zu Ende des Analyseprozesses zur Verfügung. Aus diesem Grund verwendet die Inferenzmaschine eine Kombination aus einer datengetriebenen Bottom-Up-Strategie und einer zielgetriebenen Top-Down-Strategie, die ein schnelles Erkennen komplexerer Muster sicherstellt[NSW+02].

Zunächst muß hierzu zu Beginn des Erkennungsprozesses jedem Prädikat des PDN ein Rang gemäß seiner Position in der Abhängigkeitshierarchie zugewiesen werden. Anhand dieses Ranges wird in dem nachfolgenden Inferenzalgorithmus die Reihenfolge der Regelanwendungen bestimmt. Axiome erhalten einen Rang von Null. Prädikate, die von anderen Prädikaten abhängen erhalten entsprechend ihrer topologischen Ordnung einen höheren Rang als die Prädikate aus ihrem Vorbereich. An einem Zyklus beteiligte Prädikate erhalten alle denselben Rang und werden zusammen mit ihren eingehenden Implikationen als rekursiv gekennzeichnet.
Formal gilt:

Definition 2.4: Rang

a) Der Rang eines Prädikates p ist gegeben durch die Funktion $rang: P \rightarrow \mathbb{N}$ mit

$$rang(p) = 0 \text{ falls } p \in P_A$$
$$rang(p) > rang(q)\ \forall q \in pre(p) \setminus cycl(p)$$
$$rang(p) = rang(q)\ \forall q \in cycl(p)$$

b) Der Rang einer Implikation i ist gegeben durch die Funktion $rang: I \rightarrow \mathbb{N}$ mit

$$rang(i) = rang(post(i))$$

Diese Definition läßt verschiedene Möglichkeiten zu, die Ränge nicht voneinander abhängiger Prädikate festzulegen und so etwa die Erkennungsreihenfolge zu optimieren. Im Folgenden wird soweit nicht anders angegeben stets der natürliche Rang verwendet.

Definition 2.5: Natürlicher Rang

Der natürliche Rang $rang_0$ eines Prädikates p ist gegeben durch

$$rang_0(p) = Max\{r'(q) | q \in cycl(p) \cup \{p\}\} \text{ mit}$$
$$r'(p) = 0 \text{ für } p \in P_A \text{ und}$$
$$r'(p) = Max\{rang_0(q) | q \in pre_P(p) \setminus cycl(p)\} + 1 \text{ für } p \in P_P$$

Mit Hilfe des Ranges kann nun der Erkennungsalgorithmus beschrieben werden. Abbildung 2.8 zeigt eine Momentaufnahme des Algorithmus. Das schwarze Oval zeigt die Annotation einer Generalisierung, die bereits in der Bottom-Up-Analyse erkannt worden ist. Die grauen Ovale symbolisieren noch nicht abgeschlossene Top-Down-Analysen der entsprechenden Pattern. Die Zahlen oberhalb der Ovale sind die Rangnummern der Prädikate, die sich nach Definition 2.5 für das PDN aus Abbildung 2.7 ergeben. Die gerichteten Kanten verdeutlichen die Reihenfolge des Analyseprozesses und die Kantenbeschriftungen bezeichnen die Kontextobjekte, die für die Anwendung der Graphtransformationsregeln von der Inferenzmaschine bereitgestellt werden.

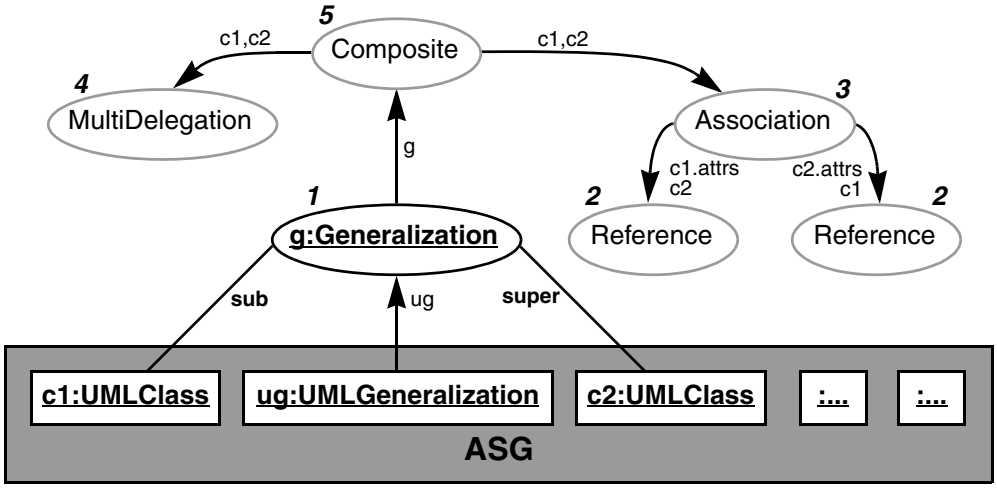


Abbildung 2.8: Beispielausführung des Erkennungsprozesses

Bottom-Up Strategie

Der Analyseprozess startet im Bottom-Up Modus. Die Informationen über die im Bottom-Up Modus durchzuführenden Analysen werden in einer Liste abgelegt, die die Struktur einer Prioritätswarteschlange hat. Die Einträge dieser Bottom-Up Queue sind Tupel bestehend aus einer Implikation und einem Kontext-Objekt aus dem abstrakten Syntaxgraph. Die Implikation legt die auszuwertende Patternspezifikation fest und das Kontext-Objekt dient als Ansatzpunkt für die Anwendung der zugehörigen Graphtransformationsregel. Die Einträge in der Bottom-Up Queue sind absteigend nach dem Rang der Implikationen sortiert.

Zur Initialisierung der Bottom-Up Queue werden zunächst alle ASG-Objekte ermittelt, die Axiomen im PDN entsprechen. Für jedes dieser Objekte werden im PDN die Implikationen aus seinem Nachbereich bestimmt, mit denen es über eine Triggerkante verbunden ist. Für jede gefundene Implikation wird ein Tupel bestehend aus der Implikation und dem ASG-Objekt als Kontext in die Bottom-Up Queue eingetragen.

Nach der Initialisierung wird die Bottom-Up Queue sukzessive abgearbeitet. Dazu wird jeweils das erste (und damit höchstrangige) Element zur Analyse aus der Liste entnommen. In dem Beispiel aus Abbildung 2.8 ist dies die Implikation zum Generalization-Pattern mit dem Vererbungsobjekt ug als Kontext. Nun wird versucht, die in der zugehörigen Patternspezifikation definierte Graphersetzungsregel auf den Kontext anzuwenden. Gelingt dies wie in dem Beispiel für die Generalization g, so ist das Muster erfolgreich erkannt worden und es wird eine entsprechende Annotation im Syntaxgraph erzeugt. Anschließend werden die Implikationen, die gemäß PDN von dem neuen Pattern getriggert werden, mit dem Pattern als Kontext in die Bottom-Up Queue eingetragen. In dem Beispiel ist dies die Implikation zum Composite Pattern[1] mit dem soeben gefundenen Generalization Pattern als Kontext. Das neu erzeugte Tupel ist nun aufgrund seines Ranges das erste in der Bottom-Up Queue und wird als nächstes abgearbeitet.

1. Hier ist zur einfacheren Darstellung die Implikationskante zwischen Generalization und Composite als Triggerkante angenommen worden. Abbildung 2.7 zeigt dagegen die in der Praxis sinnvollere Alternative mit MultiDelegation als Trigger.

Es ist also zu erkennen, daß die Priorisierung hochrangiger Implikationen schnell zur Analyse komplexerer Pattern führt.

Um das Composite Pattern im Beispiel erfolgreich erkennen zu können, ist neben der bereits erkannten Generalization noch eine MultiDelegation und eine Association erforderlich. Es wird daher nun zunächst versucht, diese noch fehlenden Muster herzuleiten. Für diese zielgerichtete Herleitung wechselt der Inferenzalgorithmus nun in den Top-Down Modus.

Top-Down Strategie

Kann eine Implikation im Bottom-Up Modus nicht schließen, weil für die Elemente aus ihrem Vorbereich noch keine Annotationen im abstrakten Syntaxgraph existieren, so wird das Tupel aus Implikation und Kontext in die Top-Down Queue eingetragen. Hierbei handelt es sich ebenfalls um eine Prioritätswarteschlange, die jedoch nach aufsteigendem Rang sortiert ist, so daß Tupel mit niedrigem Rang bevorzugt werden. Im Top-Down Modus wird dann versucht, die unerfüllten Vorbedingungen der Implikation zielgetrieben herzuleiten. Diese Herleitung kann im Gegensatz zu anderen Top-Down Verfahren effizient ausgeführt werden, weil durch den vorgegebenen Kontext der zu analysierende Suchraum auf einige wenige Elemente beschränkt ist.

Bei der Abarbeitung der Top-Down Queue wird wiederum sukzessive das erste Element in der Queue betrachtet. Kann die zugehörige Implikation nicht schließen, weil Bedingungen aus ihrem Vorbereich nicht erfüllt sind, werden die Implikationen zu den fehlenden Pattern mit einem entsprechenden Kontext in die Top-Down Queue eingetragen. Der erforderliche Kontext kann aus dem Kontext der fehlgeschlagenen Implikation und der zugehörigen Patternspezifikation ermittelt werden. Hierfür kann es unter Umständen mehrere Möglichkeiten geben. In diesem Fall wird für jeden möglichen Kontext ein Tupel in die Liste eingetragen. Im Beispiel ergeben sich sowohl für die MultiDelegation als auch für die Association jeweils die UMLClass-Objekte c1 und c2 als mögliche Kontexte.

Im Gegensatz zum Bottom-Up Modus wird ein Tupel aus der Top-Down Queue erst entfernt, wenn die abgelegte Implikation entweder erfolgreich schließen konnte oder die Suche nach einem fehlenden Muster aus dem Vorbereich der Implikation für alle ermittelten Kontexte fehlgeschlagen ist. Durch die Priorisierung niederrangiger Analysen ist gewährleistet, daß die Suche so schnell wie möglich auf Basisfakten des ASG zurückgeführt wird. Dadurch wird ein schnelles Fehlschlagen nicht erfüllbarer Implikationen erreicht. Ist ein Pattern erfolgreich erkannt worden, so werden die dadurch getriggerten Implikationen für die spätere Auswertung in die Bottom-Up Queue eingetragen.

Durch Composite-artige Strukturen im PDN kann es zu zyklischen Abhängigkeiten zwischen Prädikaten entlang von Vererbungshierarchien kommen. Die Untersuchung solcher zyklisch voneinander abhängigen Prädikaten kann im Top-Down Modus zu einer Endlos-Rekursion führen. Um dies zu verhindern, wird im Top-Down Modus ein Stack anstelle der Top-Down Queue verwendet, sobald eine als rekursiv gekennzeichnete Implikation eingetragen werden soll. Erst nachdem der Stack vollständig abgearbeitet ist, wird die Abarbeitung der Top-Down Queue fortgesetzt.

Der gesamte Erkennungsprozess ist abgeschlossen, sobald die Bottom-Up Queue leer ist. Zu diesem Zeitpunkt sind alle ASG-Objekte vollständig analysiert worden und für alle Übereinstimmungen mit den Patternspezifikationen existieren Annotationen im abstrakten Syntaxgraph, die in den zugehörigen Klassendiagrammen angezeigt werden.

2.3 Die Fuzzy Evaluation

Nachdem die Mustererkennung angehalten hat, folgt als nächstes die Bewertung der Analyseergebnisse. Da die Musterspezifikationen, die die Basis der Analysen bilden, sehr allgemein formuliert sind und False Positives unter den Ergebnissen nicht ausgeschlossen werden können, wird jeder Annotation zu einer Patterninstanz ein Fuzzywert zwischen 0 und 100 zugeordnet. Dieser Fuzzywert soll dem Reverse Engineer als Anhaltspunkt für die Verlässlichkeit des Ergebnisses dienen und ihn bei der Auswahl der Patterninstanzen unterstützen, für die eine weitere Analyse sinnvoll erscheint. Die Berechnung dieser Fuzzy-Werte erfolgt durch ein Fuzzy Petrinetz[Wen01, Jah99], das während des Erkennungsprozesses im Hintergrund mitaufgebaut wird.

2.3.1 Struktur des Fuzzy Petrinetzes

Das Fuzzy Petrinetz (FPN) ist ein klassisches Stellen-Transitions-Netz[Jah99]. Die Stellen im Fuzzy Petrinetz entsprechen Analyseergebnissen, während die Transitionen Implikationsregeln repräsentieren.

Für jede gefundene Patterninstanz wird während des Erkennungsprozesses neben einer Annotation im ASG zusätzlich eine Stelle im FPN erzeugt, so daß jede Annotation eine korrespondierende Stelle im FPN hat, die deren Fuzzy-Wert bestimmt. Ebenso wird jedes ASG-Objekt, das einem Axiom im PDN entspricht, von einer Stelle repräsentiert. Die Implikationen, deren Graphtransformationsregeln die Annotationen erzeugt haben, besitzen im FPN entsprechende Transitionen. Der Nachbereich einer Transition besteht dabei aus der Stelle, die die zugehörige Annotation repräsentiert. Der Vorbereich wird gebildet von den Stellen, die den Annotationen zu Teilpattern des erkannten Musters entsprechen. Stellen zu Axiomen des PDN haben entsprechend keine Transitionen in ihrem Vorbereich.

Durch die in Abschnitt 2.2 beschriebenen Anpassungen ist auch im FPN sichergestellt, daß jede Transition genau eine Stelle in ihrem Nachbereich hat. Umgekehrt hat auch jede Stelle, die kein Axiom repräsentiert, genau eine Transition in ihrem Vorbereich. Da Zyklen im PDN zudem im Erkennungsprozess eine endliche Rekursion bewirken, die im FPN keine Kreise erzeugt, und auf negative Implikationskanten, die Rückkopplungskreise im FPN bewirken können, verzichtet worden ist, ist das resultierende FPN zyklenfrei und streng vorwärtsgerichtet.

Jede Stelle im FPN besitzt ein Fuzzy Belief Marking, bei dem es sich um eine natürliche Zahl zwischen 0 und 100 handelt. Das Fuzzy Belief Marking (FBM) entspricht nach der Auswertung des FPN dem Fuzzy-Wert der zugehörigen Annotation. Die Berechnung der Fuzzy Belief Mar-

kings erfolgt auf Basis der Vertrauens- und Schwellwerte der Patternspezifikationen. Diese Werte sind im FPN den Transitionen zugeordnet und bestimmen das Schaltverhalten des FPN wie im nächsten Abschnitt erläutert. Der errechnete FBM-Wert einer Stelle kann zudem vom Benutzer nach eigenen Einschätzungen korrigiert werden, wie in Abschnitt 2.4 beschrieben wird. Der errechnete Wert wird hierbei nicht einfach ersetzt. Stattdessen wird der benutzerdefinierte Wert der Stelle zusätzlich zugewiesen und überdeckt den errechneten Wert. So ist zum einen gewährleistet, daß der korrigierte Wert in späteren Berechnungen des FPN nicht wieder durch den errechneten Wert überschrieben wird. Zum anderen ist durch den Erhalt des errechneten Wertes aber vor allem ein Vergleich zwischen dem benutzerdefinierten Wert als Sollgröße und dem errechneten Istwert möglich, der entscheidend für die in Kapitel 3 und 4 vorgestellten Adaptionsalgorithmen ist.

Abbildung 2.9 zeigt ein Beispiel-FPN zu dem PDN aus Abbildung 2.7. Es ist zu erkennen, daß die Struktur des FPN der des PDN stark ähnelt. Da die Stellen des FPN konkrete Instanzen der im PDN enthaltenen Pattern widerspiegeln und die Transitionen für erfolgreiche Anwendungen der Implikationen stehen, kann das FPN als Instanzgraph mit dem PDN als Typgraph aufgefasst werden.

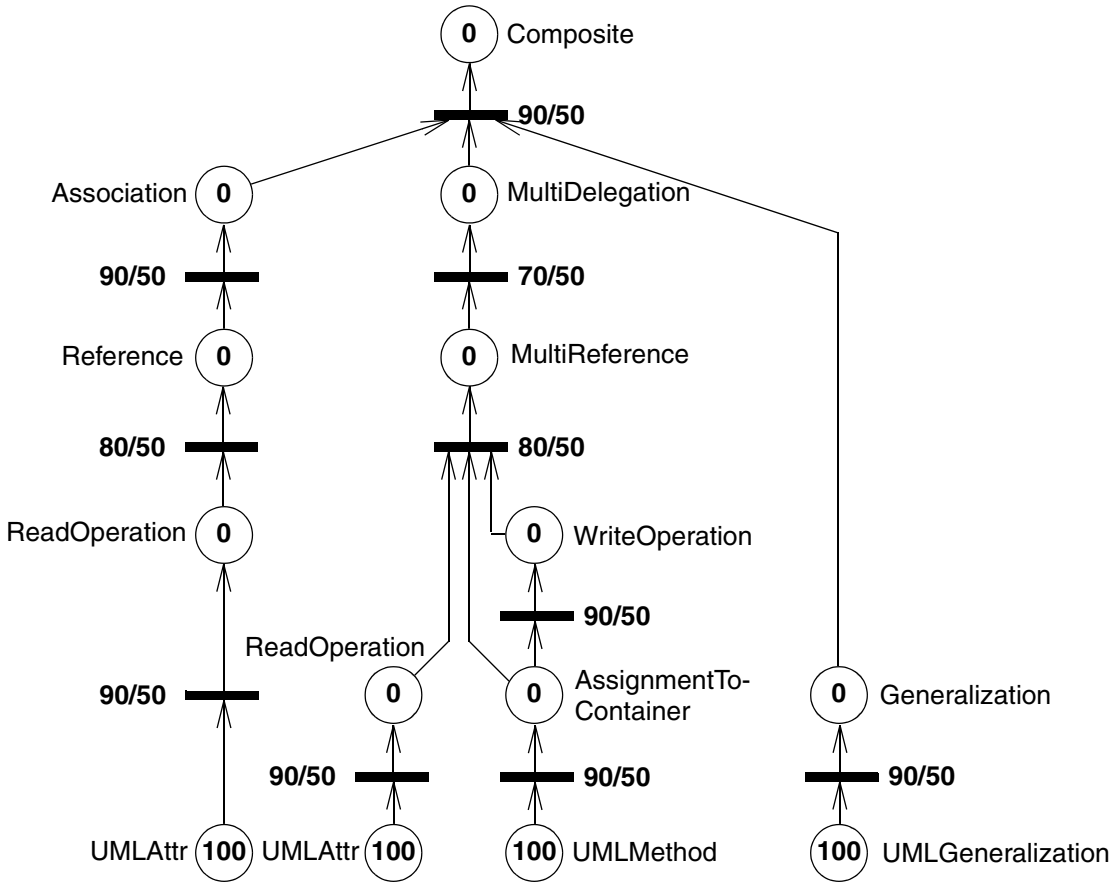


Abbildung 2.9: Ausschnitt eines Fuzzy Petrinetzes

Formal ist ein Fuzzy Petrinetz wie folgt definiert.

Definition 2.6: Fuzzy Petrinetz

Ein Fuzzy Petrinetz ist ein Tupel (S, T, F, cf, th, fbm) mit:

- S ist eine endliche Menge von Stellen
- T ist eine endliche Menge von Transitionen mit $T \cap S = \varnothing$
- $F \subset (S \times T) \cup (T \times S)$ ist die Kantenmenge des FPN und wird als Flußrelation bezeichnet
- $cf: T \rightarrow [0, 100]$ ist die Konfidenzfunktion, die jeder Transition einen Vertrauenswert zuweist
- $th: T \rightarrow [0, 100]$ ordnet jeder Transition einen Schwellwert zu.
- $fbm: S \rightarrow [0, 100]$ ist die Markierungsfunktion, die jeder Stelle ein Fuzzy Belief Marking zuordnet. Hierbei wird zwischen dem errechneten und dem benutzerdefinierten FBM unterschieden. Es gilt:
  $fbm_c: S \rightarrow [0, 100]$ ordnet jeder Stelle das errechnete FBM zu.
  $fbm_u: S \rightarrow [0, 100] \cup \{-1\}$ ordnet jeder Stelle ein benutzerdefiniertes FBM oder den Sonderwert -1 zu, wobei -1 die Abwesenheit eines benutzerdefinierten Wertes symbolisiert.
  Damit ergibt sich für eine Stelle s:

$$fbm(s) = \begin{cases} fbm_u(s) & \text{falls } fbm_u(s) \neq -1 \\ fbm_c(s) & \text{sonst} \end{cases}$$

- $typ_s: S \rightarrow P$ und $typ_t: T \rightarrow I$ ordnet jeder Stelle und jeder Transition die zugehörigen Elemente des PDN zu
- $inst_p: P \rightarrow P(S)$ und $inst_i: I \rightarrow P(T)$ ordnet jedem Prädikat und jeder Implikation die zugehörigen Instanzen im FPN zu

Definition 2.2 und Definition 2.3 zu Vor- und Nachbereichen gelten entsprechend, wobei Prädikate durch Stellen und Implikationen durch Transitionen zu ersetzen sind.

2.3.2 Auswertung des Fuzzy Petrinetzes

Zu Beginn des Auswertungsprozesses sind alle Stellen des Fuzzy Petrinetzes, die Axiome des PDN repräsentieren, mit einem Fuzzy Belief Marking von 100 markiert. Dies drückt die absolute Sicherheit der Axiome aus, die ja für Fakten im ASG stehen. Alle anderen Stellen sind mit einem FBM von 0 initialisiert.

Die Auswertung des FPN erfolgt schrittweise. Da das FPN zyklenfrei ist, muß anders als in [Wen01] beschrieben das FBM für jede Stelle bei geeigneter Berechnungsreihenfolge nur einmal aktualisiert werden. Dazu werden zunächst die FBMs für alle Stellen angepasst, die nur Axiom-Stellen in ihrem Vorbereich haben. Anschließend können die Stellen aktualisiert werden, deren Vorbereich nur aus Axiom-Stellen und den soeben angepassten Stellen besteht. Dies kann sukzessive fortgesetzt werden, bis das gesamte FPN ausgewertet ist.

Die Aktualisierung des Fuzzy Belief Markings einer Stelle geschieht gemäß [Wen01] in zwei Phasen. In der ersten Phase wird für die Transition im Vorbereich der Stelle ein Fuzzy Truth Token (FTT) bestimmt.

Definition 2.7: Fuzzy Truth Token

Das Fuzzy Truth Token zu einer Transition t wird berechnet durch $ftt:T \rightarrow [0, 100]$ mit

$$\tilde{ftt}(t) = Min\{fbm(s) | s \in pre(t)\} \text{ und}$$

$$ftt(t) = \begin{cases} Min\{cf(t), \tilde{ftt}(t)\} & \text{falls } \tilde{ftt}(t) \geq th(t) \\ 0 & \text{sonst} \end{cases}$$

Zur Berechnung des FTT wird also zunächst das Minimum über die Fuzzy Belief Markings aller Stellen im Vorbereich der Transition gebildet. Ist dieser Wert größer als der Schwellwert der Transition, so ist die Transition aktiv und das FTT ergibt sich aus dem Minimum des berechneten Wertes und des Schwellwertes der Transition. Die Werte aus dem Vorbereich einer Transition und dem Vertrauenswert, die das Minimum des FTT prägen, werden als dominante Werte bezeichnet. Sie haben eine wichtige Bedeutung für die in Kapitel 3 und Kapitel 4 vorgestellten Lernverfahren.

In der zweiten Phase schalten alle aktiven Transitionen und die errechneten FBMs nehmen die Werte der Fuzzy Truth Tokens aus dem Vorbereich der Stellen an[1]. Ist für eine Stelle ein Fuzzy-Wert vom Benutzer vorgegeben, so wird dieser statt des errechneten Wertes als FBM angenommen und für die weitere Auswertung des FPN verwendet.

Abbildung 2.10 zeigt das Fuzzy Petrinetz aus Abbildung 2.9 nach der vollständigen Auswertung. Sobald die Auswertung des FPN abgeschlossen ist, werden die neu ermittelten Werte an die Annotationen weitergegeben und mit diesen zusammen in den jeweiligen Klassendiagrammen angezeigt. Anschließend ist der Inferenzprozeß abgeschlossen und der Reverse Engineer kann die gefundenen Pattern analysieren und die Fuzzy-Bewertung der Annotationen seinen Schätzungen entsprechend anpassen. Dies wird im folgenden Abschnitt beschrieben.

2.4 Benutzerinteraktion

Der vorangegangene Abschnitt hat gezeigt, wie die Qualität ungenauer Analyseergebnisse mit Hilfe eines Fuzzy Petrinetzes automatisch bewertet werden kann. Eine solche Bewertung kann jedoch auch stets nur eine Abschätzung des tatsächlichen Ergebnisses darstellen und ist daher nur in Verbindung mit einer Feedback-Möglichkeit des Benutzers an das Erkennungssystem sinnvoll. Denn letztlich kann nur der Benutzer eine endgültige Bewertung der Analyseergebnisse vornehmen und entscheiden, ob die Abschätzung durch das FPN angepaßt werden muß oder unter Umständen sogar ein Fehler in der Patternspezifikation vorliegt. Aus diesem Grund

1. Laut [Wen01] erfolgt hier eine Maximumsbildung über die FTTs aller eingehenden Transitionen. Dies kann entfallen, da jede Stelle höchstens eine eingehende Transition besitzt.

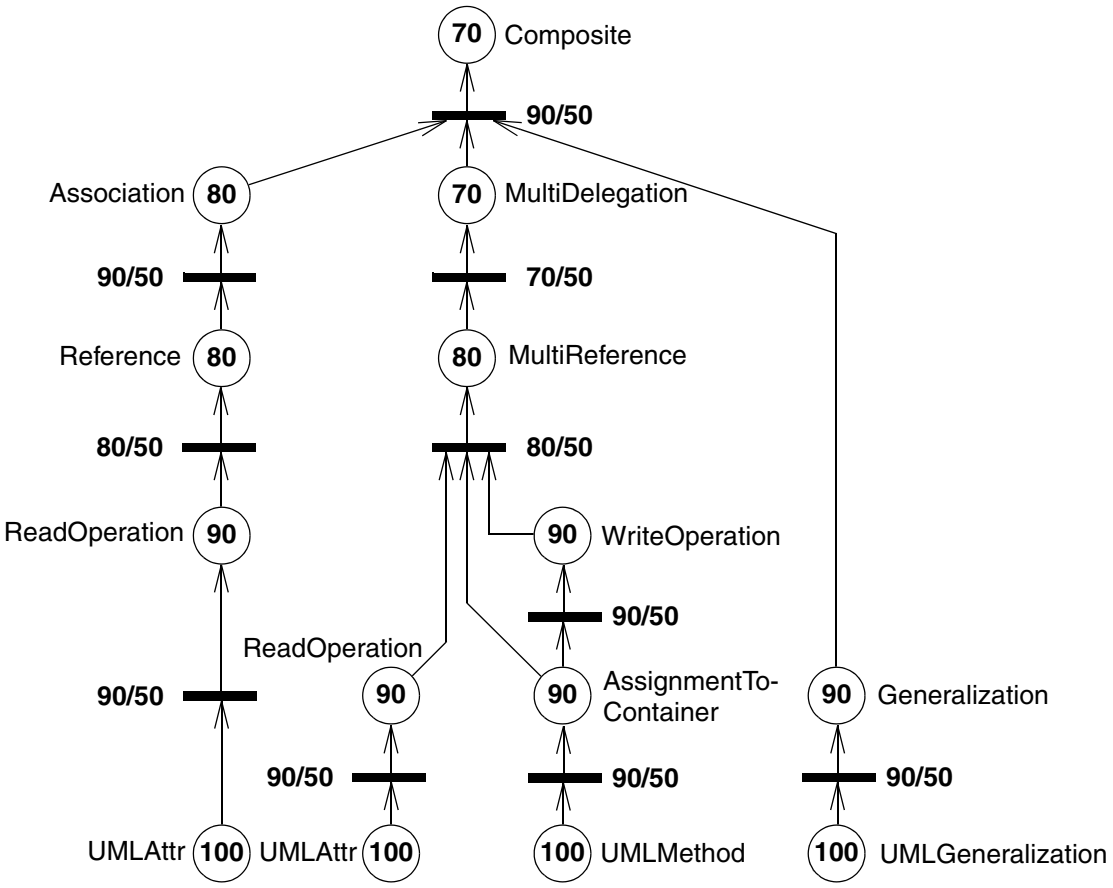


Abbildung 2.10: Fuzzy Petrinetz nach der Auswertung

ist der Erkennungsprozess so gestaltet, daß der Benutzer mit seinem Hintergrundwissen und seinen dem System überlegenen analytischen Fähigkeiten von Anfang an in den Mustererkennungsprozess integriert ist. Ein Überblick über den Ablauf des Analyseprozesses und die Interaktionsmöglichkeiten des Benutzers ist in Abbildung 2.11 in Form eines Statecharts dargestellt.

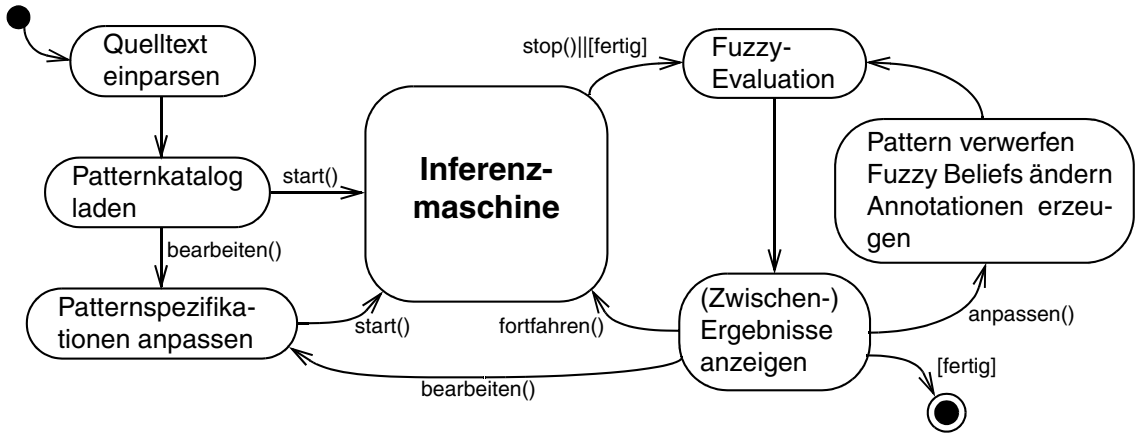


Abbildung 2.11: Inferenzprozess Statechart

Nachdem der Quelltext des zu analysierenden Systemes eingeparst und ein geeigneter Patternkatalog ausgewählt worden ist, hat der Benutzer vor dem Start des Inferenzprozesses die Möglichkeit, letzte Änderungen an den Patternspezifikationen vorzunehmen. So kann etwa a priori bekannten Besonderheiten in dem zu analysierenden System Rechnung getragen werden. Nach dem Starten der Inferenzmaschine gewährleistet der in Abschnitt 2.2 vorgestellte Erkennungsalgorithmus eine frühzeitige Erkennung komplexer Muster, die voraussichtlich für den Benutzer von Interesse sind. Wird ein für den Benutzer interessantes Muster gefunden, so kann er dieses unter Zuhilfenahme des Quelltextes und seines Hintergrundwissens sofort analysieren und so auf eventuelle Fehler frühzeitig reagieren, bevor weitere Analysen auf möglicherweise falschen Grundlagen erfolgen.

Um dem Benutzer die nötige Zeit für seine Analysen zu geben, kann er den automatischen Erkennungsprozess jederzeit unterbrechen und später fortsetzen, ohne dabei Zwischenergebnisse zu verlieren. Dies ist möglich, da alle bis dahin gefundenen Muster Zwischenergebnisse darstellen, die durch entsprechende Annotationen im ASG festgehalten sind und durch die weitere automatische Analyse nicht mehr verändert werden. Auch die Daten einer unvollständigen Top-Down-Analyse gehen bei einer Unterbrechung nicht verloren, da alle für die Fortsetzung relevanten Informationen in den entsprechenden Warteschlangen festgehalten sind.

Ist der Erkennungsprozess abgeschlossen oder wird er vom Benutzer unterbrochen, so erfolgt automatisch die Fuzzy-Bewertung der bis dahin gefundenen Ergebnisse wie in Abschnitt 2.3 beschrieben. Anhand der berechneten Fuzzy-Werte kann der Benutzer nun die Muster für seine weiteren Untersuchungen auswählen. Stellt er bei diesen Untersuchungen fest, daß ein Muster falsch erkannt oder bewertet worden ist, so kann er dies korrigieren und damit die Erkennungsgenauigkeit der darauf aufbauenden Analysen verbessern. Bei der Fuzzy-Bewertung erhalten Muster in der Regel einen Fuzzy-Wert unterhalb des Maximalwertes von 100. Ist sich der Benutzer sicher, daß ein Muster korrekt erkannt worden ist, so kann er diesen Wert auf 100 erhöhen. Stellt er hingegen fest, daß es sich bei dem gefundenen Muster um ein False Positive handelt, so muß er das Ergebnis zurückweisen. Hierzu setzt er den Fuzzy-Wert des Musters auf 0. Ist sich der Benutzer hinsichtlich der Korrektheit des Musters nicht sicher, kann er natürlich auch entsprechende Werte zwischen 0 und 100 vergeben.

Die Anpassung der Fuzzy-Werte durch den Benutzer ist für diese Arbeit von vorrangigem Interesse, da sie die Grundlage für die Ausnivellierung der Fuzzy-Parameter darstellt, die Gegenstand der weiteren Arbeit ist. Daneben hat der Benutzer jedoch auch noch die Möglichkeit, selbst neue Annotationen zu erzeugen, falls ein Muster nicht von dem Erkennungsmechanismus gefunden worden ist. Auch diese Interaktion ist mit dem Algorithmus aus Abschnitt 2.2 einfach zu realisieren. Es muß lediglich für die neue Annotation eine Stelle im FPN erzeugt und die Bottom-Up Queue um die von dem Muster getriggerten Implikationen ergänzt werden. Werden häufig Muster nicht erkannt oder gibt es andere Probleme mit den Analyseergebnissen, so kann der Benutzer an dieser Stelle auch eine erneute Kontrolle der Musterspezifikationen vornehmen und diese gegebenenfalls anpassen. Da sich hierdurch die Grundlage der Mustererkennung ändert, kann in diesem Fall jedoch der Erkennungsprozess nicht wieder aufgenommen werden, sondern muß neu gestartet werden.

Nach jeder Anpassung durch den Benutzer wird das Fuzzy Perinetz neu ausgewertet, wodurch dem Benutzer unmittelbar die Konsequenzen seiner Änderungen auf andere Muster angezeigt werden. So kann er auf dieser Grundlage eventuell weitere Anpassungen vornehmen. Ist er mit den Zwischenergebnissen zufrieden, so kann er die Mustererkennung fortsetzen. Nachdem der Erkennungsprozess abgeschlossen ist, hat der Benutzer dann eine vollständige Übersicht über die gefundenen Muster in dem analysierten System.

KAPITEL 3

Adaption von Wissen

Um im nächsten Kapitel ein auf die in Kapitel 2 beschriebene Fuzzy-Bewertung zugeschnittenes Lernverfahren entwickeln zu können, ist es erforderlich, zunächst die in der Einführung nur informal beschriebenen Lernziele formal zu präzisieren. Dies geschieht in Abschnitt 3.2. Die hierzu benötigten Grundbegriffe und Modelle maschinellen Lernens, die auch im weiteren Verlauf der Arbeit benötigt werden, werden zuvor in Abschnitt 3.1 eingeführt. Anschließend werden in Abschnitt 3.3 einige existierende Ansätze betrachtet, aus deren Problemen mit dem hier betrachteten Modell Rückschlüsse für das zu entwickelnde Lernverfahren gezogen werden können.

3.1 Maschinelles Lernen

An dieser Stelle soll zunächst geklärt werden, was unter (maschinellem) Lernen zu verstehen ist. Außerdem werden weitere Begriffe aus dem Bereich des Lernens erläutert, um eine Basis für die weiteren Ausführungen zu schaffen.

Lernen

Um den Begriff des maschinellen Lernens zu erläutern, ist es hilfreich, zunächst generell den Begriff des Lernens zu betrachten. Allgemein beschreibt Lernen die Vermehrung des eigenen Wissens. Diese Definition ist jedoch zu unspezifisch, um sie in einen algorithmischen Ansatz zu übertragen. Eine spezifischere Definition von Lernen beschreibt das Erlernen von Verhaltensweisen:

> „*Lernen beschreibt das Anpassen von Verhalten auf der Basis gesammelter Erfahrungen*“

Hieraus läßt sich unmittelbar eine Definition von maschinellem Lernen ableiten. Verhalten wird in diesem Kontext im Allgemeinen durch Algorithmen beschrieben. Die Erfahrungen, die zur Anpassung eines Algorithmus herangezogen werden können, sind Datensätze aus der Problemdomäne des betrachteten Algorithmus. Cherkassky und Mulier [CM98] definieren daher maschinelles Lernen wie folgt:

> „*Maschinelles Lernen bezeichnet den Prozess zur Verbesserung eines Algorithmus auf der Basis seiner Eingabedaten.*“

Weiterhin unterteilen sie maschinelle Lernansätze in drei Problembereiche: Klassifikationsprobleme, Regressionsprobleme und die Schätzung von Verteilungsdichten.

Klassifikationsprobleme befassen sich mit der Einordnung von Datensätzen in Gruppen mit gleichen oder ähnlichen Eigenschaften. Das Ziel von Lernverfahren dieses Problembereiches ist es, die hierfür erforderlichen Klassifizikationsgrenzen zu erlernen.

Ansätze zum Schätzen von Verteilungsdichten dienen zur Vorhersage von Ereignissen in einem kontinuierlichen Ereignisraum anhand der Verteilung bisher beobachteter Ereignisse.

Regression

(Parametrische) Regression bezeichnet den Prozess zur Abschätzung der Parameter einer qualitativ bekannten Funktion auf der Basis rauschbehafteter Trainingsdaten. In Regressionsproblemen kann die Ausgabe des Systems beschrieben werden als Vektor **y** von Zufallswerten, der interpretiert werden kann als die Summe einer deterministischen Funktion f mit unbekanntem Parametervektor ω_0 und einem zufälligen, mittelwertfreien Fehler:

$$\mathbf{y} = f(\mathbf{x}, \omega_0) + \varepsilon \qquad \textbf{(3.1)}$$

Lernen eines Regressionsproblem entspricht somit dem Finden von Parametern ω^*, für die die Abweichung zwischen $f(\mathbf{x}, \omega^*)$ und **y** für die gegebenen Trainingsdaten der Form (**x**, **y**) minimal ist. Da die Menge der Trainingsdaten nicht notwendigerweise alle Eigenschaften der Systemfunktion erkennen läßt, muß dabei auch bei einem optimalen Lernerfolg nicht unbedingt $\omega^*=\omega_0$ gelten. Lernmethoden für die Lösung des Regressionsproblems werden in Abschnitt 3.3 und Abschnitt 3.4 behandelt.

Supervised und Unsupervised Learning

Anhand der zum Lernen herangezogenen Daten unterscheidet man weiterhin zwischen *unsupervised learning* (nicht überwachtem Lernen) und *supervised learning* (überwachtem Lernen). Unsupervised learning bezeichnet Lernverfahren, die ohne eine zu erlernende Sollgröße wie etwa die korrekte Ausgabe eines Systems lernen. Hierbei handelt es sich zumeist um sogenannte Clustering-Algorithmen zur Berechnung von Wahrscheinlichkeitsdichten. Ein weiteres Beispiel für unsupervised learning ist der in Anhang B skizzierte Algorithmus zur Optimierung der Inferenzreihenfolge.

Supervised learning ist die Abschätzung eines nicht näher bekannten Zusammenhanges zwischen Ein- und Ausgabewerten eines Systems anhand einer endlichen Menge von vorgegebenen Ein-/Ausgabewert-Paaren. Die Domäne des Lernproblems wird dabei in drei Teile aufgeteilt, das abzuschätzende System, die Lernmaschine und die Trainingsdaten. Diese sind gemäß [CM98] wie folgt definiert:

System

Das System beschreibt den exakten Zusammenhang zwischen den betrachteten Eingabedaten und den zugehörigen korrekten Ausgaben. Dieser Zusammenhang ist in der Praxis zumeist nicht oder nicht vollständig bekannt und häufig nicht vollständig algorithmisch erfassbar.

Ziel des Lernprozesses ist es, die a priori nicht bekannte Systemfunktion anhand von einer Menge von Ein-/Ausgabepaaren des Systems zu approximieren.

Lernmuster

Ein Paar von Eingabewerten für das System und den zugehörigen korrekten Ausgabewerten wird als Muster (engl. Sample) bezeichnet. Zur Abgrenzung von Entwurfsmustern wird in dieser Arbeit statt des Begriffes Muster die eindeutigere Bezeichnung Lernmuster verwendet.

Trainingsdaten und Testdaten

Die endliche Teilmenge von Lernmustern, die als Eingabe für den Lernprozess verwendet wird, wird als Trainingsdaten bezeichnet. Sie bildet die Basis für die Abschätzung der Systemfunktion. Neben den Trainingsdaten gibt es meistens noch weitere Lernmuster, die nicht für den Lernvorgang verwendet werden. Sie dienen vielmehr der Kontrolle der gelernten Systemfunktion nach Abschluß des Lernprozesses und werden als Testdaten bezeichnet.

Lernmaschine

Die Aufgabe der Lernmaschine ist die Nachbildung der unbekannten Systemfunktion zu einem gegebenen System. Hierzu steht der Lernmaschine im Allgemeinen eine Menge von Funktionen zur Verfügung, die aufgrund von a priori-Wissen über die zu bewältigende Lernaufgabe ausgewählt worden ist. Das Problem, das die Lernmaschine zu lösen hat, ist damit die Auswahl der Funktion aus der von ihr unterstützten Funktionsmenge, die die tatsächliche Systemreaktion, gegeben durch eine endliche Menge von Trainingsdaten, am besten approximiert.

Adaption

Als Adaption oder Training wird die Anpassung der Systemabschätzung aufgrund neuer Trainingsdaten bezeichnet. Insbesondere bezieht sich dies auf die Anpassung der Parameter ω in einem Regressionsproblem für einen neuen Satz von Trainingsdaten.

Der Zusammenhang zwischen dem System, der Lernmaschine und den Trainingsdaten ist in Abbildung 3.1 für ein typisches Regressionsproblem dargestellt. Die Lernmaschine erzeugt für eine Eingabe $\mathbf{x}$ aus den Trainingsdaten mit den Parametern ω eine Ausgabe $\tilde{\mathbf{y}}$. Durch den Vergleich mit der tatsächlich vom System produzierten Ausgabe $\mathbf{y}$ kann ω nun für $(\mathbf{x},\mathbf{y})$ angepasst werden, um die Systemfunktion $f(\mathbf{x},\omega_0)+\varepsilon$ besser zu approximieren.

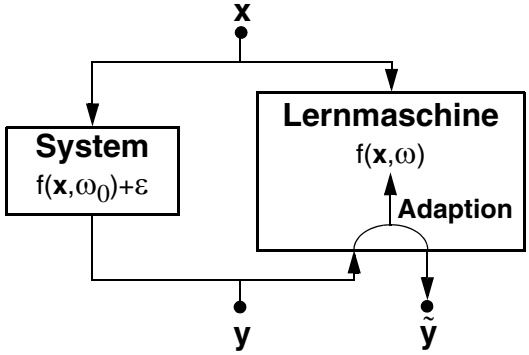


Abbildung 3.1: Lernen eines Regressionsproblems

Die Lernmaschine läßt sich hierbei, wie in Abbildung 3.2 zu sehen ist, in der Regel noch weiter unterteilen in die Systemfunktion, die das Ein-Ausgabe-Verhalten der Lernmaschine beschreibt, und den Lernalgorithmus, der anhand der Ist- und Sollwerte die Systemfunktion anpasst, so daß sie das tatsächliche System besser approximiert.

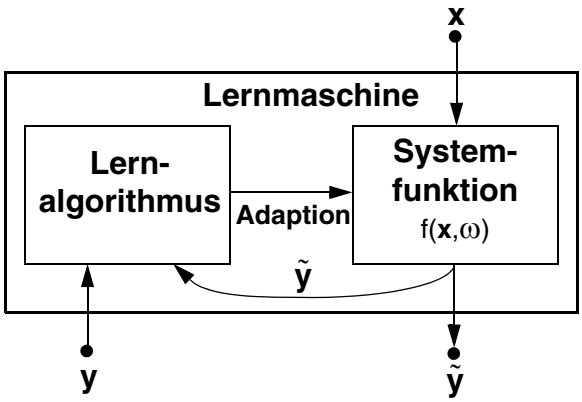


Abbildung 3.2: Die Lernmaschine im Detail

3.2 Adaption von Fuzzy-Regeln und unscharfen Petrinetzen

Mit Hilfe der soeben definierten Begriffe und der Beschreibung der Fuzzy-Bewertung aus Kapitel 2 kann nun die in der Einführung beschriebene Lernaufgabe eingeordnet und präzisiert werden.

Ziel dieser Arbeit ist die Verbesserung der Ergebnisse der Fuzzy-Bewertung auf der Basis der Korrekturen durch den Benutzer im Erkennungsprozess. Bei der Fuzzy-Bewertung wird für jede gefundene Patterninstanz ein Fuzzy-Wert bestimmt. Nach der Beschreibung in Abschnitt 3.1 muß das System demnach jedem Pattern seinen korrekten Fuzzy-Wert zuordnen. Das System, das dies leistet, besteht daher aus dem gesamten interaktiven Bewertungsprozess, also sowohl der automatischen Fuzzy-Bewertung als auch den Korrekturen durch den Benutzer. Die Eingabe dieses Systems besteht aus allen gefundenen Patterninstanzen, sowie ihren Abhängigkeiten untereinander und ist somit durch die Informationen im während der Mustererkennung erzeugten Fuzzy Petrinetz gegeben.

Die Bewertung der gefundenen Pattern erfolgt wie in Kapitel 2 beschrieben durch das Fuzzy Petrinetz. Die Struktur des Netzes ist dabei durch den Erkennungsprozess vorgegeben und kann im Lernprozess nicht verändert werden. Daher kann die gewünschte Adaption nur durch Anpassung der in die Fuzzy-Berechnung eingehenden Parameter erfolgen. Dies sind die Vertrauens- und Schwellwerte der Patternspezifikationen, die den Transitionen im Fuzzy Petrinetz zugeordnet sind. Die vorliegende Lernaufgabe kann somit als Regressionsproblem aufgefaßt werden. Da der Verlauf der durch das FPN realisierten Funktion nichtlinear ist und mehr als eine variable Größe enthält, handelt es sich genauer um ein nonlineares, multivariantes Regressionsproblem.

Zur Lösung eines solchen Problems existieren verschiedene Ansätze, von denen einige im nachfolgenden Abschnitt diskutiert werden.

3.3 Regression und Gradient Descendant Techniken

Es existiert eine Vielzahl unterschiedlicher Lösungsansätze für das in Abschnitt 3.2 vorgestellte Regressionsproblem. Eine Arbeit, die eine zu der vorliegenden Arbeit fast identische Problemstellung behandelt, stammt aus dem Bereich des Datenbank-Reengineering. Im Kontext des in Kapitel 2 schon erwähnten Varlet-Projektes zur Rekonstruktion relationaler Datenbankschemata beschreibt Christoph David Strebin in seiner Diplomarbeit „Adaption unsicheren Wissens auf Basis konnektionistischer Methoden“ [Str99] einen Ansatz zur Adaption von Vertrauens- und Schwellwerten in Generic Fuzzy Reasoning Nets, aus denen die hier verwendeten Pattern Dependency Nets entstanden sind.

Der Ansatz von Strebin verwendet ein neuronales Netz zur Realisierung einer Lernmaschine, die mittels des zur Familie der Gradient Descendent Algorithmen gehörenden Backpropagation-Algorithmus trainiert wird. Gradient Descendant Algorithmen stellen einen akzeptierten Standard zur Lösung von Regressionsproblemen dar [Gal93]. Daher wird diese Algorithmenfamilie zunächst allgemein betrachtet, bevor in Abschnitt 3.4 auf den Ansatz von Strebin eingegangen wird. Da Strebins Arbeit neben seinem eigenen Ansatz eine ausführliche Diskussion zu verschiedenen alternativen Lernansätzen und ihren Problemen für den hier betrachteten Fall enthält, soll auf die Vorstellung weiterer Ansätze an dieser Stelle verzichtet werden.

Das Regressionsproblem ist in Abschnitt 3.1 beschrieben worden als Optimierungsproblem zur Minimierung der Abweichung einer Funktion $f(\mathbf{x},\omega)$ zu den tatsächlichen Werten

$$\mathbf{y} = f(\mathbf{x}, \omega_0) + \varepsilon$$

Für die Herleitung eines möglichst optimalen Parametersatzes ω^* aufgrund der Lernmuster $(\mathbf{x},\mathbf{y})$ wird dabei in der Regel nicht die Funktion f betrachtet, sondern eine Fehlerfunktion e, die die Abweichung zwischen $\mathbf{y}$ und f für alle Lernmuster in Abhängigkeit von dem Parametervektor ω ausdrückt. Die am häufigsten betrachtete Fehlerfunktion ist der mittlere quadratische Fehler (Mean Square Error, MSE), der definiert ist als Summe

$$e(\omega) = \frac{1}{|TD|} \sum_{(x,y) \in TD} (\mathbf{y} - f(\mathbf{x}, \omega))^2 \qquad \textbf{(3.2)}$$

über die Menge TD der Trainingsdaten. Die Optimierung von f bezüglich ω kann damit zurückgeführt werden auf die Minimierung der Fehlerfunktion $e(\omega)$ [Gal93].

Die Fehlerfunktion $e(\omega)$ ist in der Regel nichtlinear und kann daher meist nicht oder nur sehr schwer mit analytischen Methoden minimiert werden. Daher müssen numerische Approximationsverfahren angewendet werden. Gradient Descendent Algorithmen gehören hierbei zu den wichtigsten und meistangewendeten Verfahren [Wer94, And97]. Bekannte Algorithmen dieser Familie sind etwa Newton-Raphson, Polak-Ribiere oder der in Abbildung 3.4 vorgestellte Backpropagation-Algorithmus [And97]. Die Gemeinsamkeit dieser Verfahren ist die iterative Appro-

ximation des gesuchten Minimums mit Hilfe des Gradienten. Der Gradient der Fehlerfunktion ist gegeben durch

$$\nabla e(\omega) = \frac{\partial}{\partial \omega} e(\omega) \tag{3.3}$$

und beschreibt einen Vektor, der in Richtung des steilsten Anstiegs von e zeigt. Das heißt durch Anpassung von ω in Richtung von $\nabla e(\omega)$ würde der Fehler am stärksten wachsen. Umgekehrt verringert der entgegengesetzte Vektor $-\nabla e(\omega)$ für hinreichend kleine Anpassungsschritte $\Delta\omega$ den Fehler in der Umgebung von ω. Dies wird in den Gradientenverfahren ausgenutzt, um sich iterativ an ein Minimum anzunähern. Ausgehend von einem beliebig gewählten Startwert ω_1 wird der Parametervektor iterativ angepaßt:

$$\omega_{i+1} = \omega_i - \rho \nabla e(\omega_i) \tag{3.4}$$

Der Parameter ρ gibt dabei die Schrittweite der Anpassung an. Je nach verwendetem Algorithmus ist er fest gewählt oder wird bei jeder Iteration dynamisch errechnet. Die Iteration wird solange wiederholt, bis die Differenz aufeinanderfolgender Werte hinreichend klein ist. Dann hat $e(\omega)$ (näherungsweise) ein Minimum angenommen.

3.4 Neuronale Netze und Backpropagation

Der Backpropagation-Algorithmus für neuronale Netze ist das wohl bekannteste Gradient Descendent Verfahren im Bereich konnektionistischer Modelle[Gal93]. Aufgrund eines besonders effizienten Berechnungsverfahrens für den Gradienten und die übersichtliche Darstellung komplexer Systemzusammenhänge durch das neuronale Netz ist er vor allem für größere Modelle mit vielen anzupassenden Parametern geeignet.

Christoph David Strebin stellt in seiner Arbeit [Str99] ein Fuzzy Neural Net vor, das mit Hilfe des Backpropagation-Algorithmus die Vertrauenswerte eines Generic Fuzzy Reasoning Nets auf der Basis von Benutzerkorrekturen lernen kann. Da es sich bei dem Backpropagation-Algorithmus um ein akzeptiertes Standard-Lernverfahren zum Lösen von Regressionsproblemen handelt, soll er in dieser Arbeit als Vergleichsalgorithmus zur Bewertung der Lernergebnisse des im nächsten Kapitel vorgestellten Lernverfahrens dienen. Daher wird im Folgenden genauer auf diesen Ansatz eingegangen. Zunächst wird die Struktur des verwendeten neuronalen Netzes beschrieben und an die hier verwendeten Pattern Dependency Nets angepaßt. Anschließend wird auf die Konstruktion der Lernmuster eingegangen, bevor zum Schluß der Backpropagation-Algorithmus beschrieben wird. Eine formalere Beschreibung neuronaler Netze und die Herleitung des Backpropagation-Algorithmus sind in [Str99] zu finden.

3.4.1 Das Fuzzy Neural Net

Grundlagen neuronaler Netze

Ein neuronales Netz besteht aus einer Menge von Berechnungseinheiten, die als Neuronen bezeichnet werden. Sie sind durch gerichtete Kanten miteinander verbunden, denen Gewichte

zugeordnet sind. Das Gewicht $w_{i,j}$ einer Kante gibt den Einfluß des Neurons u_i auf das Neuron u_j wieder. Es wird unterscheiden zwischen Eingabe-Neuronen, inneren (oder versteckten) Neuronen und Ausgabeneuronen. Eingabe-Neuronen haben keine einlaufenden Kanten. Sie erhalten einen externen Eingabewert zugewiesen und führen keine Neuberechnung ihres Ausgabewertes durch. Sie werden durch Rechtecke dargestellt, die den Eingabewert enthalten. Ausgabeneuronen haben typischerweise keine ausgehenden Kanten. Sie berechnen Werte, die die Ausgabe des neuronalen Netzes darstellen. Alle Neuronen, die nicht Ein- oder Ausgabeneuronen sind, werden als innere Neuronen bezeichnet. Ausgabe- und innere Neuronen werden durch Kreise dargestellt, die zusätzliche Symbole zur Kennzeichnung ihrer Berechnungsvorschrift enthalten können.

Der von jedem Neuron berechnete skalare Ausgabewert wird als Aktivierung bezeichnet [Gal93][1]. Sowohl die Eingabewerte eines Neurons als auch die Aktivierung können Werte aus dem Einheitsintervall [0,1] annehmen. Eingabeneuronen berechnen ihre Aktivierung aus ihren externen Eingabewerten. Die Berechnung der Aktivierung innerer und Ausgabeneuronen erfolgt auf Basis der Aktivierungen der mit ihnen durch einlaufende Kanten verbundenen Neuronen und der zugehörigen Kantengewichte. Daneben ist diesen Neuronen noch ein als Bias bezeichneter Wert zugeordnet, der als Gewicht $w_{0,i}$ einer Kante von einem speziellen Neuron u_0 mit Aktivierung 1 zu jedem anderen Neuron u_i (außer Eingabeneuronen) aufgefaßt werden kann und daher bei den folgenden Ausführungen nicht von den anderen Gewichten unterschieden werden muß. Zur einfacheren Darstellung des neuronalen Netzes wird u_0 nicht explizit abgebildet. Stattdessen werden die Bias-Gewichte in den jeweiligen Ziel-Neuronen plaziert.

Die Berechnung der Aktivierung a_i eines Neurons u_i erfolgt in zwei Schritten [Gal93]. In einem ersten Schritt wird durch die Netzeingabefunktion $NET_i:([0, 1] \times \mathbb{R})^{n+1} \rightarrow \mathbb{R}$ aus den eingehenden Aktivierungen und Gewichten und dem Bias des Neurons die Netzeingabe net_i bestimmt, wobei n die Anzahl eingehender Kanten von u_i ist. Aus net_i wird dann durch die Aktivierungsfunktion $f_i:\mathbb{R} \rightarrow [0, 1]$ die Aktivierung a_i des Neurons berechnet. In herkömmlichen neuronalen Netzen ist die Netzeingabefunktion zumeist für alle Neuronen definiert als gewichtete Summe über die eingehenden Kanten von Neuronen uj mit

$$net_i = \sum_{j=0}^{n} w_{i,j} a_j \qquad \textbf{(3.5)}$$

Die Aktivierungsfunktion f_i wird auch häufig als Komprimierungsfunktion (squashing function)

1. Die formale Definition neuronaler Netze in [Str99] unterscheidet zwischen Aktivierung und Ausgabe und verwendet eine komplexere Form der Aktivierungsfunktion. Hier wird dagegen eine vereinfachte Beschreibung neuronaler Netze verwendet, die in der gängigen Literatur [Gal93, Wer94] häufig zu finden ist und auch im weiteren Verlauf von [Str99] verwendet wird.

bezeichnet, da sie die Ausgabe der Netzeingabefunktion auf das Intervall [0,1] komprimiert. Der Aufbau eines Neurons ist in Abbildung 3.3 noch einmal schematisch dargestellt [Gal93].

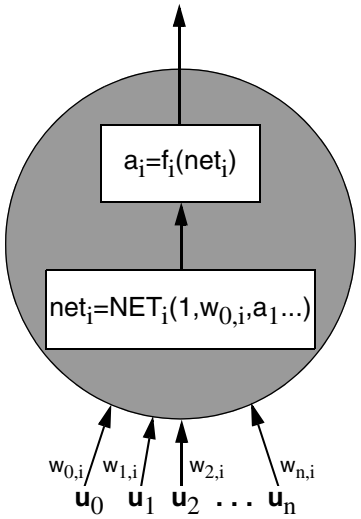


Abbildung 3.3: Ein Neuron

Aufgabe des in Abschnitt 3.4.3 beschriebenen Backpropagation-Algorithmus ist es, die Kantengewichte und Bias-Werte des neuronalen Netzes entsprechend den gegebenen Lernmustern anzupassen. Um ein neuronales Netz mit diesem Algorithmus trainieren zu können, muß es zwei wichtige Bedingungen erfüllen. Da es sich bei dem Backpropagation-Algorithmus um ein Gradient Descendant Verfahren handelt, muß zum einen sichergestellt sein, daß der Gradient für alle Neuronen berechnet werden kann. Hierzu müssen alle verwendeten Netzeingabe- und Aktivierungsfunktionen partiell differenzierbar sein. Zum anderen setzt der Backpropagation-Algorithmus voraus, daß das neuronale Netz keine Zyklen enthält. Solche zyklenfreien Netze werden auch als feed-forward bezeichnet.

Struktur des Fuzzy Neural Nets

Damit das verwendete neuronale Netz die korrekten Vertrauens- und Schwellwerte lernen kann, muß es die Berechnungsvorschriften der Fuzzy Petrinetze nachbilden, aus denen die Lernmuster stammen. Hierbei ergibt sich das Problem, daß sich die Fuzzy Petrinetze aus unterschiedlichen Analyseprozessen voneinander unterscheiden, selbst wenn sie auf demselben PDN basieren. Das neuronale Netz muß indes für alle Lernmuster strukturell unverändert bleiben. Das in [Str99] vorgestellte Fuzzy Neural Net (FNN) orientiert sich in seiner Struktur daher nicht an den Fuzzy Petrinetzen, sondern an dem konstant bleibenden PDN. Aufbau und Verarbeitungsfunktionen des FNN sind dabei so gewählt, daß das FNN die Berechnungsvorschrift der Fuzzy Petrinetze für die verwendeten Lernmuster möglichst gut approximiert. Bei diesem Ansatz kann zu jedem Pattern des PDN nur eine Instanz zugleich im FNN repräsentiert werden, während im FPN typischerweise viele Stellen zu Instanzen desselben Patterns zu finden sind. Die Trainingsdaten für das FNN entstehen daher durch Aufteilung der Patterninstanzen eines FPNs auf mehrere Lernmuster, wie in Abschnitt 3.4.2 noch beschrieben wird.

Jede Implikation des PDN wird im Fuzzy Neural Net durch mehrere Stellen ausgedrückt, die gemeinsam die Berechnungsvorschrift für das normalerweise von der zugehörigen Transition im FPN bestimmte Fuzzy Truth Token aus Definition 2.7 realisieren. Die Minimums- und Schwellwertbildung ist dabei in den Netzeingabefunktionen realisiert, da nur dort alle Eingabewerte einzeln vorliegen. Dies ersetzt damit die erwähnte, normalerweise übliche Summenbildung. Die Aktivierungsfunktion, die üblicherweise die charakteristische Berechnungsfunktion eines Neurons ist, wird bei diesem Ansatz nicht benötigt und kann als Identitätsabbildung angenommen werden. Abbildung 3.4 zeigt den Ausschnitt eines FNN, der die darüber abgebildete Implikation nachbildet.

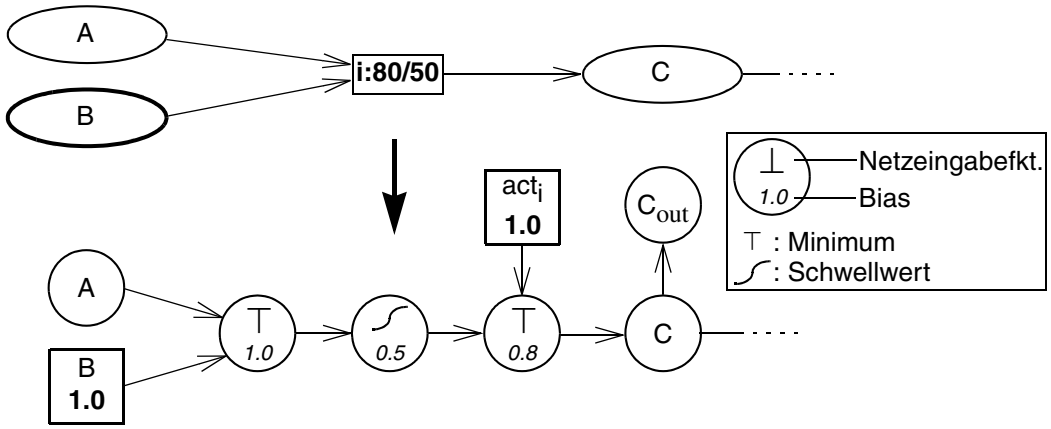


Abbildung 3.4: Implikation im FNN

Jedes Prädikat des PDN wird durch ein Neuron dargestellt. Axiome werden zu Eingabeneuronen. Ihre externe Eingabe ist stets 1, was ihrem Wert von 100 im FPN nach Skalierung auf das Einheitsintervall des FNN entspricht. In der Abbildung ist dies für das Axiom B dargestellt. Die berechneten Fuzzy-Werte zu allen anderen Prädikaten sind Ausgaben des Fuzzy-Bewertungssystems und erhalten daher im FNN ein zusätzliches Ausgabeneuron, wie für Prädikat C dargestellt. Die Implikation selbst wird durch drei Neuronen realisiert, die die Berechnungsschritte der FTT-Berechnung nachbilden. Das erste Neuron berechnet das Minimum der Eingabewerte. Sein Bias ist fest mit 1.0 vorgegeben und wird vom Lernprozess nicht verändert. Das nächste Neuron realisiert die Schwellwertfunktion. Sein Bias entspricht dem Schwellwert der Implikation nach Skalierung auf das Einheitsintervall. Das dritte Neuron berechnet schließlich das Minimum aus dem bisherigen Wert und dem Vertrauenswert der Implikation. Sein Bias entspricht wiederum dem skalierten Vertrauenswert der Implikation. Kantengewichte werden für die Nachbildung des PDN nicht benötigt und sind daher nicht dargestellt. Sie werden als Gewichte der Größe 1 angenommen und vom Lernprozess nicht verändert.
Da das FNN für alle Lernmuster in seiner Struktur gleich bleiben soll, muß es eine Möglichkeit geben, das Netz anhand seiner Eingabe für die aus dem FPN extrahierten Lernmuster zu konfigurieren, um etwa Verbindungen über Implikationen, die im Erkennungsprozess nicht geschaltet haben, zu deaktivieren. Hierzu werden Aktivierungsneuronen in das FNN eingefügt, die eine externe Eingabe von 0 oder 1 haben können. Ein Wert von 0 deaktiviert dabei die betreffende

Implikation, indem er das Berechnungsergebnis der Minimumsfunktion auf 0 senkt. Ein Wert von 1 aktiviert die Implikation.

Mit der Einführung von Vererbung und optionalen Kanten im PDN tritt bei der Generierung des FNN aus dem PDN das Problem auf, daß nicht alle Konfigurationen, in denen eine Implikation schalten kann, im PDN erfaßt sind. Durch Polymorphie und nicht erfüllte optionale Prämissen entstehen neben den im PDN explizit aufgeführten Konfigurationen weitere Möglichkeiten, die im FPN aufgefunden werden können. Diese Möglichkeiten müssen im FNN entsprechend ausgedrückt werden. Hierzu wird für jede Kombination von Prädikaten, die im Vorbereich einer Implikation unter Berücksichtigung von Polymorphie und optionalen Kanten vorgefunden werden kann, ein Neuron im FNN erzeugt, das die initiale Minimumsbildung der FTT-Berechnung übernimmt. Um die tatsächlich im FPN für ein Lernmuster vorgefundene Konfiguration auswählen zu können, wird jedem dieser Neuronen zudem ein Aktivierungsneuron zugeordnet. Da bei der Konstruktion der Lernaufgabe unter Umständen mehrere Konfigurationen für eine Implikation benötigt werden, wird zudem ein weiteres Neuron zwischen die Minimumsneuronen und das Schwellwertneuron geschaltet, das eine Disjunktion zwischen den aktivierten Neuronen durch Maximumsbildung realisiert. Ein Beispiel für die resultierende FNN-Struktur ist in Abbildung 3.5 dargestellt.

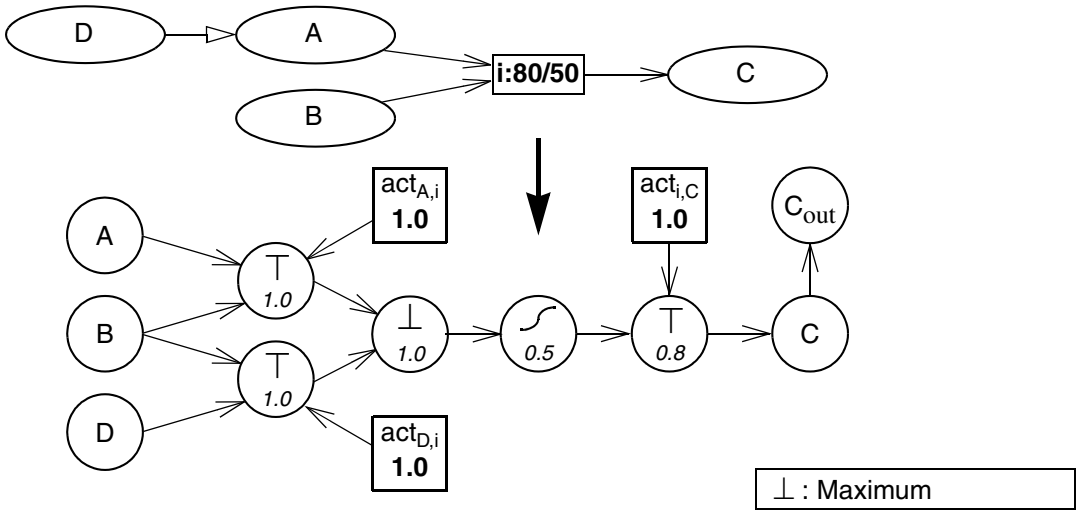


Abbildung 3.5: Umsetzung von Vererbung im FNN

Ein Problem bei der Generierung des FNN stellen Kreise im PDN dar. Sowohl Kreise, die nur durch Implikationskanten erzeugt werden, als auch solche, die unter Berücksichtigung von Vererbungskanten entstehen, induzieren Zyklen im FNN. Hierdurch wird die Forderung nach einem Feed-Forward-Netz für den Backpropagation-Algorithmus verletzt. In [Str99] werden solche Kreise durch eine Umformung des GFRN aufgelöst, die auf einer Umkehrung der Implikationsrichtung basiert. Diese Umkehrung ist dort möglich, da die Implikationsbedingungen des GFRN aus quantorenlogischen Ausdrücken bestehen, die durch Äquivalenzumformungen geeignet umgestellt werden können. Da in den hier verwendeten PDNs stattdessen nicht umformbare Graphtransformationsregeln benutzt werden, ist dieses Verfahren nicht übertragbar. Da das Lernen mit neuronalen Netzen hier lediglich als Vergleichsansatz dient, ist dieses

Problem im Rahmen dieser Arbeit jedoch nicht weiter betrachtet worden. Stattdessen wird für die weitere Betrachtung der neuronalen Netze von einem kreisfreien PDN als Grundlage ausgegangen.

Die Verarbeitungsfunktionen des Fuzzy Neural Net

Um die Funktion eines Fuzzy Petrinetzes nachbilden zu können, müssen die Verarbeitungsfunktionen der Neuronen die Minimums- und die Schwellwertfunktion realisieren. Daneben wird, wie erläutert, noch die Maximumsfunktion für die Disjunktion unterschiedlicher Implikations-Konfigurationen benötigt. Da für den Gradientenabstieg des Backpropagation-Algorithmus partiell differenzierbare Funktionen benötigt werden, werden nach [Str99] für das Fuzzy Neural Net die Funktionen des FPN durch stetige Funktionen approximiert. Für das Minimum und das Maximum sind dies die t-Norm $\top$ und die dazu duale t-Conorm $\perp$ mit

$$\top(a, b) = 1 - [(1-a)^p + (1-b)^p]^{1/p} \tag{3.6}$$

$$\perp(a, b) = [a^p + b^p - a^p b^p]^{1/p} \tag{3.7}$$

und für den Schwellwert t die Sigmoid-Funktion S mit

$$S(a, t) = \frac{a}{1 + e^{-p(a-t)}} \tag{3.8}$$

Der Parameter p bestimmt hierbei die Qualität der Approximation und ist stets positiv. Für größere Werte von p nähern sich die verwendeten Funktionen stärker den approximierten Funktionen an. Da dadurch jedoch auch über weite Strecken der Verlauf der Funktionen flacher wird und somit der Gradient abnimmt, sinkt auch die Konvergenzgeschwindigkeit des Backpropagation-Algorithmus.

3.4.2 Konstruktion der Lernmuster

Ein Lernmuster für das neuronale Netz besteht aus einem Vektor von Eingabewerten für alle Eingabeneuronen des Netzes und einem zugehörigen Vektor von zu erlernenden Ausgabewerten für die Ausgabeneuronen. Die Informationen für die zu erlernenden Abhängigkeiten sind wie in Abschnitt 3.2 beschrieben in den Fuzzy Petrinetzen zu finden und müssen in geeignete Ein- und Ausgabewerte für das FNN transformiert werden.

Durch die Anlehnung des neuronalen Netzes an die Struktur des PDN ergibt sich das Problem, daß das FNN nur jeweils eine Instanz jedes Patterns zugleich repräsentieren kann. Das FPN, das wie in Abschnitt 3.2 beschrieben die Systemeingabe repräsentiert, enthält dagegen typischerweise mehrere Instanzen desselben Patterns. Um das Fuzzy Neural Net als Lernmaschine für das Fuzzy-Bewertungssystem einsetzen zu könne, ist es daher erforderlich, die Informationen des FPN auf geeignete Lernmuster für das FNN aufzuteilen, so daß die Berechnungsergebnisse des FNN weiterhin näherungsweise mit denen des FPN übereinstimmen. Die möglichen Abhängigkeiten zwischen Stellen des FPN sind im FNN statisch realisiert. Bei der Abbildung des FPN auf das FNN besteht die Aufgabe daher darin, das FNN mit Hilfe der Aktivierungsneuronen entsprechend der im FPN vorgefundenen Situation zu konfigurieren. Die entscheidenden Eingabe-

werte für ein Lernmuster bestehen daher aus entsprechenden Werten für die Aktivierungsneuronen. Daneben gehören auch die Werte zu Axiomen des PDN zur Eingabemenge. Sie sind jedoch fest mit 1.0 festgelegt und unabhängig von dem Lernmuster.

In [Str99] wird ein Verfahren zur Aufteilung der Patterninstanzen und Abhängigkeiten eines FPN auf mehrere Lernmuster beschrieben. Hierzu werden zunächst die Patterninstanzen ausgewählt, für die das Netz trainiert werden soll. Dies sind zunächst alle Instanzen, für die gesicherte Daten in Form von benutzerdefinierten Fuzzy-Werten vorliegen. Desweiteren werden die übrigen Instanzen aller Pattern verwendet, zu denen mindestens eine Instanz mit benutzerdefininerten Werten existiert, da sie gegebenenfalls Gegenbeispiele zu den benutzerdefinierten Werten bilden, die im Lernprozess zu berücksichtigen sind.

Für die ausgewählten Patterninstanzen können anschließend Lernmuster konstruiert werden. Hierzu werden für jede Patterninstanz alle für den berechneten Fuzzy-Wert relevanten Stellen im FPN bestimmt. Die relevanten Stellen ergeben sich dabei durch die Rückverfolgung der dominierenden Werte im FPN ausgehend von der Stelle zu der gewählten Startinstanz. Bei der Rückverfolgung wird versucht, die Patterninstanzen zu den relevanten Stellen an die entsprechenden Neuronen des FNN zu binden. Da auch unter den relevanten Instanzen häufig noch mehrere Instanzen desselben Patterns sind, kann es hierbei zu Konflikten kommen. Diese werden gelöst, indem die zueinander in Konflikt stehenden möglichen Bindungen in verschiedenen Lernmustern realisiert werden. Die so entstandenen Bindungen werden anschließend vervollständigt, indem Stellen aus den Vorbereichen der gebundenen, relevanten Stellen an die noch freien Neuronen gebunden werden. Konflikte durch mehrere Instanzen gleicher Pattern werden hierbei wiederum durch die Aufteilung auf verschiedene Lernmuster gelöst. Abbildung 3.6 zeigt die möglichen Bindungen für einen Ausschnitt eines FPN und das dazugehörige PDN. Die relevanten Stellen im FPN sind fett dargestellt. Da die relevanten Stellen b1 und b2 beide vom Typ B sind, stehen sie zueinander in Konflikt und werden in unterschiedlichen Bindungen berücksichtigt. Um den Vorbereich der Implikation i1 vollständig zu binden, wird neben den relevanten Stellen noch jeweils eine Stelle für das Pattern C benötigt. Hierzu stehen die zwei Alternativen c1 und c2 zur Verfügung. Da unter Umständen jede der beiden Stellen durch den Lernprozess zu einer relevanten Stelle werden kann, müssen sie beide bei der Konstruktion der Lernmuster berücksichtigt werden. Die sich ergebenden Möglichkeiten werden wiederum in unterschiedlichen Bindungen realisiert, so daß insgesamt die vier dargestellten Bindungen gefunden werden. Durch die Verwendung der relevanten Stellen ist für jede dieser Bindungen vor dem Lernprozess sichergestellt, daß der vom FNN für die Ausgabepattern berechnete Fuzzy-Wert eine gute Approximation des im FPN berechneten Wertes ist.

Aus den gefundenen Bindungen ergeben sich unmittelbar die Lernmuster für das neuronale Netz. Die Ausgabewerte für ein Lernmuster sind dabei durch die Fuzzy-Werte aller zu Beginn ausgewählten Ausgabe-Patterninstanzen gegeben. Die Werte für die Aktivierungsneuronen, die die Eingabe des Lernmusters bilden, werden durch die gefundenen Bindungen festgelegt. Für jede gebundene Patterninstanz wird hierzu die Transition aus dem Vorbereich der Stelle im FPN betrachtet. Das Aktivierungsneuron für die zugehörige Implikation wird aktiviert, indem es im Lernmuster einen Eingabewert von 1.0 erhält. Entsprechend wird auch das Aktivierungsneuron

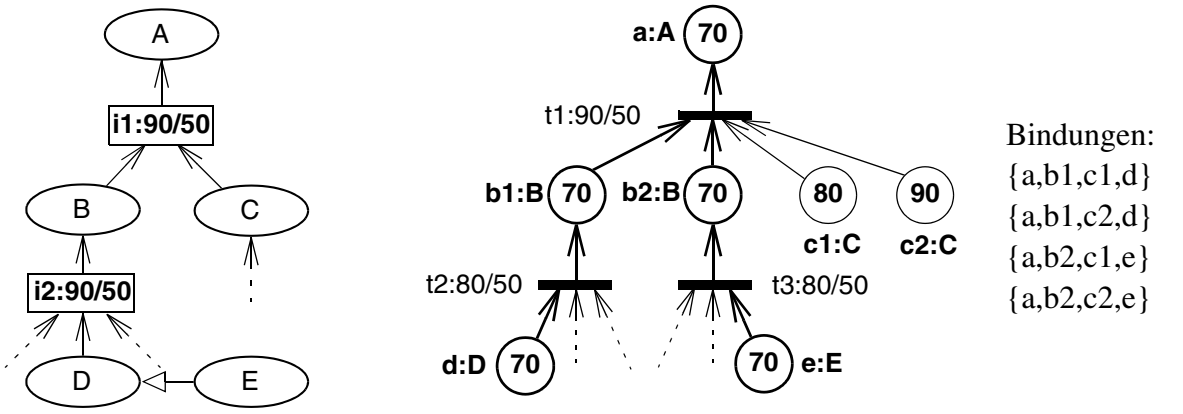


Abbildung 3.6: Fuzzy Petrinetz nach der Auswertung

für den im FPN vorgefundenen Vorbereich der Transition mit einem Eingabewert von 1.0 aktiviert. Alle Aktivierungsneuronen, die nicht durch gebundene Patterninstanzen aktiviert werden, erhalten einen Eingabewert von 0.

3.4.3 Der Backpropagation-Algorithmus

Der Backpropagation-Algorithmus gehört zur Familie der in Abschnitt 3.3 beschriebenen Gradient Descendant Algorithmen. Er adaptiert die Kantengewichte und Bias-Werte Sein Name stammt von dem besonders effizienten Verfahren, mit dem der Gradient der Fehlerfunktion berechnet wird. Bei diesem Verfahren wird die Gewichtsanpassung schrittweise für alle Gewichte des neuronalen Netzes berechnet, wobei das Netz beginnend mit den Ausgabeneuronen rückwärts zu den Eingabeneuronen durchlaufen wird. Dabei können große Teile der Gradientenberechnung für ein Gewicht auf bereits vorher berechnete Werte zurückgeführt werden. Diese wiederverwertbaren Teile der Gradientenberechnung werden für jedes Neuron ui als Fehlersignal di berechnet mit

$$\delta_i = -\frac{\partial e(w)}{\partial net_i} \qquad \textbf{(3.9)}$$

Für Ausgabeneuronen ist δ_i direkt gegeben durch

$$\delta_i = (y_i - a_i) \cdot f'(net_i) \qquad \textbf{(3.10)}$$

Für innere Neuronen kann δ_i mit Hilfe der bereits berechneten Werte δ_j der Nachfolger-Neuronen bestimmt werden. Für ein Neuron u_i mit n Nachfolger-Neuronen u_j gilt:

$$\delta_i = \sum_{j=1}^{n} \frac{\partial NET_j}{\partial a_i} \delta_j \cdot f'(net_i) \qquad \textbf{(3.11)}$$

Die Gewichtskorrektur nach Gleichung (3.4) kann damit wegen (3.9) ausgedrückt werden als

$$w_{j,i}^{(k+1)} = w_{j,i}^{(k)} + \rho\delta_i f_i'(net_i) \cdot \frac{\partial NET_i}{\partial w_{j,i}^{(k)}} \quad \textbf{(3.12)}$$

Es muß also nur die partielle Ableitung für die Netzeingabefunktion für das anzupassende Gewicht neu berechnet werden, wodurch die Berechnung der Gewichtskorrektur besonders effizient wird.

Mit Hilfe dieser Berechnungsvorschriften führt der Backpropagation-Algorithmus die iterative Anpassung der Gewichte durch, wie in Abschnitt 3.3 beschrieben. Hierzu wird nacheinander für jedes aus dem FPN ermittelte Lernmuster zunächst das FNN in einem Vorwärtsdurchlauf ausgewertet. Hierbei werden die Aktivierungen ai und die Netzeingaben neti für alle Neuronen berechnet und stehen für die anschließende Berechnung der di zur Verfügung. Nach der Auswertung des Netzes folgt ein Rückpropagationsschritt, in dem mit Gleichung (3.11) und (3.12) die di berechnet und die Gewichtsanpassungen vorgenommen werden. Dieser Vorgang wird solange wiederholt, bis die Gewichtsanpassungen hinreichend klein geworden sind. Abbildung 3.7 zeigt den Algorithmus in Pseudo-Code.

```
repeat
   foreach Sample s do
      evaluate FNN for s.input
      compare FNN.output with s.output
      backpropagate error:
         calculate d_i for all Neurons u_i with (3.11)
         update weights w with (3.12)
until weight correction Δw≈0
```

Abbildung 3.7: Backpropagation als Pseudo-Code

3.5 Probleme von Gradient Descendant Verfahren

Bei der Anwendung von Gradient Descendant Verfahren zum Lernen von Regressionsproblemen ergeben sich zahlreiche Schwierigkeiten, die sie insbesondere für das hier betrachtete Adaptionsproblem als nicht geeignet erscheinen lassen.

Differenzierbare Fehlerfunktion

Eine wichtige Voraussetzung für die Anwendung von Gradient Descendant Verfahren ist die Differenzierbarkeit der Fehlerfunktion. Für das hier betrachtete Problem stellt dies ein Problem dar, da die im Fuzzy Petrinetz verwendeten Minimums- und Schwellwertoperationen nicht differenzierbar sind. Die Abbildung auf differenzierbare Funktionen hat unweigerlich einen Näherungsfehler zu Folge. Da eine gute Annäherung an die nicht-stetigen Funktionen gleichzeitig

eine Verschlechterung der Konvergenzgeschwindigkeit bewirkt (siehe [Str99]), ist hier zwischen der Konvergenzgeschwindigkeit und der Qualität der Ergebnisse abzuwägen.

Lokale Minima

Durch das Zurückverfolgen des Gradienten kann nicht garantiert werden, daß ein globales Minimum gefunden wird. Vielmehr konvergiert das Verfahren bereits bei Erreichen eines lokalen Minimums. Durch geeignete Wahl der Schrittweite bei der Iteration kann dies zwar teilweise vermieden werden. Eine vollständige Vermeidung dieses Falles ist jedoch im Allgemeinen sehr umständlich und rechenintensiv und daher in der Praxis kaum realisierbar.

Feste Systemfunktion

Ein besonderes Problem im Kontext der in dieser Arbeit zu lösenden Lernaufgabe stellt die approximierte Systemfunktion f dar. Alle herkömmlichen Ansätze zur Lösung des Regressionsproblems gehen davon aus, daß die zu approximierende Funktion für alle Lernmuster dieselbe ist und lediglich eine Veränderung der Parameter stattfindet. Bei der Fuzzy-Evaluation der Design Pattern wird diese Funktion durch das Fuzzy Petrinetz realisiert. Da das Petrinetz von den erkannten Pattern abhängt, ist die Systemfunktion aber nach jedem Erkennungsprozess eine andere und nicht für alle Lernmuster konstant. Lediglich die im Fuzzy Petrinetz verwendeten Parameter sind für alle Lernmuster dieselben. Um dennoch ein herkömmliches Regressionsverfahren wie Backpropagation auf diese Lernaufgabe anwenden zu können, muß also die vom Fuzzy Petrinetz vorgegebene Systemfunktion in der Lernmaschine durch eine gleichbleibende Funktion ausgedrückt werden. Dies geschieht wie beschrieben durch die Abbildung der PDN-Struktur auf das FNN. Hierdurch ergibt sich jedoch das Problem, daß die Lernmuster, die ursprünglich durch die vom Inferenzprozess erzeugten FPNs gegeben sind, auf diese Struktur abgebildet werden müssen wie in Abschnitt 3.4.2 beschrieben. Dies führt zum einen zu weiteren Approximationsfehlern im Lernprozess. Zum anderen hat die Abbildung zur Folge, daß zur Berücksichtigung aller entscheidenden Kombinationen von Ein- und Ausgabestellen des FPN bei der Abbildung aus jedem FPN eine große Anzahl von Lernmustern erzeugt wird. Im ungünstigsten Fall kann die Zahl der Lernmuster exponentiell mit der Tiefe des FPN wachsen.

Konvergenzgeschwindigkeit und Divergenz

Ein weiteres Problem stellt die Konvergenz von Gradient Descendant Ansätzen dar.

Die für den Gradientenabstieg verwendete Fehlerfunktion weist bei den meisten praxisrelevanten Problemen über weite Strecken einen flachen Verlauf mit vielen Sattelstellen auf. Dieser Effekt wird als Ill-Conditioning[Wer94] bezeichnet. Durch die Sattelstellen und flachen Regionen benötigt der Gradientenabstieg sehr lange, um zu vernünftigen Werten zu gelangen. [Sar99] zeigt, daß bereits einfachste Netze mit nur einem Eingabe- und einem Ausgabeneuron und ohne innere Neuronen aufgrund von Ill-Conditioning je nach Startwerten der Kantengewichte von schneller Konvergenz bis zu völliger Divergenz schwanken können. Das Problem des Ill-Conditioning kann zwar durch sorgfältige Einstellung des neuronalen Netzes und gegebenenfalls strukturelle Veränderungen vermieden werden[vdSH98]. Dies setzt jedoch eine vorsichtige,

manuelle Konstruktion des neuronalen Netzes voraus, die bei dem hier betrachteten Ansatz nicht gegeben ist.

Im praktischen Einsatz zeigt sich, daß durch die aus der Abbildung auf das FNN resultierende extrem hohe Anzahl an zu verarbeitenden Lernmustern zusammen mit Ill-Conditioning Problemen der Backpropagation-Ansatz bereits für FPNs mit mehr als etwa hundert Stellen nicht mehr sinnvoll einsetzbar ist. Daher wird im folgenden Kapitel ein Lernansatz vorgestellt, der das gegebene Regressionsproblem mit einem nicht-iterativen Algorithmus auf Basis statistischer Überlegungen effizient löst.

KAPITEL 4

Der Lösungsansatz: Statistische Adaption im FPN

Gegenstand diese Kapitels ist der Entwurf eines heuristischen Adaptionsalgorithmus zur Anpassung der Vertrauens- und Schwellwerte eines Musterkataloges auf der Basis der vom Benutzer vorgenommenen Korrekturen. Hierzu werden in Abschnitt 4.1 zunächst die Anforderungen aus Kapitel 1 und die in Kapitel 3 identifizierten Probleme herkömmlicher Ansätze analysiert. Auf dieser Basis wird dann in Abschnitt 4.2 ein Lösungsansatz entwickelt, der in den folgenden Abschnitten weiter ausgeführt wird. Abschnitt 4.3 behandelt die Aufbereitung der Daten eines einzelnen Lernmusters, bevor diese in Abschnitt 4.4 in die Adaption der Vertrauenswerte eingehen. Dasselbe Verfahren wird anschließend auf die Adaption der Schwellwerte angewandt.

4.1 Entwurfskriterien

Wie in der Einführung beschrieben, besteht das Hauptproblem für den zu realisierenden Lernansatz in der Größe der betrachteten Systeme. Bei zu analysierenden Systemen von mehreren Hunderttausend bis hin zu Millionen Zeilen Quelltext muß ein in der Praxis einsetzbares Lernverfahren besonders gut skalierbar sein, um die Laufzeitkomplexität der gesamten Analyse nicht nachhaltig zu verschlechtern oder durch erhöhten Speicherbedarf die Maximalgröße der analysierbaren Systeme einzuschränken. Die beiden Hauptkriterien für den Entwurf sind daher die *Laufzeiteffizienz* und die *Speichereffizienz* des Lernverfahrens.

Die Laufzeitprobleme der in Kapitel 3 betrachteten iterativen Gradientenverfahren legen nahe, daß mit einer iterative Lösung nicht die erforderliche Laufzeiteffizienz zu erreichen ist. Iterative Ansätze erfordern die wiederholte Auswertung der Systemfunktion, also des Fuzzy Petrinetzes, wodurch die Netzgröße einen starken Einfluß auf die Laufzeit solcher Ansätze hat. Daher soll für den im Folgenden entwickelten Ansatz eine *One-Shot-Lösung*[CM98] verwendet werden, bei der jedes Lernmuster nur genau einmal betrachtet werden muß. Ein solcher Ansatz ist zwar verfahrensbedingt in der Regel ungenauer, als eine iterative Lösung. Jedoch stellt dies in dem hier betrachteten Fall einen akzeptablen Kompromiss zugunsten der Laufzeit dar, insbesondere da die berechneten Werte stets nur Richtgrößen für die Analyse durch den Reengineer darstellen können, also eine absolute Genauigkeit von untergeordneter Bedeutung ist, solange die Abweichungen nicht zu hoch sind.

Ein weiteres wichtiges Kriterium ist die Realisierung der Systemfunktion nach Abbildung 3.2. Ein geeignetes Berechnungsnetz ist bereits in Form des Fuzzy Petrinetzes vorhanden. Es ist daher wünschenswert, dieses Netz direkt im Lernalgorithmus zu verwenden, anstatt wie im Fall

der FNNs aus Kapitel 3 ein zusätzliches, Lernalgorithmus-spezifisches Netz einzuführen. Damit entfällt zum einen die Mehrfachauswertung der Fuzzy-Werte durch unterschiedliche Netze. Zum anderen werden Approximationsfehler durch Abbildung auf das Lern-Netz vermieden, wie sie etwa im Fall der Neuronalen Netze entstanden sind. Da mit den Fuzzy Petrinetzen die Systemfunktion des zu entwerfenden Lernverfahrens bereits realisiert ist, muß für die vollständige Beschreibung der Lernmaschine nur noch der Lernalgorithmus entwickelt werden.

Besonderes Augenmerk ist hierbei auf die Tatsache zu richten, daß jeder der anzupassenden Parameter in den Fuzzy Petrinetzen mehrfach vorkommen kann. Dies, zusammen mit der hohen Knotenzahl der FPNs, stellt für herkömmliche Regressionsansätze, wie in Kapitel 3 beschrieben, ein großes Problem dar. Andererseits bedeutet es aber auch, daß für jeden dieser Parameter eine große Zahl von Korrekturdaten, und damit wichtigen Lerninformationen, zur Verfügung steht. Der im Folgenden beschriebene Ansatz soll daher dieses hohe Datenaufkommen bestmöglich ausnutzen, und so diesen Nachteil herkömmlicher Ansätze in einen Vorteil verwandeln.

Der auf diesen Kriterien basierende Ansatz wird nun im folgenden Abschnitt zunächst konzeptionell hergeleitet, bevor im weiteren Verlauf die detaillierte Ausarbeitung erfolgt.

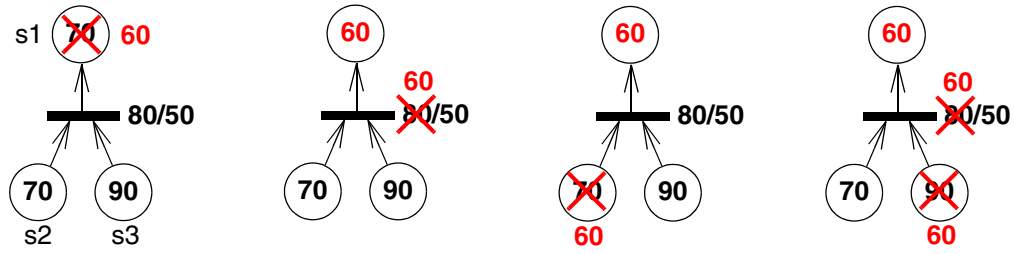
4.2 Statistische Adaption

Der zu entwerfende Adaptionsalgorithmus hat zwei grundlegende Probleme zu lösen, das der *Uneindeutigkeit* und das der *Widersprüchlichkeit*.

Wenn der Benutzer den Fuzzy-Wert einer Patterninstanz ändert, wird diese Änderung in Form eines benutzerdefinierten Fuzzy Belief Markings an die zugehörige Stelle im FPN weitergegeben. Um diesen Wert auch für den vom FPN errechneten Belief zu erhalten, müssen gemäß Abschnitt 2.3 die Vertrauenswerte im FPN entsprechend angepasst werden. Hierbei ergibt sich das Problem der Uneindeutigkeit. Die erforderlichen Anpassungen lassen sich nicht eindeutig aus dem Korrekturwert und dem FPN ableiten. Vielmehr gibt es zahlreiche unterschiedliche Realisierungsmöglichkeiten für jede Benutzerkorrektur.

Abbildung 4.1 zeigt einige dieser Möglichkeiten am Beispiel der Senkung des Fuzzy-Wertes. In Abbildung 4.1a ist die Situation nach der Benutzereingabe dargestellt. Der Wert des Knotens *s1* ist durch den Benutzer von 70 auf 60 gesenkt worden. Eine mögliche Realisierung dieser Änderung ist die lokale Anpassung des Vertrauenswertes der Transition wie in Abbildung 4.1b dargestellt. Eine andere Möglichkeit ist die Anpassung des Wertes eines oder mehrerer Teilpattern, die im FPN durch Vorbereichsstellen repräsentiert werden, wie in Abbildung 4.1c gezeigt. Abbildung 4.1d schließlich zeigt, daß auch beliebige Kombinationen dieser Realisierungen möglich sind.

Widersprüche ergeben sich, wenn für eine anzupassende Größe mehrere unterschiedliche Korrekturwerte vorliegen. Dies ist der Fall, wenn die Korrekturen verschiedener Benutzereingaben gemäß Abbildung 4.1c durch eine gemeinsame Vorbereichsstelle realisiert werden sollen. Derartige Widersprüchlichkeiten können häufig aufgelöst werden, indem andere Realisierungs-



a) Benutzereingabe *b) lokale Korrektur* *c) Teilpattern* *d) mehrere Knoten*

Abbildung 4.1: Realisierungsmöglichkeiten

möglichkeiten für die Korrekturen gewählt werden. Ist dies nicht möglich, so ist der resultierende Wert so zu wählen, daß der Fehler hinsichtlich aller beteiligten Benutzereingaben minimal ist.

Um eine globale Fehlerminimierung zu erreichen, müssen Uneindeutigkeiten und Widersprüche unter Berücksichtigung aller Benutzereingaben in sämtlichen gesammelten Lernmustern aufgelöst werden. Eine Ansatz, hierfür eine optimale Lösung zu finden, wäre die gleichzeitige Verfolgung aller Realisierungsmöglichkeiten, die sich aus den Lernmustern ergeben. Anschließend könnte dann die Lösung gewählt werden, die den geringsten Fehler in Bezug auf die Benutzereingaben erzeugt. Dieser Ansatz ist jedoch in der Praxis nicht sinnvoll umsetzbar, da die Anzahl der zu betrachtenden Varianten mit der Anzahl der FPN-Stellen und Benutzerkorrekturen in den Lernbeispielen exponentiell wächst und daher nur für sehr kleine Beispiele praktikabel wäre.

Stattdessen wird im Folgenden ein heuristisches Verfahren vorgestellt, das auf der Idee der gleichzeitigen Verfolgung der möglichen Realisierungen basiert, jedoch die problematische Einzelbetrachtung der verschiedenen Realisierungsmöglichkeiten durch eine einfache und effiziente statistische Auswertung ersetzt.

Hierzu werden für jede Benutzerkorrektur die erforderlichen Zielwerte an alle Stellen und Transitionen im FPN weitergegeben, an denen eine Realisierung der erforderlichen Anpassung möglich ist. Betrachtet man die so aufbereiteten Änderungsdaten mehrerer FPNs an den zugehörigen Implikationen des zugrundeliegenden PDNs, so steht zu erwarten, daß sich an den für eine Anpassung am besten geeigneten Implikationen ein entsprechender Zielwert aus diesen Daten herauskristallisiert. Andererseits werden nachteilige Änderungen durch entsprechende Gegenbeispiele in den Änderungsdaten unterdrückt.

Das Optimierungsproblem wird somit auf mehrere lokale Probleme reduziert, die anschließend einfacher zu lösen sind. Dabei wird ausgenutzt, daß die anzupassenden Werte des PDN in den FPNs in der Regel mehrfach vorkommen, so daß eine entsprechend große und damit zuverlässige statistische Basis für die Auswertung zur Verfügung steht.

Das bereits geschilderte Problem der Uneindeutigkeit betrifft in besonderem Maße die hier angestrebte Realisierung eines Lernansatzes als One-Shot-Lösung, da eine einmal getroffene

Entscheidung nicht auf der Basis späterer Erkenntnisse revidiert werden kann. In diesem Zusammenhang stellt die Kopplung der Vertrauenswerte mit den Schwellwerten bei der Berechnung der Fuzzy Beliefs nach Definition 2.7 ein besonderes Problem dar. Denn die Ablehnung oder nachträgliche Akzeptierung einer Musterinstanz kann sowohl durch Anpassung des Vertrauenswertes als auch durch Änderung des Schwellwertes erreicht werden. Zudem ist es für die Aufbereitung der Korrekturdaten von Nachteil, daß Fuzzy Belief-Werte durch vorangegangene Schwellwertbildungen maskiert werden, da so die Auswirkungen von Änderungen schwer zu beurteilen sind. Daher wird für die folgenden Betrachtungen eine Vereinfachung vorgenommen, die die getrennte Behandlung von Vertrauens- und Schwellwerten erlaubt. Hierzu wird die Berechnung des *Fuzzy Truth Tokens* aus Definition 2.7 geändert zu

$$\mathrm{ftt}(t) = \mathrm{Min}\{\{\mathrm{cf}(t)\} \cup \{\mathrm{fbm}(s) | s \in \mathrm{pre}(t)\}\} \quad \textbf{(4.1)}$$

Zusätzlich wird ein *Fuzzy Threshold Token*

$$\mathrm{fth}(t) = \begin{cases} 0 & \text{falls } \exists s \in \mathrm{pre}(t): \mathrm{fbm}(s) < \mathrm{th}(t) \vee \mathrm{fth}(s)=0 \\ 1 & \text{sonst} \end{cases} \quad \textbf{(4.2)}$$

eingeführt, das zusammen mit dem FTT berechnet wird und die Schwellwertbildung realisiert. Die Stellen des FPN halten den durch das FTT gegebenen Belief-Wert und die Schwellwertinformation des FTH getrennt. Der Fuzzy-Wert einer Musterinstanz ist damit gegeben durch den (nicht schwellwertbehafteten) Fuzzy Belief der zugehörigen Stelle falls das FTH der Stelle den Wert 1 hat und wird ansonsten zu 0. Definition 2.6 ist entsprechend anzupassen.

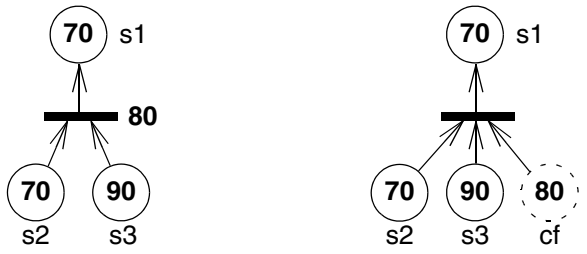
Die getrennte Betrachtung der Vertrauens- und Schwellwerte ist zulässig, da zwar beide Größen in die Berechnung der Fuzzy-Werte eingehen, jedoch ihre semantische Bedeutung grundlegend unterschiedlich ist. Während der Vertrauenswert die eigentliche Bewertung realisiert, dient der Schwellwert als Filter, der unwahrscheinliche Analyseergebnisse aussortiert. Konzeptionell kann dieser Filter aber ebensogut nach der Bewertung angewendet werden, statt in sie integriert zu sein. Daher kann auch bei der Anpassung der Vertrauens- und Schwellwerte eine getrennte Behandlung der beiden Größen vertreten werden, auch wenn dadurch bei der Adaption die Wechselwirkungen dieser beiden Werte vernachlässigt werden.

Im Folgenden wird deshalb der Lernansatz zunächst am Beispiel der Vertrauenswerte entwikkelt, bevor das Verfahren dann in Abschnitt 4.6 auf die Schwellwerte angewendet wird. Zunächst wird im folgenden Abschnitt die Aufbereitung der Korrekturdaten eines einzelnen Lernmusters, gegeben durch ein FPN und eine Menge von Benutzerkorrekturen, betrachtet. Anschließend wird die Anpassung durch Zusammenführung und statistische Auswertung der aufbereiteten Daten erläutert.

4.3 Adaption der Vertrauenswerte

Durch die getrennte Auswertung von Vertrauens- und Schwellwerten bei der Berechnung der Fuzzy Truth Tokens können die Schwellwerte in diesem Schritt zunächst vernachlässigt werden.

Daher kann die Berechnung des Fuzzy Beliefs aus Definition 2.7 in der vereinfachten Form aus Gleichung (4.1) erfolgen. Der Vertrauenswert einer Transition wird damit bei der Berechnung des FTT nicht mehr anders behandelt, als die Werte aus ihrem Vorbereich. Für die Formulierung des Adaptionsalgorithmus bietet es sich daher zur Vereinfachung an, den Schwellwert ebenfalls genauso zu behandeln, wie die Fuzzy-Werte der Stellen im FPN. Dazu werden zusätzliche Stellen in das FPN eingefügt, die die Schwellwerte repräsentieren und als Bias-Stellen bezeichnet werden. An dem Beispiel in Abbildung 4.2 ist zu sehen, daß die neu hinzugefügte Bias-Stelle cf mit dem Schwellwert der Transition initialisiert ist und die Schwellwert-Angabe an der Transition dafür fehlt.



a) Transition im FPN *b) mit Bias-Stelle cf*

Abbildung 4.2: Vertrauenswerte als Bias-Stellen

Für das so vereinfachte FPN lassen sich nun recht einfach zwei Bedingungen angeben, die eine Stelle *s* erfüllen muß, um einen vom Benutzer vorgegebenen Wert *x* anzunehmen. Diese Bedingungen folgen unmittelbar aus der Minimierungseigenschaft in Gleichung (4.1).

Damit *s* den Wert *x* annehmen kann, muß es mindestens eine Stelle im Vorbereich von *s* geben, die ebenfalls den Wert *x* hat. Darüber hinaus darf es keine Werte kleiner als *x* im Vorbereich von *s* geben. Es ergibt sich also:

$$\exists s' \in \mathrm{pre}_S(s) : \mathrm{fbm}(s') = x \qquad \textbf{(4.3)}$$

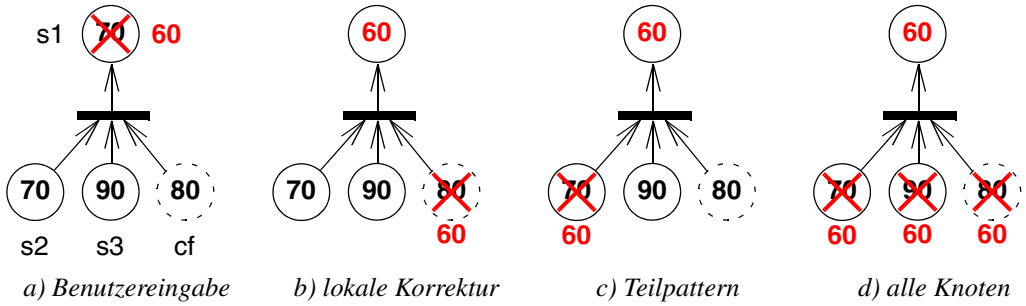
$$\forall s' \in \mathrm{pre}_S(s) : \mathrm{fbm}(s') \geq x \qquad \textbf{(4.4)}$$

Diese beiden Bedingungen können auch als Handlungsanweisungen für die Umsetzung der Benutzerkorrekturen aufgefaßt werden und bilden damit die Grundlage für die im Folgenden beschriebenen Adaptionsverfahren.

Bei der Adaption eines Vertrauenswertes muß zwischen dem Senken und Erhöhen des Wertes unterschieden werden. Die in beiden Fällen angewandten Verfahren sind zwar nahezu identisch, jedoch gibt es einige Unterschiede in der Motivation des Vorgehens. Daher wird im folgenden Abschnitt zunächst das Verfahren zum Senken des Vertrauenswertes beschrieben, bevor darauf aufbauend in Abschnitt 4.3.2 das Erhöhen des Vertrauenswertes behandelt wird. Daneben hat der Benutzer noch die Möglichkeit, einen Vertrauenswert als korrekt zu bestätigen. Dies wird in Abschnitt 4.3.3 behandelt.

4.3.1 Senken des Vertrauenswertes

Bei der Senkung des Vertrauenswertes ergibt sich, wie im Beispiel aus Abbildung 4.1 bereits erwähnt, das Problem, daß nicht eindeutig bestimmt werden kann, welche Werte zur Realisierung dieser Änderung angepasst werden müssen. Abbildung 4.3 zeigt nochmal einige resultierende Möglichkeiten in der Darstellung mit Bias-Stellen. Es ist zu sehen, daß die lokale Anpassung des Vertrauenswertes der Transition in Abbildung 4.3b keinen Sonderfall mehr gegenüber den Anpassungen in Abbildung 4.3c und d darstellt.



a) Benutzereingabe *b) lokale Korrektur* *c) Teilpattern* *d) alle Knoten*

Abbildung 4.3: Realisierungsmöglichkeiten

Aus Gleichung (4.3) folgt, daß mindestens einer der Vorbereichswerte der geänderten Stelle angepaßt werden muß. Zu diesem Zeitpunkt kann jedoch nicht entschieden werden, welche der verfügbaren Möglichkeiten im Kontext aller Lernmuster die optimale Variante ist. Daher wird zunächst keine Anpassung im FPN durchgeführt. Stattdessen wird der neue Wert als Änderungsanforderung an alle Stellen weiterpropagiert, die diese Änderung potentiell realisieren könnten, also nach Gleichung (4.3) an alle Stellen im Vorbereich des geänderten Knotens, wie in Abbildung 4.3d dargestellt. Mit diesen Informationen kann dann nach der Auswertung aller gesammelten Lernmuster eine Entscheidung zur Anpassung der beteiligten Vertrauenswerte im PDN getroffen werden.

Durch die Propagierung an alle Vorbereichsstellen werden jedoch viele Änderungsanforderungen an Stellen weitergegeben, auf die sie bei einer optimalen Lösung keinen Einfluß hätten. Bei der späteren Auswertung der Änderungsanforderungen beeinflussen diese falsch propagierten Werte das Endergebnis negativ. Dieser negative Einfluß kann mit Hilfe einer einfachen Heuristik stark reduziert werden. Dazu werden Änderungsanforderungen bevorzugt an solche Stellen weitergegeben, für die die erforderliche Anpassung minimal ist, aufgrund der Minimierungseigenschaft (4.1) also Stellen mit niedrigem Fuzzy Belief.

Die Bevorzugung geschieht durch eine *Gewichtsfunktion*, die jeder Stellen-Transitions-Kante im FPN ein Gewicht aus dem Intervall [0, 1] zuordnet. Das Gewicht ist abhängig von der Differenz zwischen dem Fuzzy Belief Marking der Ausgangsstelle und dem der Stelle aus dem Nachbereich der Transition und steigt mit abnehmender Differenz an. Darüber hinaus wird jeder Änderungsanforderung ein Gewicht zugeordnet, das zunächst für die vom Benutzer durchgeführte Korrektur mit dem Maximalwert Eins initialisiert ist. Bei der Propagierung der Änderungsanforderungen wird dieses Gewicht mit dem Kantengewicht der verfolgten FPN-Kante

multipliziert, so daß Änderungsanforderungen an bevorzugte Stellen ein hohes Gewicht behalten, während Anforderungen an Stellen mit starker Abweichung von der Ursprungsstelle durch ein niedrigeres Gewicht penalisiert werden. Das Gewicht der Änderungsanforderungen drückt in diesem Kontext gewissermaßen die Unsicherheit darüber aus, ob eine Anforderung auf die jeweilige Stelle Einfluß haben sollte.

Diese Heuristik basiert auf der Annahme, daß mit hoher Wahrscheinlichkeit bei der initialen Festlegung der Vertrauenswerte während der Musterspezifikation nur ein verhältnismäßig geringer quantitativer Schätzfehler gemacht wird. Im Gegensatz dazu ist ein Fehler bei der qualitativen Einordnung der Musterspezifikationen zueinander, also der Einschätzung, ob eine Spezifikation qualitativ vertrauenswürdiger ist, als eine andere, deutlich unwahrscheinlicher. Diese Annahme deckt sich auch mit Erfahrungen aus dem praktischen Einsatz des Inferenzprozesses aus Kapitel 2. Die Idee der Bevorzugung der wahrscheinlichsten Lösung hat sich auch in vielen anderen Lernansätzen bewährt, insbesondere in Form sogenannter *Maximum-Likelihood Schätzer*[CM98], bei denen ein zu optimierender Wert so gewählt wird, daß die Wahrscheinlichkeit eines korrekten Ergebnisses hinsichtlich einer gegebenen Wahrscheinlichkeitsverteilung maximiert wird.

Die für die Gewichtung zu verwendende Gewichtsfunktion ist nicht eindeutig vorgegeben. Es gibt jedoch eine Reihe von Rahmenbedingungen, anhand derer eine geeignete Funktion ausgewählt werden kann. Die Gewichtsfunktion muß für die Differenz zwischen den Vertrauenswerten von Vor- und Nachbereichsstelle auf das Intervall [0,1] beschränkt sein und monoton fallen. Zudem ist es sinnvoll, eine Funktion zu wählen, die zu Beginn relativ flach verläuft, um kleine Abweichungen nicht zu überbewerten. Eine Funktion, die diese Kriterien erfüllt und auch häufig in Maximum-Likelihood-Ansätzen auftritt, ist die logistische Funktion

$$\mathrm{lf}:\mathbb{R} \to [0,1],\ \mathrm{lf}(x) = \frac{1}{1 + e^{-B(x - M)}} \qquad \textbf{(4.5)}$$

mit geeignetem Wachstumsfaktor B und Wendepunkt M. In praktischen Tests hat sich diese Funktion bewährt. Mit B=-0,2 und M=25 ergibt sich die Gewichtsfunktion

$$c:F \to [0,1],\ c((s,t)) = \mathrm{lf}(\Delta\mathrm{fbm}) = \frac{1}{1 + e^{0{,}2\cdot(\Delta\mathrm{fbm} - 25)}} \qquad \textbf{(4.6)}$$

beziehungsweise

$$c(s,s') = c((s,t)),\ t = \mathrm{pre}(s') \in T \qquad \textbf{(4.7)}$$

für eine Stellen-Transitions-Kante *(p, t)* und

$$\Delta\mathrm{fbm} = \mathrm{fbm}(s) - \mathrm{fbm}(\mathrm{post}(t)), \qquad \textbf{(4.8)}$$

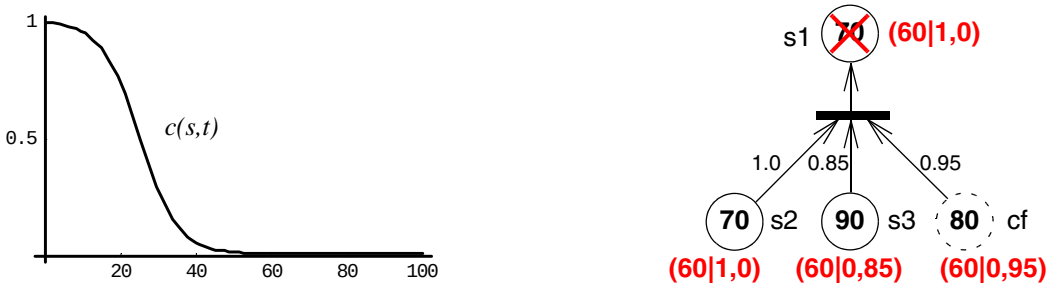
wobei zu beachten ist, daß *Δfbm* nach Gleichung (4.1) nie negativ wird. Abbildung 4.4a zeigt die Gewichtsfunktion für Differenzen *Δfbm* zwischen 0 und 100.

Mit diesen Gewichten ergibt sich damit für die zu propagierenden Änderungsanforderungen die folgende formale Gestalt.

Definition 4.1: Änderungsanforderung

Eine Änderungsanforderung (engl. change request) an eine Stelle $s \in S$ ist ein Tupel $(v|w) \in [0, 100] \times \mathbb{R}^+$ aus einem Vertrauenswert *v* und einem zugehörigen Gewicht *w*. Die Menge der Änderungsanforderungen $cr = (v|w)$ zu einer gegebenen Stelle *s* ist vollstängig beschrieben durch die Folge von Gewichten $w_{s,v} = w$.

Ein Beispiel für die Änderungspropagierung mit Gewichten ist in Abbildung 4.4b dargestellt. Die vom Benutzer durchgeführte Änderung des Fuzzy Beliefs von *s1* von 70 auf 60 wird mit dem Gewicht von 1,0 initialisiert. Die Kanten von den Vorbereichsstellen von *s1* zur Transition sind mit Gewichten[1] versehen, die bei der Propagation des Tupels (60|1,0) mit dem Gewicht der Änderung multipliziert werden. Es ist zu sehen, daß Änderungsanforderungen an die Stellen *s2* und *cf*, deren Werte näher an dem ursprünglichen Wert von *s1* liegen, gegenüber der an *s3* höher gewichtet werden. Es ist auch zu sehen, daß kleine Abweichungen von dem ursprünglichen Wert von *s1* nur wenig „bestraft“ werden (*cf*), während die Gewichtsabnahme für größere Abweichungen zunimmt (*s3*).



a) Gewichtsfunktion *b) Propagation mit Gewichten*

Abbildung 4.4: Gewichtete Propagation der Vertrauenswerte

Die propagierten Änderungsanforderungen können an den Stellen, an die sie propagiert worden sind, wiederum genauso behandelt werden, wie Benutzerkorrekturen, lediglich mit den entsprechend reduzierten Gewichten. So ergibt sich eine rekursive Propagierungsvorschrift für das gesamte Teilnetz unterhalb der ursprünglich vom Benutzer geänderten Stelle. Stellen, für die bereits Benutzereingaben vorliegen, erhalten keine Änderungsanforderungen aus ihrem Nachbereich. Liegen für eine Stelle mehrere Änderungsanforderungen vor, so werden Anforderungen mit demselben Vertrauenswert zusammengefaßt, indem ihre Gewichte summiert werden. Anforderungen mit verschiedenen Vertrauenswerten werden erst bei der späteren statistischen Auswertung miteinander verrechnet (siehe Abschnitt 4.4).

Abbildung 4.5 zeigt das Beispiel aus Abbildung 4.4b in einem größeren Kontext und verdeutlicht die rekursive Propagierung im FPN. Die Stelle *s3* erhält hier zusätzlich zu der Änderungsanforderung von *s1* mit Vertrauenswert 60 eine weitere Anforderung von *s2* mit Vertrauenswert 50. Da sich die Vertrauenswerte der beiden Anforderungen unterscheiden, werden sie nicht

1. Zur Vereinfachung der Darstellung werden gerundete Werte verwendet

zusammengefaßt. Die Anforderungen werden von *s3* an *s5* weiterpropagiert. *s6* erhält sie jedoch nicht, da für diese Stelle eine Benutzereingabe vorliegt. Es ist zu sehen, daß an *s5* das Gewicht der Änderungsanforderung von *s2* durch die Multiplikation mit dem Kantengewicht weiter abgenommen hat. Die Anforderung von *s1* hat denselben Vertrauenswert, wie eine weitere von der Stelle *s4* stammende. Diese beiden Anforderungen werden daher zusammengefaßt, das resultierende Gewicht entspricht der Summe der Einzelanforderungen nach Multiplikation mit den Kantengewichten:

$$1{,}42 \;=\; (0{,}6 \cdot 0{,}95) + (1{,}0 \cdot 0{,}85) \qquad \textbf{(4.9)}$$

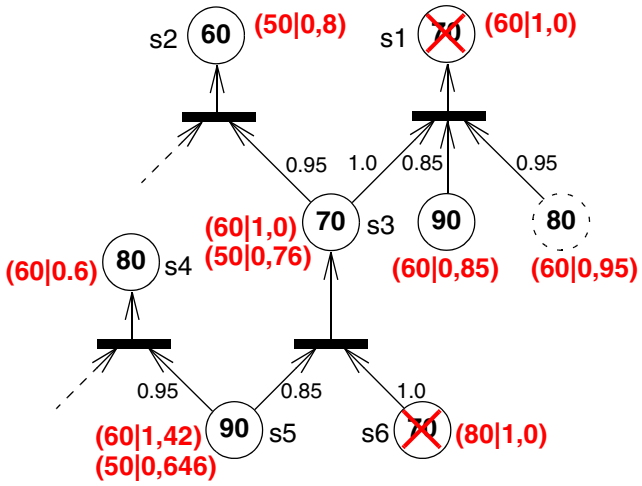


Abbildung 4.5: Propagierung der Änderungsanforderungen

Nach dieser informalen Erläuterung wird der Propagierungsalgorithmus nun formal beschrieben. Dazu ist es zunächst erforderlich, den Teilgraphen des FPN zu definieren, an den die Änderungsanforderungen einer Benutzereingabe propagiert werden.

Definition 4.2: Influenzbereich einer Änderungsanforderung

Der *Influenzbereich inf(s)* einer Stelle *s* ist die Menge aller Stellen im Teil-FPN unterhalb von *s*, ausgenommen solcher Teilgraphen, die von weiteren Stellen mit Benutzereingaben aufgespannt werden. Sei S_u die Menge der Stellen mit Benutzereingaben im FPN und $S_u(s)$ die Teilmenge von S_u im Teilgraph unter *s*

$$S_u \;=\; \{s' \mid s' \in S \wedge \mathrm{fbm}_u(s') \neq -1\}\,, \qquad \textbf{(4.10)}$$

$$S_u(s) \;=\; S_u \cap \mathrm{pre}_S{}^*(s)\,. \qquad \textbf{(4.11)}$$

Dann gilt für *inf(s)*

$$\mathrm{inf}{:}S \to P(S),\;\; \mathrm{inf}(s) \;=\; \mathrm{pre}_S{}^*(s) \backslash \mathrm{pre}_S{}^*(S_u(s)) \qquad \textbf{(4.12)}$$

Mit Hilfe des Influenzbereiches kann nun der Propagierungsalgorithmus formuliert werden.

Definition 4.3: Gewichtete Propagierung von Änderungsanforderungen

Seien s_i, $i \in [1, m]$ die Stellen des FPN, deren Fuzzy Belief vom Benutzer gesenkt worden ist. Ihre benutzerdefinierten Werte seien durch v_i, $i \in [1, m]$ gegeben. Die Änderungsaufforderungen mit Wert *v* an einem Knoten *s* sind zum Zeitpunkt *t* durch ihre Gewichte $w_{s,v,t}$ definiert.
Zum Zeitpunkt t=0 sind alle Gewichte mit 0 initialisiert. Für die Stellen s_i mit Benutzereingaben gilt

$$w_{s_i, v, t} = \begin{cases} 1 & \text{für } t \geq i,\ v = v_i \\ 0 & \text{sonst} \end{cases} \qquad \textbf{(4.13)}$$

Nach der Propagierung der Änderungsanforderungen der ersten *t* Stellen gilt für jede Stelle s aus dem Vorbereich der geänderten Stellen:

$$w_{s, v, t} = w_{s, v, t-1} + \sum_{s' \in post(s)} c(s, s') \cdot \Delta w_{s', v, t} \quad , s \in \bigcup_{i=1..t} inf(s_i) \qquad \textbf{(4.14)}$$

$\Delta w_{s',v,t}$ ist die Änderung an der Stelle *s'* durch die Propagierung der Änderungsanforderung von s_t, für t>0 gegeben durch

$$\Delta w_{s', v, t} = w_{s', v, t} - w_{s', v, t-1} \qquad \textbf{(4.15)}$$

Aus (4.14) und (4.15) folgt, daß nach der Propagation aller *m* Änderungsanforderungen gilt:

$$\bar{w}_{s, v} = w_{s, v, m} = \sum_{s' \in post_S(s)} c(s, s') \cdot \bar{w}_{s', v} \qquad \textbf{(4.16)}$$

Außerdem folgt aus der Zyklenfreiheit des FPN (siehe Abschnitt 2.3.1), daß der Algorithmus immer in endlicher Zeit terminiert.

Die bisherigen Überlegungen zur Verarbeitung der Benutzereingaben haben sich im wesentlichen auf die Bedingung (4.3) konzentriert. Zusätzlich hierzu kann die Bedingung (4.4) herangezogen werden, um die Ergebnisse weiter abzusichern.

Für eine Benutzerkorrektur, die den Fuzzy Belief einer Stelle *s* auf einen neuen Wert *x* senkt, müssen nach Gleichung (4.4) alle Stellen im Vorbereich von *s* einen Wert größer oder gleich *x* annehmen. Erhält eine Stelle *s'* aus dem Vorbereich von *s* neben der aus der Anpassung von *s* resultierenden Änderungsanforderung *cr*=(x|w) eine weitere Änderungsanforderung *cr'*=(x'|w'), deren Wert *x'* kleiner als *x* ist, so steht diese Anforderung *cr'* nach (4.4) im Widerspruch zu cr. Ist *x'* hingegen größer als *x*, so ist *cr* mit dieser Anforderung nach (4.4) konsistent und kann somit als Positivindikator für die Umsetzung von *cr'* in *s'* oder dessen Vorbereich interpretiert werden. Formt man Gleichung (4.4) wie folgt um

$$\forall s' \in pre_S(s): fbm(s') = x \vee fbm(s') > x \quad , \qquad \textbf{(4.17)}$$

so ist zu erkennen, daß dies dem zweiten Teil der Bedingung entspricht, während der erste Teil bereits durch die Änderungsanforderung cr ausgedrückt wird.

Es liegt nahe, in einem solchen Fall das Gewicht der konsistenten Änderungsanforderung *cr'* zu erhöhen. Hierzu wird entsprechend dem zweiten Teil von (4.17) zusammen mit jeder Ände-

rungsanforderung *cr*=(x|w) ein Endorsement-Gewicht *e* für das Intervall [x, 100] propagiert, das bei der späteren Auswertung zur Verstärkung aller Änderungsanforderungen in [x,100] verwendet wird, die an denselben Knoten wie *cr* auftreten. Da die obere Intervallgrenze fest mit 100 vorgegeben ist, reicht es für die Propagierung, die untere Grenze *x* zu propagieren. Berücksichtigt man ferner, daß die beiden Teile von (4.17) zueinander komplementär sind, so ist es naheliegend, auch den Wert von e als Komplement des Gewichts w der Änderungsanforderung anzunehmen:

$$e = 1 - w \tag{4.18}$$

Es ist zu beachten, daß die Propagierung der Endorsement-Gewichte zwar genauso erfolgt, wie für die Änderungsanforderungen beschrieben, jedoch mit sinkenden Gewichten für die Änderungsanforderungen der Wert der Endorsement-Gewichte gemäß (4.18) steigt.

Der Zusammenhang zwischen Änderungsanforderungen und Endorsement-Gewichten ist in Abbildung 4.6 beispielhaft zu sehen. Die Propagierung erfolgt genauso, wie in dem Szenario in Abbildung 4.5. Zusätzlich wird hier jedoch das Endorsement-Gewicht, dargestellt als dritter Wert des Tupels, mitpropagiert. Es ist zu erkennen, daß das Endorsement-Gewicht initial für Benutzereingaben mit 0 angesetzt ist (*s1*) und mit sinkendem Gewicht der Änderungsanforderung ansteigt (*s3*). Endorsement-Gewichte werden genauso wie die Gewichte der Änderungsanforderungen durch Addition zusammengefaßt, wenn die zugehörigen Vertrauenswerte übereinstimmen (*s5*). Bei der in Abschnitt 4.4 beschriebenen Auswertung verstärken die so propagierten Endorsement-Gewichte an jeder Stelle die Änderungsanforderungen, deren Vertrauenswert über dem der Endorsements liegt. Für die Stelle *s5* würde etwa die Änderungsanforderung *cr1* durch das mit *cr2* propagierte Endorsement-Gewicht von 0,35 verstärkt werden.

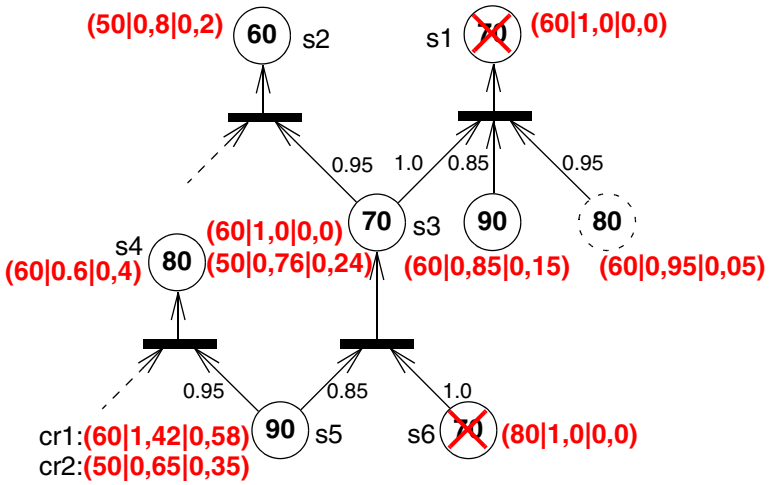


Abbildung 4.6: Propagierung mit Endorsements

Formal läßt sich dies wie folgt ausdrücken.

Definition 4.4: Endorsement-Gewicht

Das Endorsement *ew* für ein Intervall [v,100] an einer Stelle $s \in S$ ist gegeben durch das Tupel $ew = (v|e) \in [0, 100] \times \mathbb{R}^+$ aus einem Vertrauenswert *v* und einem zugehörigen Endorsement-Gewicht *e*. Die Menge der Endorsements $ew = (v|e)$ zu einer gegebenen Stelle *s* ist vollstängig beschrieben durch die Folge von Endorsement-Gewichten $e_{s,v} = e$. Analog zu Definition 4.3 gilt nach der Propagierung der t-ten Änderungsanforderung

$$e_{s, v, t} = e_{s, v, t-1} + \sum_{s' \in post(s)} (\Delta e_{s', v, t} + (1 - c(s, s')) \cdot \Delta w_{s', v, t}) \qquad \textbf{(4.19)}$$

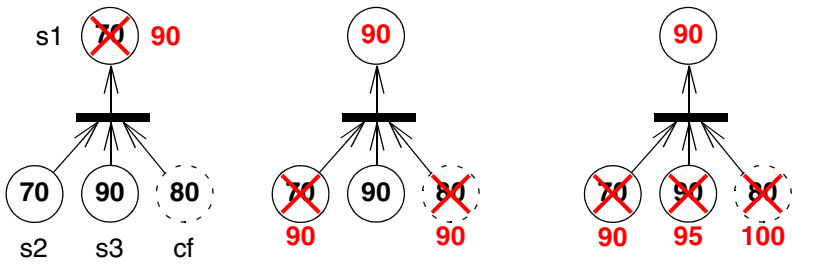
mit

$$\Delta e_{s', v, t} = e_{s', v, t} - e_{s', v, t-1} \qquad \textbf{(4.20)}$$

4.3.2 Erhöhen des Vertrauenswertes

Das Erhöhen der Fuzzy Beliefs durch den Benutzer wird weitestgehend analog zum Senken behandelt. Es sind wiederum zur Realisierung des benutzerdefinierten Wertes die Vertrauenswerte des FPNs geeignet anzupassen. Allerdings gibt es einige kleine Unterschiede, insbesondere in der Motivation der verwendeten Verfahren, auf die im Folgenden eingegangen werden soll.

Beim Senken der Fuzzy Beliefs ergibt sich nach Gleichung (4.3) das Problem, zu entscheiden, welche Stellen zur Umsetzung einer Benutzerkorrektur anzupassen sind. Dies ist in Abbildung 4.3 verdeutlicht. Betrachtet man im Vergleich dazu Abbildung 4.7b, so ist zu erkennen, daß durch Gleichung (4.4) die minimal anzupassenden Stellen direkt gegeben sind. Dies sind genau die Stellen mit einem Wert unter dem vom Benutzer vorgegebenen. Allerdings sind durch (4.4) die Werte nur nach unten beschränkt. Da in (4.3) nur gefordert ist, daß mindestens eine Stelle den Wert der Benutzereingabe annimmt, ist etwa auch das in Abbildung 4.7c dargestellte Szenario möglich.



a) Benutzereingabe *b) minimale Korrektur* *c) Alternative Korrektur*

Abbildung 4.7: Realisierungsmöglichkeiten

Zur Behandlung dieser Uneindeutigkeit werden die Änderungen des Benutzers wiederum als gewichtete Änderungsanforderungen durch das FPN propagiert. Während die Gewichte beim

Senken jedoch die Unsicherheit bezüglich der zu ändernden Stellen ausdrücken, beschreiben sie in diesem Fall die Unsicherheit über das Ausmaß der Erhöhung.

Eine Erhöhung über den vom Benutzer vorgegebenen Wert hinaus, wie sie Abbildung 4.7c für die Stellen s3 und cf zeigt, können allerdings nicht direkt aus dieser Benutzereingabe geschlossen werden. Stattdessen werden sie analog zum Vorgehen in Abschnitt 4.3.1 durch Endorsement-Gewichte unterstützt. Änderungsanforderungen aus Benutzereingaben mit höheren Vertrauenswerten werden hierdurch verstärkt.

Genauso kann für Vorbereichsstellen mit einem Wert über dem vom Benutzer vorgegebenen gemäß (4.4) aus der Benutzereingabe keine Indikation zum Senken dieser Stellen entnommen werden. Daher ist die für die Propagation von Erhöhungen zu verwendende Gewichtsfunktion gegenüber (4.7) leicht anzupassen, so daß Änderungsanforderungen an solche Stellen das Gewicht 0 erhalten:

$$c^{+}(s, s', v) = \begin{cases} c(s, s') & \text{falls } fbm(s) \le v \\ 0 & \text{sonst} \end{cases} \qquad \textbf{(4.21)}$$

v ist hierbei der vom Benutzer vorgegebene Wert. Für den zweiten Fall ist zu beachten, daß gemäß (4.18) das zugehörige Endorsement-Gewicht zu 1 wird.

Für Stellen s_i, $i \in [m+1, n]$ mit vom Benutzer erhöhten Werten folgt analog zu Definition 4.3 nach der Propagation der Benutzereingaben an den Stellen s_{m+1} bis s_t für Stellen s:

$$w_{s, v, t} = w_{s, v, t-1} + \sum_{s' \in post(s)} c^{+}(s, s', v) \cdot \Delta w_{s', v, t} \quad , s \in \bigcup_{i=1..t} inf(s_i) \qquad \textbf{(4.22)}$$

Nach der Propagierung aller Änderungsanforderungen haben damit die Erhöhungen einen Beitrag von

$$w^{+}_{s, v} = w_{s, v, n} - w_{s, v, m} = \sum_{s' \in post_S(s)} c^{+}(s, s', v) \cdot w^{+}_{s', v} \qquad \textbf{(4.23)}$$

an den Gesamtgewichten $w_{s,v,n}$. Somit lassen sich die Gesamtgewichte beschreiben durch

$$w_{s, v} = w_{s, v, n} = w^{-}_{s, v} + w^{+}_{s, v} \qquad \textbf{(4.24)}$$

$$w_{s, v} = \sum_{s' \in post_S(s)} (c(s, s') \cdot w^{-}_{s', v} + c^{+}(s, s', v) \cdot w^{+}_{s', v}) \qquad \textbf{(4.25)}$$

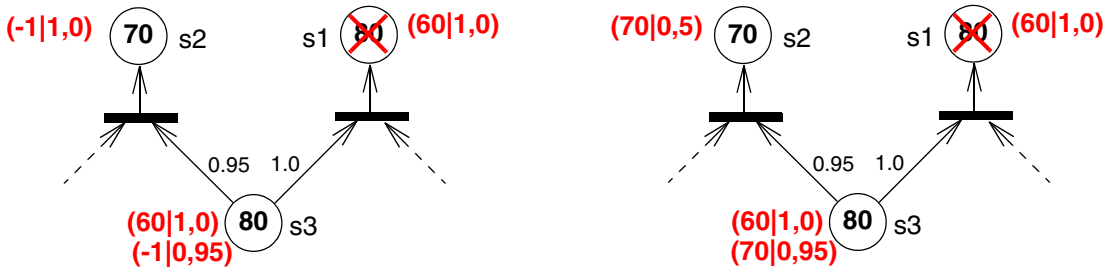
Die Endorsement-Gewichte werden mit der Gewichtsfunktion (4.21) wie in Definition 4.4 beschrieben propagiert.

4.3.3 Bestätigen des Vertrauenswertes

Neben dem Erhöhen und Senken von Vertrauenswerten kann der Benutzer auch korrekte Bewertungen von Musterinstanzen bestätigen. Eine solche Bestätigung sollte Änderungen im Vorbereich der bestätigten Stelle entgegenwirken, die den bestätigten Wert verändern würden.

Hierzu werden an den bestätigten Stellen spezielle „Änderungsanforderungen“ erzeugt und an deren Vorbereiche propagiert, die das Beibehalten der bisherigen Werte signalisieren. Diese Änderungsaufforderungen sind mit dem speziellen Vertrauenswert -1 außerhalb des normalen Wertebereiches von [0,100] markiert, der an jeder FPN-Stelle vor der Speicherung der Änderungsaufforderung durch den aktuellen Fuzzy Belief der Stelle ersetzt wird. Die Propagierung erfolgt analog zu der in Abschnitt 4.3.1 beschriebenen Senkung des Vertrauenswertes, so daß der gesamte Vorbereich der bestätigten Stelle durch die Änderungsanforderungen erfaßt wird.

Abbildung 4.8a stellt die Bestätigung eines Vertrauenswertes an einem Beispiel vor. Hier ist der Wert der Stelle *s1* vom Benutzer auf 60 herabgesetzt worden, während er den Wert der Stelle *s2* mit 70 bestätigt hat. Die Bestätigung wird in Form der Änderungsanforderung (-1|1,0) an den Vorbereich von *s2* weiterpropagiert wie in Abschnitt 4.3.1 beschrieben. Nach der in Abbildung 4.8b dargestellten Ersetzung des Sonderwertes -1 durch die jeweiligen Fuzzy Beliefs der Stellen ist zu erkennen, daß etwa der Änderung der Stelle *s3* auf den Wert 60, die als Seiteneffekt den die Änderung des Wertes von *s2* bewirkt, entgegengewirkt wird.



a) Bestätigung des Wertes *b) Auflösen der speziellen Markierungen*

Abbildung 4.8: Bestätigen von Fuzzy-Werten

4.4 Statistische Auswertung

Nachdem die im vorherigen Abschnitt beschriebene Propagierung von Änderungsanforderungen abgeschlossen ist, sind an jeder Stelle des Fuzzy Petrinetzes alle Daten vorhanden, die für die Auswertung der Benutzereingaben und anschließende Anpassung der Vertrauenswerte im PDN benötigt werden. Die Auswertung dieser Daten erfolgt in vier Schritten, die in den folgenden Teilabschnitten beschrieben werden. Am Ende des Auswertungsprozesses steht schließlich die Anpassung der Vertrauenswerte im PDN gemäß der ausgewerteten Daten.

4.4.1 Sammeln der Änderungsanforderungen

Da nach abgeschlossener Propagierung alle Änderungsanforderungen lokal an den Stellen des FPN verfügbar sind, ist die Struktur des FPN für die weitere Auswertung nicht mehr relevant. Die Auswertung und anschließende Anpassung der Vertrauenswerte geschieht im PDN. Hierzu müssen für jede Implikation des PDN die zugehörigen Korrekturdaten aus dem FPN gesammelt werden. Dies sind genau die Änderungsanforderungen und Endorsement-Gewichte an den Bias-Stellen des FPN, die wie in Abschnitt 4.2 beschrieben die Vertrauenswerte des PDN repräsentieren. Da bezüglich der in dieser Arbeit betrachteten Vertrauens- und Schwellwerte nur die Implikationen im PDN informationstragende Knoten sind, sind die Daten aller übrigen Stellen, die die Prädikate des PDN repräsentieren, für die weitere Betrachtung nicht von Bedeutung und stellen lediglich Zwischenergebnisse während der Propagierung dar.

Für eine Implikation *i* ergeben sich somit aus den Änderungsanforderungen $cr=(v|w_{cf,v})$ der Bias-Stellen *cf* zu *i* Gewichte $w_{i,v}$ mit

$$w_{i,v} = \sum_{cf \in inst(i)} w_{cf,v} \qquad \textbf{(4.26)}$$

und analog für Endorsement-Gewichte

$$e_{i,v} = \sum_{cf \in inst(i)} e_{cf,v} \ . \qquad \textbf{(4.27)}$$

In der Regel ist nach diesem Schritt kein Zugriff auf die einzelnen Daten im FPN mehr erforderlich. In diesem Fall kann ganz auf die Speicherung der propagierten Daten in den Stellen des FPN verzichtet werden. Stattdessen können diese bereits während der Propagierung an den entsprechenden Implikationen gesammelt werden. Damit ist der benötigte Speicherbedarf des Adaptionsverfahrens nicht mehr abhängig von der Größe des FPN, sondern nur noch von der Anzahl der Implikationen im PDN.

In einem optionalen zweiten Schritt können die soeben gesammelten Daten von Ausreißern bereinigt werden, um die Lerngenauigkeit zu erhöhen. Dieser Schritt ist für das Lernverfahren nicht zwingend erforderlich und wird daher als Erweiterung in Abschnitt 4.5.1 erläutert.

4.4.2 Zusammenführen der Lernmuster

Nachdem für das aktuelle Lernmuster, gegeben durch das vom Benutzer korrigierte FPN, alle Daten im PDN gesammelt und gegebenenfalls bereinigt sind, können sie mit den Daten älterer Lernmuster zusammengeführt werden. Sind bereits k-1 ältere Lernmuster vorhanden, so resultieren für die Gesamtgewichte w_T und Endorsements e_T die Werte

$$w_{Ti,v,k} = w_{Ti,v,k-1} + w'_{i,v} \qquad \textbf{(4.28)}$$

$$e_{Ti,v,k} = e_{Ti,v,k-1} + e_{i,v} \ . \qquad \textbf{(4.29)}$$

Diese Zusammenführung hat den Effekt, daß sich für zusätzliche Lernmuster lediglich die Gewichte erhöhen, jedoch keine Erhöhung des Speicherbedarfs entstehen. Die Gewichte w_T und e_T bieten sich somit zur Speicherung der gesammelten Lerndaten an, bis eine ausreichende Menge an Daten aus unterschiedlichen Reengineering-Sitzungen für die Anpassung vorhanden ist.

Das initiale Gewicht $w_{Ti,cf(i),0}$ für den bei der Musterspezifikation vergebenen Vertrauenswert cf(i) ist frei wählbar. Die Wahl dieses Gewichtes bestimmt jedoch die Lernrate, mit der sich initial die Vertrauenswerte ändern. Ein hohes initiales Gewicht bewirkt eine langsame Veränderung der Vertrauenswerte beziehungsweise erfordert entsprechend eine größere Anzahl an Lernmustern zur Veränderung. Ein niedriges Startgewicht resultiert hingegen in einer schnellen Revision des vom Reengineer bei der Spezifikation vergebenen Wertes. In der Praxis hat sich je nach Größe der analysierten Projekte ein Wert im Bereich von 5 bis 10 bewährt.

4.4.3 Adaption der Vertrauenswerte

Sobald eine ausreichende Menge unterschiedlicher Lernmuster gesammelt worden ist, können die Vertrauenswerte der Implikationen im PDN mit Hilfe der Lerndaten angepaßt werden. Ziel ist es, den Gesamtfehler über alle betrachteten Lernmuster zu minimieren. Hierzu ist für jede Implikation der Vertrauenswert so zu wählen, daß der Fehler im Bezug auf die an der Implikation gesammelten Änderungsanforderungen minimal wird. Unter der Annahme, daß die Änderungsanforderungen sinnvoll propagiert worden sind, verringert sich damit folglich auch der Gesamtfehler der Lernmuster.

Zuerst muß dafür jedoch die Berechnung der endgültigen Änderungsgewichte aus den Gewichten der Änderungsanforderungen und den Endorsements erfolgen. Es ist nicht sinnvoll, die Endorsements voll auf jedes Änderungsgewicht aus ihrem Einflußintervall aufzuaddieren, da hierdurch Endorsements mit niedrigem Vertrauenswert und damit großem Einflußintervall einen übermäßig großen Einfluß erhalten und die Gesamt-Gewichtsverteilung dadurch beeinträchtigen. Die Verstärkung der Änderungsanforderungen sollte vielmehr so erfolgen, daß die Endorsements anteilig auf die von ihnen beeinflußten Änderungsgewichte verteilt werden und das Verhältnis der geänderten Gewichte zueinander nicht durch die Anwendung dieses Endorsements verändert wird. Ein einzelnes Endorsement $e_{Ti,v'}$ wirkt sich somit auf die beeinflußten Änderungsgewichte $w_{Ti,v}$, v>v', wie folgt aus:

$$w'_{Ti,\,v} = w_{Ti,\,v} + \frac{w_{Ti,\,v}}{\sum_{k \geq v'} w_{Ti,\,k}} \cdot e_{Ti,\,v'} \qquad \textbf{(4.30)}$$

Hieraus folgt für ein Änderungsgewicht $w_{Ti,v}$ unter Berücksichtigung aller Endorsements $e_{Ti,v'}$ mit v'<v

$$w'_{Ti,\,v} = w_{Ti,\,v} \cdot \left[1 + \sum_{v' \leq v} \left(\sum_{u \geq v'} w_{Ti,\,u} \right)^{-1} \cdot e_{Ti,\,v'} \right] . \qquad \textbf{(4.31)}$$

Ist die Summe $w_{total}(i)$ der bisherigen Gewichte $w_{Ti,v}$ bekannt, so folgt durch Umformung die effizienter zu berechnende iterative Form

$$w'_{Ti,\,v} = w_{Ti,v} \cdot [1 + \omega(i, v)] \tag{4.32}$$

mit

$$\begin{aligned} \omega(i, v) &= \sum_{v' \le v} w_{sum}(i, v')^{-1} \cdot e_{Ti,\,v'} \\ &= \omega(i, v-1) + w_{sum}(i, v)^{-1} \cdot e_{Ti,v} \end{aligned} \tag{4.33}$$

und

$$\begin{aligned} w_{sum}(i, v) &= \sum_{u \ge v} w_{Ti,\,u} \\ &= w_{sum}(i, v-1) - w_{Ti,v-1} \\ w_{sum}(i, 0) &= w_{total}(i) \end{aligned} \tag{4.34}$$

Mit Hilfe der nach (4.32) neu gewichteten Änderungsanforderungen kann nun für jede Implikation *i* ein neuer Vertrauenswert $cf_{neu}(i)$ ermittelt werden. Dieser Vertrauenswert ist so zu wählen, daß der bezüglich der Änderungsanforderungen entstehende Fehler, ausgedrückt durch eine Fehlerfunktion $e:[0, 100] \times [0, 100] \to \mathbb{R}$, minimal ist. Mit dem Gesamtfehler *e(i,cf)* eines Vertrauenswertes *cf* an der Implikation *i*, gegeben durch

$$e(i, cf) = \sum_{v=0}^{100} w'_{Ti,\,v} \cdot e(v, cf) \tag{4.35}$$

folgt also für cf_{neu}

$$e(i, cf_{neu}(i)) = Min\{e(i, cf) \,|\, cf \in [0, 100]\} \tag{4.36}$$

Wie in Kapitel 3 beschrieben, wird als Fehlermaß häufig die quadratische Fehlerfunktion verwendet:

$$e(v, cf) = (v - cf)^2 \tag{4.37}$$

Für dieses Fehlermaß wird das Minimum in (4.36) genau dann angenommen, wenn $cf_{neu}(i)$ gleich dem gewichteten Mittel über die Änderungsanforderungen ist. Man wählt also

$$cf_{neu}(i) = \frac{\sum_{v=0}^{100} w'_{Ti,\,v} \cdot v}{\sum_{v=0}^{100} w'_{Ti,\,v}} \tag{4.38}$$

als neuen Vertrauenswert der Implikation *i*. Setzt man eine „vernünftige“ Propagierung der Änderungsanforderungen voraus, so wird durch diese neuen Vertrauenswerte auch der Fehler der verwendeten Lernmuster minimiert. Die Gültigkeit dieser Annahme und das Ausmaß des Lernerfolges werden in Kapitel 5 empirisch untersucht.

4.5 Erweiterungen

Die Ausführungen der vorangegangenen Abschnitte beschreiben bereits einen vollständigen Ansatz zur Anpassung der Vertrauenswerte mit Hilfe statistischer Auswertungen. In diesem Abschnitt werden zusätzlich noch zwei optionale Erweiterungen vorgestellt, die den Lernerfolg des Ansatzes weiter verbessern können, jedoch für den Adaptionsmechanismus nicht zwingend erforderlich sind.

4.5.1 Bereinigung der Datensätze

Nach der in Abschnitt 4.4.1 geschilderten Sammlung der Lerndaten an den Implikationen kann optional eine statistische Bereinigung der gesammelten Daten vor der weiteren Auswertung erfolgen. Ziel einer solchen Bereinigung ist die weitere Reduktion ungünstig propagierter Änderungsanforderungen, die sich als Ausreißer in der Menge der Datensätze manifestieren. Hierzu gibt es eine Vielzahl statistischer Standardverfahren[Sac04]. Ein häufig angewendetes Verfahren basiert auf der Standardabweichung σ als Maß für die Kohärenz der Datensätze. Die Standardabweichung einer Menge von gewichteten Werten x_k mit Gewichten w_k, $k \in [1, n]$ ist definiert als Quadratwurzel aus der Varianz σ^2 mit

$$\sigma^2 = \frac{\sum_{k=1}^{n} w_k \cdot (x_k - \bar{x})^2}{\sum_{k=1}^{n} w_k} \qquad \textbf{(4.39)}$$

für das gewichtete Mittel $\bar{x}$ der Werte x_k

$$\bar{x} = \frac{\sum_{k=1}^{n} w_k \cdot x_k}{\sum_{k=1}^{n} w_k} \; . \qquad \textbf{(4.40)}$$

Zur Bereinigung können nun alle weit außerhalb der mit Hilfe der Standardabweichung bemessenen Hauptmenge liegenden Datensätze entfernt werden:

$$w_k' = \begin{cases} w_k & \text{falls } |x_k - \bar{x}| \leq k\sigma \\ 0 & \text{sonst} \end{cases} \qquad \textbf{(4.41)}$$

Für *k* wird üblicherweise ein Wert im Bereich zwischen 1,5 und 2 verwendet [Sac04].

Um dieses Verfahren auf die Daten der Änderungsanforderungen anzuwenden, muß bei der Berechnung des Mittelwertes und der Standardabweichung die Veränderung der Änderungsgewichte durch die Endorsements berücksichtigt werden. Die Berechnung der Änderungsgewichte unter Berücksichtigung von Endorsements erfolgt wie in Abschnitt 4.4.3 erläutert, jedoch nur für die im ersten Schritt in Abschnitt 4.4.1 gesammelten Gewichte. Die resultierenden Gewichte für eine Implikation i seien gegeben als $\tilde{w}_{i,v}$.

Weiterhin ist zu beachten, daß Benutzereingaben mit den Werten v=0 und v=100 nicht als unscharfe Schätzwerte interpretiert werden, sondern eine absolute Ablehnung beziehungsweise Bestätigung darstellen. Obwohl diese Werte stets am äußersten Rand des Wertebereiches liegen,

ist ihre Entfernung im Rahmen der Bereinigung daher unerwünscht. Eine Menge solcher Ablehnungen und Bestätigungen kann jedoch in Form ihres Mittelwertes wieder als unscharfer Wert $\tilde{w}_{i,\,v'}$ in die Berechnung der Standardabweichung einfließen. Aus Gleichung (4.39) folgt daher insgesamt für die Varianz der Änderungsanforderungen einer Implikation i

$$\sigma_i^2 = \frac{\tilde{w}_{i,\,v'} \cdot (v' - \bar{v})^2 + \sum_{v=1}^{99} \tilde{w}_{i,\,v} \cdot (v - \bar{v})^2}{\hat{w}_{i,\,v'} + \sum_{v=1}^{99} \hat{w}_{i,\,v}} \quad , \tag{4.42}$$

$$\tilde{w}_{i,\,v'} = \tilde{w}_{i,\,0} + \tilde{w}_{i,\,100} \quad , \quad v' = \frac{\tilde{w}_{i,\,100} \cdot 100}{\tilde{w}_{i,\,v'}} \quad . \tag{4.43}$$

Mit Gleichung (4.42) kann nun die Bereinigung der ursprünglichen Gewichte $w_{i,\,v}$ für $v \in [1, 99]$ gemäß (4.41) erfolgen.

Ein weiterer Ansatz zur Bereinigung mit Hilfe der Varianz basiert auf deren Interpretation als Maß für die Datenkonsistenz. Konzentrieren sich alle Änderungsanforderungen an einer Stelle *s* auf einen kleinen Wertebereich, ist dies ein starkes Indiz für die Qualität dieser Anforderungen und für eine entsprechende Änderung des Wertes dieser Stelle. Eine hohe Streuung und damit eine hohe Varianz der Änderungsaufforderungen an einer Stelle *s* deutet hingegen darauf hin, daß für *s* zu viele zueinander inkonsistente Daten verfügbar sind, weshalb aus der Analyse dieser Daten keine gesicherten Schlüsse gezogen werden können. Durch eine zusätzliche Gewichtung aller Änderungsanforderungen einer Stelle *s* auf Basis der Varianz von *s* bei der in Abschnitt 4.4.1 beschriebenen Datensammlung können Stellen mit derartigen inkonsistenten Daten zugunsten von Stellen mit konsistenteren Daten im Gesamtergebnis unterdrückt werden. Es ist zu beobachten, daß hierdurch häufig solche Änderungen unterdrückt werden, die aus einer ungünstigen Propagierung im FPN resultieren.

Allerdings müssen für diesen Ansatz die Änderungsanforderungen an den Stellen des FPN verfügbar sein, was eine Speicheroptimierung wie in Abschnitt 4.4.1 beschrieben ausschließt und dadurch insbesondere für größere Projekte eine deutliche Erhöhung des Speicherbedarf bewirkt. Aus diesem Grund ist der Ansatz in dieser Form nur für kleine bis mittlere Projekte in der Größenordnung bis zu einigen wenigen zehntausend Zeilen Quelltext sinnvoll einsetzbar. Zur Wahrung der Speichereffizienz wird daher ein Ansatz verwendet, der statt der einzelnen Stellen im FPN die Menge aller Änderungsdaten $\tilde{w}_{i,\,v}$ einer Implikation nach der Varianz gewichtet:

$$\tilde{w}'_{i,\,v} = (1 - \sigma_{0i}^2) \cdot \tilde{w}_{i,\,v} \tag{4.44}$$

Als Gewicht σ_{0i}^2 wird hierbei die auf das Intervall [0,1] genormte Varianz der Implikation i verwendet mit

$$\sigma_{0i}^2 = \frac{\sigma_i^2}{\sigma_{max}^2} \quad , \quad \sigma_{max}^2 = 50^2 \tag{4.45}$$

Dieser Ansatz benötigt keine Daten im FPN, so daß eine Speicheroptimierung nach Abschnitt 4.4.1 möglich ist. Allerdings ist er wesentlich weniger feingranular bei der Gewichtung. Statt den Einfluß einzelner inkonsistente Stellen auf die Datenmenge einer Implikation zu reduzieren, wird die gesamte Datenmenge einer inkonsistenten Implikation zugunsten konsistenterer Daten aus weiteren Lernmustern benachteiligt.

4.5.2 Wahl der Ausgabestellen

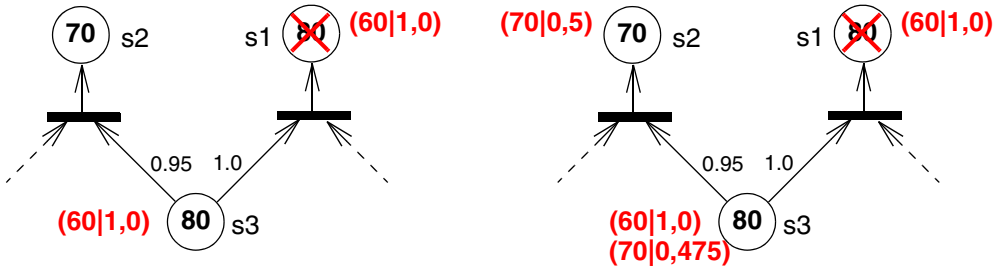
Bei allen bisherigen Betrachtungen in diesem Kapitel sind als für die Adaption betrachtete Ausgabestellen genau die Stellen des FPN gewählt worden, für die Benutzereingaben vorhanden sind. Häufig liegen jedoch nur Benutzereingaben für Stellen vor, deren Fuzzy Belief deutlich von dem Erwartungswert des Benutzers abweicht. Dementsprechend ist zumeist die Anzahl der vom Benutzer korrigierten Stellen deutlich niedriger, als die Anzahl der Musterannotationen, die tatsächlich für ihn von Interesse sind.

Insbesondere die Beliefs korrekt bewerteter Annotationen werden nur selten durch den Benutzer bestätigt. Daher stellt sich das Problem, daß es den Lernmustern an Gegenbeispielen mangelt, die unerwünschten Effekte von Benutzerkorrekturen auf andere Stellen entgegenwirken. Da nach der Definition des Regressionsproblems in Kapitel 3 auch nur Ausgabestellen in die Fehlerberechnung eingehen, haben diese Seiteneffekte bisher auch keinen negativen Einfluß auf den Fehler und damit auf den scheinbaren Lernerfolg, der diesen Fehler minimiert.

Eine Möglichkeit zur Lösung dieses Problems ist es, den Benutzer mit Hilfe einer geeignet gestalteten Benutzerschnittstelle zu weiteren Bewertungen zu bewegen, indem ihm auf der Basis der bisher von ihm evaluierten Patterninstanzen weitere Annotationen zur Begutachtung vorgeschlagen werden. Zusätzlich können weitere, nicht vom Benutzer korrigierte Stellen in die Menge der Ausgabestellen aufgenommen werden. Im einfachsten Falle können dies alle nicht vom Benutzer bewerteten Stellen ohne Nachfolger im FPN sein. Sinnvoller ist jedoch die Einbeziehung von Stellen an, die einen gemeinsamen Teilgraphen mit den vom Benutzer bewerteten Stellen teilen. Hierdurch werden insbesondere die erwähnten Seiteneffekte reduziert. Desweiteren bietet sich die Einbeziehung solcher Stellen an, die Annotationen von solchen Mustern repräsentieren, für die bereits eine oder mehrere Annotationen vom Benutzer bewertet worden sind. So ergibt sich ein vollständigeres Bild der für diese Muster erforderlichen Anpassungen.

Abbildung 4.9a zeigt das Problem der Seiteneffekte an einem Beispiel. Durch die Propagierung der Änderungsaufforderung von *s1* an die gemeinsame Vorbereichsstelle *s3* von *s1* und *s2* wird bei einer Änderung des zu *s3* gehörenden Vertrauenswertes auch *s2* beeinflußt. Diese Beeinflussung ist möglicherweise unerwünscht, was jedoch durch eine fehlende Bewertung von *s2* nicht erkennbar ist.

Für das in Abschnitt 4.3 vorgestellte Propagierungsverfahren bedeutet die Hinzunahme nicht bewerteter Ausgabestellen, daß diese Stellen so zu behandeln sind, als seien sie vom Benutzer bestätigt worden (siehe Abschnitt 4.3.3). Da durch die fehlende Benutzerbewertung jedoch



a) Nur bewertete Stellen *b) Mit weiteren Stellen*

Abbildung 4.9: Propagierung mit zusätzlichen Stellen

nicht sichergestellt ist, daß die Werte der zusätzlichen Ausgabestellen korrekt sind, ist das initiale Gewicht der Änderungsanforderungen entsprechend niedriger anzusetzen. Je nachdem, in welchem Umfang unbewertete Stellen in die Ausgabemenge aufgenommen werden, erweist sich ein Wert von 0,5 oder niedriger als angemessen.

In Abbildung 4.9b ist zu erkennen, daß die Propagierung der zusätzlichen Änderungsanforderung von *s2* (hier bereits nach der Ersetzung der aktuellen Fuzzy Beliefs) der Änderung von *s1* an *s3* entgegenwirkt, so daß eine Änderung an *s3* abgeschwächt wird.

Dieser Ansatz hat also den Vorteil, daß nicht alle Stellen vom Benutzer bewertet werden müssen, um eine ausreichende Menge an Gegenbeispielen zur Reduktion unerwünschter Seiteneffekte zu erhalten. Andererseits werden damit möglicherweise auch Änderungen behindert, für die diese Seiteneffekte erforderlich gewesen wären. Wäre etwa der Seiteneffekt an *s2* in obigem Beispiel erwünscht gewesen, so würde durch die Hinzunahme von *s2* in die Menge der Ausgabestellen der Lernerfolg reduziert werden. Ob und in welchem Umfang die Hinzunahme zusätzlicher Ausgabestellen sinnvoll ist, hängt daher stark davon ab, in welchem Umfang Korrekturen und Bestätigungen vom Benutzer zu erwarten sind. Kann davon ausgegangen werden, daß eine umfangreiche Bewertung der Analyseergebnisse stattfindet, kann auf die Hinzunahme weiterer Stellen wahrscheinlich verzichtet werden. Ist dagegen wie beschrieben zu erwarten, daß nur die gröbsten Fehler vom Benutzer korrigiert werden, so sollten weitere Stellen hinzugenommen werden.

4.5.3 Situationsspezifisches Wissen und gedächtnisbehaftetes Lernen

Ein Problem bei der Sammlung von Lernmustern besteht darin, daß durch unterschiedliche Implementierungsstile und Styleguides in verschiedenen Projekten oder sogar zwischen unterschiedlichen Programmierern die optimalen Vertrauens- und Schwellwerte voneinander abweichen können. Um trotzdem von den Daten anderer Projekte profitieren zu können, schlägt Strebin in [Str99] die Unterscheidung von *situationsspezifischem* und *Hintergrundwissen* vor. Dieser Ansatz läßt sich einfach auf das hier entwickelte Verfahren übertragen.

Für jede *Situation* (also etwa ein neues Projekt) werden die Lernmuster wie gehabt ausgewertet. Zusätzlich wird jeder Situation ein *Situationskoeffizient d* aus (0,1] zugeordnet, der angibt, wie

gut die Daten einer analysierten Situation auf andere Situationen übertragbar sind und wie gut allgemeines Hintergrundwissen auf diese Situation übertragen werden kann.

Für eine gegebene Situation mit Koeffizienten *d* geht das Hintergrundwissen w_H zu einer Implikation *i* damit wie folgt in die Auswertung der situationsspezifischen Daten $w_{Ti,v}$ ein:

$$w'_{Ti,\,v} = d \cdot w_{H,\,i,\,v} + (1 - d) \cdot w_{Ti,\,v} \qquad \textbf{(4.46)}$$

Umgekehrt werden bei Beendigung einer situationsspezifischen Analyse die gesammelten Daten gemäß dem Situationskoeffizienten zum Hintergrundwissen hinzugefügt:

$$w'_{H,\,i,\,v} = w_{H,\,i,\,v} + d \cdot w_{Ti,\,v} \qquad \textbf{(4.47)}$$

Ein weiteres Problem, das sich bei der Auswertung vieler Lernmuster ergibt, ist der mit der Anzahl verfügbarer Daten abnehmende Einfluß jedes neuen Datensatzes. Ist dies unerwünscht, zum Beispiel weil berücksichtigt werden soll, daß sich das System mit der Zeit (in kleinem Umfang) ändert, so muß ein Vergessen alter Daten realisiert werden. Nach [Str99] kann dies durch einen *Gedächtniskoeffizienten* β aus (0, 1) realisiert werden, über den der Anteil alter Daten bei der Zusammenführung der Lernmuster reduziert wird. Auch dieser Ansatz läßt sich direkt auf das hier vorgestellte Verfahren übertragen. Gleichung (4.28) wird damit zu

$$w_{Ti,\,v,\,k} = (1 - \beta) \cdot w_{Ti,\,v,\,k-1} + (1 + \beta) \cdot w'_{i,\,v} \qquad \textbf{(4.48)}$$

4.6 Adaption der Schwellwerte

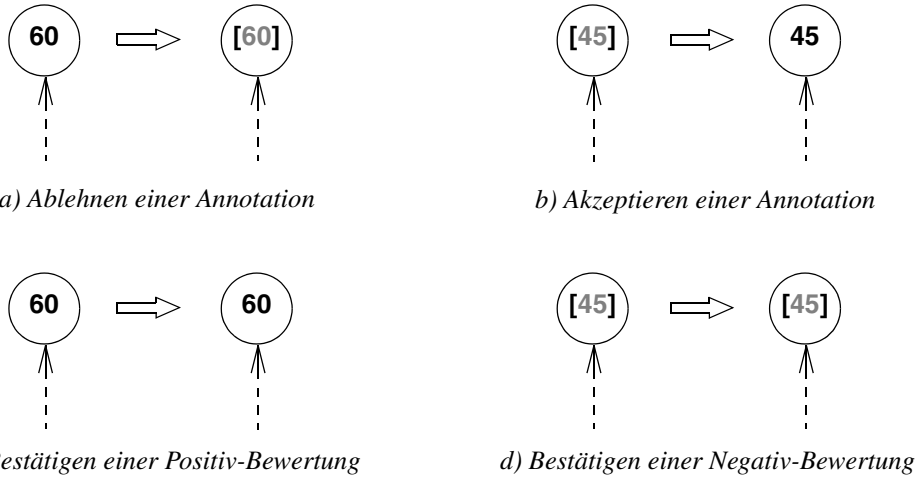
Bisher ist der Adaptionsalgorithmus ausschließlich am Beispiel der Vertrauenswerte vorgestellt worden. In diesem Abschnitt soll nun gezeigt werden, wie dasselbe Vorgehen zur Anpassung der Schwellwerte verwendet werden kann, bevor im nächsten Abschnitt auf Basis dieser beiden Beispiele das Verfahren verallgemeinert wird.

Der Schwellwert einer Musterspezifikation dient, wie bereits erwähnt, zur Ausfilterung von Annotationen, die mit einem niedrigen Fuzzy Belief bewertet worden sind. Der Reengineer kann den Schwellwert dazu verwenden, um eine Annotation abzulehnen, ohne ihren Fuzzy Belief zu ändern. Umgekehrt kann er auch Annotationen akzeptieren, die aufgrund eines niedrigen Beliefs vom Erkennungssystem abgelehnt worden sind. Im Gegensatz zur Adaption der Vertrauenswerte hat der Reengineer hier also nicht die Möglichkeit, einen Schwellwert direkt vorzugeben. Dies erweist sich als sinnvoll, da die Schwellwerte zum Filtern einer ohnehin bereits unscharfen Größe dienen und erheblich schwerer korrekt vom Reengineer abzuschätzen sind, als etwa die Vertrauenswerte. Das Akzeptieren und Ablehnen von Patternannotationen ist jedoch ausreichend, um daraus automatisch geeignete Ziel-Schwellwerte für die Musterspezifikationen im PDN abzuleiten.

Betrachtet man die vier in Abbildung 4.10 dargestellten Möglichkeiten von Benutzereingaben, so wird schnell klar, daß es bei deren Umsetzung in passende Schwellwerte ähnliche Probleme

hinsichtlich Unsicherheit und Widersprüchlichkeit gibt, wie in Abschnitt 4.3 für die Adaption der Vertrauenswerte geschildert. Auf die ersten beiden Fälle soll nun im Einzelnen eingegangen werden, die Bestätigung von Bewertungen wird etwas später behandelt.

Das Beispiel in Abbildung 4.10a zeigt die Ablehnung einer Annotation durch den Benutzer. Zur Realisierung dieses Szenarios ist es ausreichend, wenn der Schwellwert einer Transition im Vorbereich der abgelehnten Stelle höher ist, als das niedrigste Fuzzy Belief Marking in seinem Vorbereich (siehe Definition 2.7 und Gleichung (4.2)). Ähnlich dem Senken eines Vertrauenswertes ist jedoch aus der Benutzereingabe nicht erkennbar, welcher Schwellwert anzupassen ist, um diese Bedingung zu erfüllen.



a) Ablehnen einer Annotation

b) Akzeptieren einer Annotation

c) Bestätigen einer Positiv-Bewertung

d) Bestätigen einer Negativ-Bewertung

Abbildung 4.10: Benutzereingaben für Schwellwerte

Für die in Abbildung 4.10b dargestellte Akzeptierung einer zuvor vom Bewertungssystem abgelehnten Annotation müssen hingegen zur Realisierung alle Schwellwerte im Vorbereich der akzeptierten Stelle s unter den Fuzzy Beliefs ihrer jeweiligen Vorbereiche liegen. Denn nach Gleichung (4.2) würde bereits ein Fall, in dem ein Schwellwert im Teilbaum unter s nicht erreicht wird, für die Ausfilterung von s sorgen. Analog zum Erhöhen eines Vertrauenswertes sind daher alle bisher nicht erreichten Schwellwerte entsprechend zu senken. Die aktuellen Vertrauenswerte geben hierbei, wie im Falle der Vertrauenswert-Anpassung, nur die minimal erforderliche Korrektur vor, während die tatsächliche Korrektur stärker ausfallen kann.

Da der Benutzer die Schwellwerte nicht direkt vorgeben kann, steht im Gegensatz zu dem in Abschnitt 4.3 beschriebenen Ansatz die Zielgröße, hier also der gewünschte Schwellwert, nicht direkt für die Propagierung zur Verfügung. Der Schwellwert zu einer Transition kann jedoch aus den Fuzzy Beliefs abgeleitet werden, die diesen gemäß Benutzereingabe über- beziehungsweise unterschreiten sollen. Daher werden statt der nicht bekannten Schwellwerte die Fuzzy Beliefs der vom Benutzer akzeptierten oder abgelehnten Musterinstanzen als Änderungsanforderungen im FPN propagiert. Die Schwellwerte können damit dann so festgelegt werden, daß sie möglichst viele der propagierten Beliefs korrekt einordnen.

Da, wie soeben erläutert, die beiden betrachteten Fälle analog zum Senken und Steigern der Vertrauenswerte betrachtet werden können und dieselben Überlegungen zu Unsicherheit und Widersprüchlichkeit gelten, wie in Abschnitt 4.3 beschrieben, können die in Abschnitt 4.3 beschriebenen Propagationsvorschriften unmittelbar übernommen werden. Bei der Propagation werden jedoch Änderungsanforderungen $cr^-(v,w^-)$ zur Ablehnung und Anforderungen $cr^+(v,w^+)$ zur Akzeptierung an den Stellen s des FPN für die statistische Auswertung getrennt voneinander in Form von Gewichten $w^-_{s,v}=w^-$ und $w^+_{s,v}=w^+$gesammelt.

Da nur die aus der Benutzerbewertung resultierende Akzeptierung oder Ablehnung möglicher Musterinstanzen in die Änderungsanforderungen eingeht, nicht jedoch die ursprüngliche Situation, können die beiden bisher nicht betrachteten Fälle der Bestätigung einer Positiv- beziehungsweise Negativbewertung (Abbildung 4.10c und d) analog zur Akzeptierung respektive Ablehnung der Analyseergebnisse behandelt werden.

Weil zudem die neuen Schwellwerte einzig auf Basis der Ablehnung oder Akzeptierung der Fuzzy Beliefs bestimmt werden, sind die bisherigen Schwellwerte im FPN für die Propagierung nicht relevant und es kann wiederum die vereinfachte Bias-Darstellung aus Abschnitt 4.3 verwendet werden. Die Änderungsanforderungen zu jedem Schwellwert können dann bei der statistischen Auswertung nach Abschnitt 4.4 wiederum an den Bias-Stellen *cf* der zugehörigen Transitionen gesammelt werden. Durch die getrennte Betrachtung von Ablehnungs- und Akzeptierungsanforderungen ergeben sich nach der Sammlung entsprechend zwei Datenmengen:

$$\begin{aligned} w^+_{Ti,\,v} &= \sum_{cf \,\in\, inst(i)} w^+_{cf,\,v} \\ w^-_{Ti,\,v} &= \sum_{cf \,\in\, inst(i)} w^-_{cf,\,v} \end{aligned} \qquad \textbf{(4.49)}$$

Analog zur Berechnung der neuen Vertrauenswerte kann auf Basis dieser Daten ein neuer Schwellwert th_{neu} berechnet werden, der den Fehler gegenüber den gesammelten Änderungsanforderungen minimiert. Hierbei ist jedoch zu beachten, daß sich durch die Schwellwertbildung die zu verwendende Fehlerfunktion ändert. Für jeden Fuzzy-Wert v, der unter dem neuen Schwellwert liegen soll, entsteht für $v \geq th_{neu}$ ein Fehler in Höhe von v, da der Schwellwert den berechneten Fuzzy-Wert nicht wie gefordert auf 0 absenkt. Entsprechendes gilt für einen Fuzzy-Wert $v<th_{neu}$, der eigentlich über dem Schwellwert liegen soll. Die zu minimierende Fehlerfunktion für den Schwellwert *th* einer Implikation *i* ergibt sich somit zu

$$e(i,\,th) \;=\; \sum_{v \,\geq\, th} w^-_{Ti,\,v} \cdot v + \sum_{v \,<\, th} w^+_{Ti,\,v} \cdot v \qquad \textbf{(4.50)}$$

und th_{neu} ist so zu wählen, daß gilt

$$e(i,\,th_{neu}(i)) \;=\; \mathrm{Min}\{\Delta(i,\,th) \,|\, th \in [0,\,100]\} \qquad \textbf{(4.51)}$$

th_{neu} kann damit durch einen einfachen linearen Durchlauf der 101 möglichen Werte aus [0,100] bestimmt werden.

4.7 Verallgemeinerung

Bisher ist bei der Entwicklung des hier vorgestellten statistischen Adaptionsverfahrens lediglich die Anwendung auf den in Kapitel 2 geschilderten Inferenzprozess und die daran beteiligten Pattern Dependency Nets und Fuzzy Petrinetze betrachtet worden. Der dabei anhand der Adaption der Vertrauens- und Schwellwerte entworfene Algorithmus soll nun hinsichtlich seiner Anwendbarkeit auf andere Netzstrukturen untersucht und zu diesem Zweck verallgemeinert werden.

Initial ist die Frage zu beantworten, in welchen Situationen die Anwendung des geschilderten Ansatzes sinnvoll ist. Darüber geben die Entwurfskriterien aus Abschnitt 4.1 zumindest teilweise Aufschluß. Der Algorithmus ist speziell für die effiziente Bearbeitung großer Mengen von Lerndaten entworfen worden, bei denen Bearbeitungsgeschwindigkeit maximaler Genauigkeit der Lernresultate vorzuziehen ist. Für Situationen mit kleineren Datenmengen und höheren Anforderungen an die Lernresultate existieren geeignetere Ansätze (siehe Kapitel 3), die dem hier geschilderten vorzuziehen sind. Ein weiterer Indikator für den statistischen Ansatz ist eine sich von Lernmuster zu Lernmuster verändernde Systemfunktion, wie sie hier in Gestalt des FPN gegeben war. Schließlich ist das Mehrfachvorkommen der anzupassenden Größen in einem Lernmuster zu nennen, das wie in Kapitel 3 geschildert einige andere Ansätze vor Probleme stellt. Außerdem ist für den in Abschnitt 4.3 geschilderten Propagierungsalgorithmus die Terminierung nur sichergestellt, wenn die betrachteten Netze zyklenfrei sind. Ist dies nicht der Fall, kann der Ansatz jedoch unter Umständen trotzdem verwendet werden, wenn zusätzliche Maßnahmen zur Sicherstellung der Terminierung und einer sinnvollen Propagierung innerhalb der Zyklen ergriffen werden. Diese Maßnahmen liegen jedoch außerhalb des Rahmens dieser Arbeit.

Desweiteren ist zu berücksichtigen, daß der hier beschriebene statistische Ansatz eine große Menge an Lerndaten für jede anzupassende Größe benötigt, um eine repräsentative statistische Basis sicherzustellen. Aufgrund der in Abschnitt 4.4 beschriebenen Zusammenführung der Lerndaten ist es dabei unerheblich, ob diese Daten aus vielen unterschiedlichen Lernmustern stammen oder aus Mehrfachvorkommen der anzupassenden Größen in wenigen Lernmustern (vgl. jedoch Synergieeffekte in Abschnitt 5.2).

Ein weiteres wichtiges Kriterium für die Anwendung dieses Ansatzes ist ein nicht zu komplexer funktionaler Zusammenhang zwischen Vor- und Nachbereichsstellen in den betrachteten Netzen. Für die Rückpropagierung von Änderungsanforderungen von einem Knoten *k* des Netzes an alle Knoten, die den Wert von *k* beeinflussen, ist es erforderlich, bestimmen zu können, welche Möglichkeiten zur Anpassung dieser Knoten exisitieren, um den gewünschten Wert für den Knoten *k* zu erhalten. Im Falle der FPNs war dies durch die Minimierungsfunktion besonders

einfach. In anderen Fällen kann es jedoch je nach zugrundeliegender Berechnungsfunktion nicht oder nur schwer möglich sein.

Im Folgenden wird angenommen, daß diese Bedingungen erfüllt sind. Dann sei für einen Knoten k aus dem Vorbereich eines Knotens l der erforderliche Wert v, um an l den Wert u zu induzieren, gegeben durch die *Rückpropagierungsfunktion r* mit

$$r(l, k, u) = v \tag{4.52}$$

Ferner sei eine *Gewichtsfunktion c* gegeben, die für eine Änderungsanforderung cr=(u,w), die von l an k propagiert wird, ein Maß für die Sicherheit ist, daß k den Wert v=r(l,k,u) annehmen muß, um den Wert u in l zu realisieren. Eine geeignete Gewichtsfunktion kann in der Regel mit den in Abschnitt 4.3 angestellten Überlegungen und einigen empirischen Untersuchungen aus den betrachteten Netzstrukturen und funktionalen Zusammenhängen abgeleitet werden.

Bei der Propagierung einer Änderungsanforderung cr=(u|w) von einem Knoten l an einen Knoten k folgt damit für die resultierende Änderungsanforderung cr' an k (unter Vernachlässigung anderer propagierter Änderungsanforderungen)

$$cr' = (r(l, k, u) | c(l, k, u) \cdot w) \tag{4.53}$$

Für gegebene Funktionen r und c ändert sich damit die Propagierungsvorschrift (4.14) im allgemeinen Fall zu

$$w_{k, u, t} = w_{k, u, t-1} + \sum_{k' \in post(k)} c(k', k, v) \cdot \Delta w_{k', v, t} \tag{4.54}$$

$$\text{mit } v = r(k', k, u)$$

Die so propagierten Änderungsanforderungen können dann wie in Abschnitt 4.4 beschrieben ausgewertet werden, um den neuen Wert der zu Knoten k gehörigen Größe zu bestimmen.

KAPITEL 5

Evaluation

Der im vorangegangenen Kapitel hergeleitete statistische Adaptionsalgorithmus wird in diesem Kapitel empirisch untersucht. Dazu wird der in Kapitel 3 vorgestellte Backpropagation-Algorithmus als Referenzansatz für die Lösung von Regressionsproblemen zum Vergleich herangezogen. Beide Ansätze werden anhand von automatisch generierten Lernmustern auf ihre Lernerfolge und die erforderliche Laufzeit untersucht.

5.1 Vorgehen

Um gute Evaluationsergebnisse zu erhalten, ist es wichtig eine große und zugleich möglichst realistische Datenbasis von Lernbeispielen zu betrachten. Das heißt, daß viele Fuzzy Petrinetze mitt zugehörigen Korrekturdaten für die Evaluation erforderlich sind. Da jedoch die Analyse großer Systeme und insbesondere die spätere Auswertung durch den Benutzer sehr zeitaufwendig ist, ist es nicht praktikabel, eine solche Datenbasis in dem erforderlichen Umfang anhand echter Analyseergebnisse aufzubauen. Daher werden für die Evaluation von Lernenden-Systemen häufig Generatoren verwendet, die in der Lage sind, hinreichend realistische Lernbeispiele bestehend aus Eingabedaten und Systemantwort zu erzeugen.

In dem in Abbildung 3.1 dargestellten Schema eines Lernenden-Systemes ersetzt der Generator somit die Eingabekomponente und das System. Zudem bietet es sich an, zur Reproduktion der Evaluationsergebnisse und für weitere Analysen die vom Generator erzeugten Lernbeispiele zu speichern. Insgesamt ergibt sich somit das in Abbildung 5.1 dargestellte Schema.

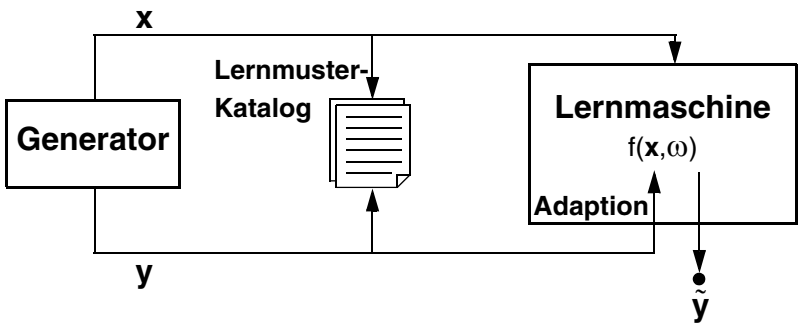


Abbildung 5.1: Generieren von Lernmustern

Ein solches Generatorkonzept wird im Folgenden für die Erzeugung von Fuzzy Petrinetzen vorgegebener Größe und zugehörige Korrekturdaten zu einem gegebenen PDN vorgestellt. Um die Struktur durch tatsächliche Analysen entstandener FPNs möglichst gut nachzuahmen, bietet es

sich an, für die Erzeugung ebenfalls den in Kapitel 2 vorgestellten Inferenzalgorithmus zu verwenden. Da hierzu kein Quelltext verfügbar ist, auf den die Musterspezifikationsregeln angewandt werden können, wird ihre Anwendung durch einen Zufallsprozess ersetzt, der über Erfolg oder Mißerfolg einer Regelanwendung entscheidet. Damit die so erzeugten Netze möglichst realistisch sind, können die für den Zufallsprozess erforderlichen stochastischen Kennwerte anhand realer Beispiele ermittelt werden. Dies sind im Einzelnen:

- Für jedes Axiom im PDN der erwartete prozentuale Anteil von Axiomsstellen an der Gesamtknotenzahl des FPNs, sowie die Varianz dieses Wertes. Hiermit kann die Anzahl der zu erzeugenden Axiomsstellen in Abhängigkeit von der gewünschten Netzgröße als normalverteilte Zufallsvariable modelliert werden.
- Für die Bottom-Up-Inferenz die bedingte Wahrscheinlichkeit $p_{up}(P|Q)$, daß eine Instanz eines Prädikates Q eine Instanz des von Q getriggerten Prädikates P in ihrem Nachbereich hat. Für eine in [0,1] gleichverteilte Zufallsvariable x gilt P dann als erfolgreich erkannt, wenn $x \leq p_{up}(P|Q)$ gilt[1].
- Für die Top-Down-Herleitung fehlender Prämissen Q zu einer erfolgreich „erkannten“ Musterinstanz s eines Prädikat P die bedingte Wahrscheinlichkeit $p_{down}(Q|P)$, daß eine Instanz von P eine Instanz von Q in ihrem Vorbereich hat.
- Für die Bindung bereits angelegter Patterninstanzen s eines Prädikates P als Prämissen weiterer Patterninstanzen die bedingte Wahrscheinlichkeit $p_{out}(x>d|P)$, daß der Ausgangsgrad der Stelle s größer als der aktuelle Grad $d=|post(s)|$ ist.

Diese Werte können durch einfaches Abzählen aus einer Reihe von Beispielanalysen realer Softwaresysteme ermittelt werden. Für p_{up} gilt etwa

$$p_{up}(P|Q) = \frac{|P_Q|}{|inst(Q)|} , \qquad \textbf{(5.1)}$$

mit der Menge P_Q von Instanzen von P mit einer Instanz von Q im Vorbereich:

$$P_Q = \{s|s \in inst(P) \wedge pre_S(s) \cap inst(Q) \neq \varnothing\} \qquad \textbf{(5.2)}$$

Die Ermittlung der übrigen Werte erfolgt analog.

Für die im Folgenden durchgeführten Versuche sind zur Bestimmung dieser Kennwerte die Mustererkennungsergebnisse mehrerer realer Systeme ausgewertet worden. Bei den betrachteten Systemen handelt es sich um die GUI-Bibliotheken AWT und Swing des Java Runtime Environment (ca. 70.000 und 150.000 LOC), sowie Teile des UML-Metamodells der Entwicklungsumgebung FUJABA (ca. 55.000 LOC). Bei dem für diese Analysen und die weitere Generierung von Lerndaten verwendeten Musterkatalog handelt es sich um den in [Wen01] beschriebenen Katalog mit den in [GHJV95] beschriebenen Entwurfsmustern (GOF-Pattern).

Neben der Erzeugung der FPNs ist der Generator auch für die Bereitstellung von simulierten Benutzereingaben verantwortlich. Hierzu werden für das betrachtete PDN neue Zielwerte für

1. Die zufällige Erkennung einer Musterinstanz ist also als Bernoulli-Experiment mit Erwartungswert p_{up} modelliert.

die Vertrauens- und Schwellwerte vorgegeben, entweder durch den Benutzer oder zufällig durch einen um den aktuellen Wert normalverteilten Zufallsprozess. Wertet man mit den neuen Vertrauens- und Schwellwerten die erzeugten FPNs aus, können die errechneten Fuzzy Beliefs als Sollwerte für Korrekturen an den FPNs mit den ursprünglichen Vertrauens- und Schwellwerten verwendet werden.

Um bei den im Folgenden betrachteten Versuchsdaten repräsentative Ergebnisse sicherzustellen, sind alle Versuche je fünfzig mal[1] mit unterschiedlichen Lernmustern und unterschiedlichen Zielwerten im PDN durchgeführt worden. Die aufgeführten Ergebnisse sind als arithmetische Mittel der Einzelergebnisse zu verstehen. Als Fehlermaß wird der aus Kapitel 3 bekannte mittlere quadratische Fehler (Mean Square Error, MSE) verwendet. Um die Fehlerwerte besser in Relation zu den Fuzzy Belief Werten setzen zu können, sind die aufgeführten Fehlerwerte die Quadratwurzeln der ursprünglichen, quadrierten Fehlerterme.

Messwerte zum Laufzeitbedarf sind mit Hilfe einer Profiler-Software ermittelt worden, so daß nur die tatsächlich von dem betrachteten System (der Inferenzmaschine beziehungsweise der Lernmaschine) verwendete Laufzeit in die Evaluation eingehen und die Ergebnisse nicht durch Hintergrundprozesse des Betriebssystems beeinflußt werden.

Als Testsystem kommt ein PC mit AMD AthlonXP 1900+ Prozessor mit 1,6GHz Taktzyklus und 1,5GB RAM zum Einsatz. Das verwendete Betriebssystem ist Microsoft Windows 2000.

5.2 Vergleich mit Fuzzy Neural Nets

Im Folgenden werden nun die mit dem in dieser Arbeit entwickelten statistischen Ansatz erzielten Lernergebnisse mit den Ergebnissen des etablierten Backpropagation-Ansatzes mit Fuzzy Neural Nets verglichen. Von Interesse hierbei sind insbesondere die für das Lernen benötigte Zeit und der erzielte Lernerfolg.

In einem ersten Versuch werden hierzu beiden Ansätzen FPNs unterschiedlicher Größe als Lernmuster übergeben, um das Verhalten beider Ansätze für verschiedene Mengen an zu erlernenden Korrekturwerten zu evaluieren. Für beide Ansätze kann die benötigte Rechenzeit in zwei Phasen unterteilt werden.

Die erste Phase ist die Aufbereitung der Lerndaten bei ihrer Erfassung, also im Anschluß an die Fuzzy-Bewertung und die Begutachtung durch den Benutzer. Bei dem Backpropagation-Ansatz wird hierbei das erzeugte FPN in Form mehrerer Lernmuster auf das FNN abgebildet. Beim statistischen Ansatz erfolgt in dieser Phase die Propagierung der Änderungsanforderungen. Da diese Phase online, also im Rahmen der Inferenz erfolgt, ist hier entscheidend, daß der verwendete Lernansatz die Dauer der Inferenz nicht signifikant erhöht. Tabelle 5.1 zeigt die Evaluationsergebnisse dieser Phase.

1. Bei stark zeitaufwendigen Versuchen zwanzig mal

FPN	Inferenz		Statistisch	FNN	
|FPN|	t_{inf}[s]	t_{fpn}[s]	t_{stat}[s]	#In	t_{FNN}[s]
20	10	0,37	0,74	3,6	1,61
100	120	1,91	4,21	22,8	9,92
1.000	900	20,4	52,7	296	181
10.000	2400	198	589	3847	2707
100.000	--[1]	1974	6416	50052	33131

1: Inferenzen dieser Größenordnung überschreiten die verfügbaren Systemressourcen der Testplattform

Tabelle 5.1 Aufbereitung der Lernmuster

Die erste Spalte der Tabelle zeigt die jeweilige Größenordnung der für das Training verwendeten FPNs. Die nächste Spalte (t_{inf}) gibt einen ungefähren Erfahrungswert für die typische Dauer des Inferenzprozesses für Analysen der gegebenen Größenordnung an. Die dritte Spalte (t_{fpn}) zeigt die durchschnittliche Auswertungsdauer eines FPNs der angegebenen Größe. Die von den beiden Ansätzen in dieser Phase pro FPN benötigte Zeit ist in den Spalten Vier (t_{stat}) und Sechs (t_{FNN}) aufgeführt. Zusätzlich enthält Spalte Fünf (#In) die durchschnittliche Anzahl der für das FNN aus jedem FPN erzeugten Lernmuster.

Es ist zu erkennen, daß bei beiden Lernverfahren die Aufbereitungsphase länger dauert, als die Auswertung des FPNs. Jedoch wächst die Laufzeit mit zunehmender Netzgröße im Fall der statistischen Auswertung der FPNs deutlich langsamer, als bei den Fuzzy Neural Nets. Für große FPNs mit etwa 10.000 Stellen, was einem analysierten System von mehreren 100.000 Zeilen Quelltext entspricht, liegt die Laufzeit im statistischen Fall damit immer noch etwa in derselben Größenordnung, wie die Auswertungszeit des FPN. Für die Fuzzy Neural Nets hingegen wird durch die Aufbereitung die Inferenzdauer mehr als verdoppelt. Außerdem ist zu erkennen, daß die Anzahl der vom FNN zu verarbeitenden Lernmuster mit steigender Netzgröße immer stärker anwächst.

Die zweite Phase des Lernprozesses beginnt für beide Ansätze sobald ausreichend Lernmuster gesammelt worden sind und der Benutzer die Anpassung der Vertrauens- und Schwellwerte auf Basis dieser Daten anstößt. Bei dem FNN-Ansatz findet nun der Backpropagation-Prozess mit den zuvor erzeugten Lernmustern statt. Für den statistischen Ansatz ist in dieser Phase hingegen lediglich die in Abschnitt 4.4 geschilderte statistische Auswertung im zugehörigen PDN erforderlich. Wie Tabelle 5.2 zeigt, ist daher die Laufzeit t_{stat} für das statistische Verfahren in dieser Phase unabhängig von der Anzahl und Größe der Lernmuster und im Vergleich zu den anderen Zeiten vernachlässigbar klein. Im Gegensatz dazu steigt die Laufzeit t_{FNN} des Backpropagation-Verfahrens im Neuronalen Netz für größere FPNs so stark an, daß bei FPN-Größen von 10.000 Stellen bereits die praktischen Grenzen der Versuchsdurchführung erreicht worden sind, so daß bei diesem Versuch auf die übliche Mittelung mehrerer Auswertungen verzichtet worden ist.

Der Lernerfolg der beiden Ansätze kann an der Verringerung des durchschnittlichen Fehlers abgelesen werden. Der Fehler e_{inf} vor Anwendung der Lernansätze ist in Spalte Vier von Tabelle

FPN		Inferenz		Statistisches Lernen		FNN	
|FPN|	FPNs	t_{inf}[s]	e_{inf}	t_{stat}[s]	e_{stat}	t_{FNN}[s]	e_{FNN}
20	10	10	27,3	3,06	17,31	15	3,61
100	10	120	23,9	2,98	9,84	193	2,92
1.000	10	900	24,1	3,11	7,13	3751	2,74
10.000	5	2400	26,1	3,09	6,75	65113[1]	2,81
100.000	5	--	24,7	3,14	6,81	--[2]	--

1: Nur ein Durchlauf, keine Mittelung mehrerer Versuche aufgrund zu hoher Laufzeit

2: Aufgrund zu erwartender Laufzeit nicht durchgeführt

Tabelle 5.2 Anpassung der Vertrauens- und Schwellwerte

5.2 aufgeführt. Er schwankt für alle betrachteten Beispiele um den Wert 25, weil für die Bestimmung der Sollwerte im PDN ein normalverteilter Zufallsprozess mit Standardabweichung 25 um den alten Vertrauenswert verwendet worden ist. Das Fuzzy Neural Net leistet in allen Fällen eine Reduzierung des Fehlers auf Werte im Bereich von 3,0. Der Lernerfolg ist hier bereits für wenige Lerndaten sehr gut und stabiliert sich schnell mit zunehmender Datenmenge.

Im Gegensatz dazu ist der Fehler e_{stat} nach Anwendung des statistischen Ansatzes offenbar stärker abhängig von der Menge der verfügbaren Lerndaten. Aufgrund des verwendeten statistischen Verfahrens ist dies auch logisch, da im Gegensatz zu dem iterativen Backpropagation-Verfahren falsche Annahmen zur Adaption nicht iterativ erkannt werden können, sondern von vornherein auf Basis der Lerndaten vermieden werden müssen. Es ist zu sehen, daß das Verfahren für wenige, sehr kleine Netze offensichtlich schlecht geeignet ist. Erst für große Netze mit tausend Stellen oder mehr stabilisiert sich das Verfahren. Die erreichte Fehlerreduktion reicht dabei zwar nicht an den FNN-Ansatz heran. Sie ist jedoch mit Fehlerwerten um 7,0 in einem Bereich, in dem die berechneten Fuzzy-Werte genau genug sind, um dem Reengineer als zuverlässige Richtwerte bei der Analyse zu dienen.

Betrachtet man die Abnahme des mittleren Fehlers mit zunehmender Stellenzahl der FPNs für den statistischen Ansatz, so stellt sich die Frage, ob die Abnahme einzig durch die Erhöhung der Gesamtmenge der Stellen hervorgerufen wird, oder ob auch bei insgesamt gleicher Stellenzahl große Netze besser für diesen Ansatz geeignet sind. Dieser Frage soll in einem zweiten Versuch nachgegangen werden. Dazu wird die Zahl der verwendeten FPNs für jede Netzgröße so gewählt, daß die Gesamtstellenzahl in allen Fällen in etwa gleich ist. Die Ergebnisse dieses Versuches sind in Tabelle 5.3 aufgeführt.

|FPN|	FPNs	e_{stat}
20	25000	11,41
100	5000	8,38
1.000	500	6,76
10.000	50	6,69
100.000	5	6,81

Tabelle 5.3 Fehler bei gleicher Stellenzahl

Offensichtlich sinkt für die kleinen Netze der Fehler mit zunehmender Anzahl betrachteter Lernmuster weiter ab. Er ist jedoch auch bei gleicher Gesamtstellenzahl noch deutlich über dem verbleibenden Fehler bei den größeren Netzen mit über 1.000 Stellen. Die Ergebnisse dieser Netze sind weitgehend stabil gegenüber dem vorangegangenen Versuch. Der statistische Ansatz erreicht also offensichtlich seine optimale Leistung erst für größere Fuzzy Petrinetze und ist damit besonders geeignet für den Einsatz bei der Analyse großer Softwaresysteme. Die bessere Leistung bei größeren Netzen kann damit erklärt werden, daß erst ab einer gewissen Netzgröße vermehrt Stellen mit einem Ausgangsgrad > 2 im FPN vorhanden sind. Hierdurch kommt es stärker zu Synergie-Effekten zwischen Stellen mit gemeinsamen Vorbereichen, als dies bei kleineren Netzen der Fall ist. Bei diesen Synergien handelt es sich vor allem um die in Kapitel 4 beschriebene Aufhebung ungünstiger Änderungsanforderungen und die Verstärkung zueinander konsistenter Änderungen durch Endorsements.

Trotz seiner für den praktischen Einsatz in der Analyse großer Softwaresysteme ungeeigneten Laufzeitanforderungen bleibt jedoch der FNN-Backpropagation-Ansatz bei der Reduktion des Fehlers dem statistischen Ansatz überlegen. Da der Backpropagation-Algorithmus ein iterativer Prozess ist, der auf Wunsch nach jeder Iteration abgebrochen werden kann, soll abschließend noch überprüft werden, ob dieser Ansatz bei vorzeitigem Abbruch eventuell auch hinreichend gute Ergebnisse in akzeptabler Zeit liefern kann. Hierzu wird in einem Versuch die Iteration abgebrochen, sobald der Fehler auf den mit dem statistischen Ansatz erreichten Wert gesunken ist. In einem weiteren Versuch wird die Trainingszeit auf die für den Inferenzprozess benötigte Zeit beschränkt. Tabelle 5.4 listet die Ergebnisse dieser Versuche auf.

|FPN|	**FPNs**	**e_{stat}**	**t_{FNN}[s]**	**t_{inf}[s]**	**e_{FNN}**
20	10	17,31	2,8	10	7,81
100	10	9,84	27,6	120	7,25
1.000	10	7,13	556	900	13,6
10.000	5	6,75	11098	2400	17,6

Tabelle 5.4 Fehler des FNN bei vorzeitigem Abbruch

In Spalte Drei und Vier von Tabelle 5.4 ist zu erkennen, daß die benötigte Berechnungszeit bis zum Erreichen von e_{stat} zwar deutlich niedriger ist, als die in Tabelle 5.2 aufgeführte Zeit bis zur Stabilisierung des Netzes. Sie liegt jedoch für große Netze immernoch deutlich über der für den Inferenzprozess benötigten Zeit. Und auch für die mittelgroßen Netze mit 100 und 1000 Stellen ist sie noch deutlich höher, als die Zeit, die für die statistische Auswertung bei gleichem Ergebnis benötigt wird.

Spalte Fünf und Sechs zeigen, daß auch bei Limitierung der Trainingszeit auf die Inferenzdauer t_{inf} nur für kleine Netze unter 1000 Stellen akzeptable Ergebnisse erreicht werden. Für größere Netze wächst durch die Zeitbeschränkung der Fehler dagegen stark an und überschreitet den bei dem statistischen Ansatz verbleibenden Fehler deutlich.

5.3 Fazit

Die bei der Evaluation gewonnenen Ergebnisse zeigen, daß die Entwurfskriterien aus Kapitel 4 erfolgreich umgesetzt worden sind. Der statistische Ansatz skaliert auch für sehr große Systeme mit resultierenden Fuzzy Petrinetzen im Umfang von über 10.000 Stellen, was einem Quelltextumfang von mehreren 100.000 Zeilen entspricht. Der durch die Lernmaschine erzeugte Mehraufwand während der Musteranalyse liegt mit dem circa dreifachen der Auswertungszeit der FPNs beziehungsweise bis zu einem Viertel der Gesamtinferenzdauer in einem vertretbaren Bereich und die für das Training mit den gesammelten Lernmustern benötigte Zeit ist vernachlässigbar klein. Damit ist der Ansatz laufzeiteffizient genug, um etwa auch ein interaktives Online-Training zu ermöglichen, bei dem für jedes neue Lernmuster das PDN angepaßt und dem Benutzer die resultierende Veränderung der Fuzzy-Werten präsentiert werden kann. Die durch die statistische Adaption erreichte Fehlerreduktion ist wie gesehen ebenfalls hinreichend gut, um die auf Basis der korrigierten Vertrauens- und Schwellwerte berechneten Fuzzy-Werte als zuverlässige Richtgrößen für die Musterauswertung durch den Reengineer verwenden zu können. Lediglich bei kleinen Netzen kann der Ansatz nicht seinen vollen Wirkungsgrad erreichen. Hier ist gegebenenfalls auf andere Methoden auszuweichen, falls hauptsächlich kleine Systeme oder Systemteile analysiert werden.

Im Gegensatz zur statistischen Adaption liefert der konventionelle Regressionsansatz mit Fuzzy Neural Nets und Backpropagation zwar eine durchgehend bessere Fehlerreduktion. Jedoch ist die hierfür benötigte Berechnungszeit so hoch, daß dieser Ansatz für die praktische Verwendung in dem hier betrachteten Einsatzfeld nicht geeignet ist. Bereits die Bearbeitung eines neuen Lernmusters überschreitet für große Systeme die Inferenzdauer deutlich und auch die Trainingsdauer ist, mit bis zu 18 Stunden für Netze über 10.000 Stellen, unvertretbar hoch. Lediglich bei kleinen Netzen bis zu 1000 Stellen kann dieser Ansatz sinnvoll eingesetzt werden. Da gerade in diesem Bereich die Schwächen der statistischen Adaption liegen, kann gegebenenfalls eine Kombinations-Strategie der beiden Ansätze in Betracht gezogen werden.

KAPITEL 6

Zusammenfassung und Ausblick

6.1 Zusammenfassung

In dieser Arbeit ist ein statistisches Adaptionsverfahren vorgestellt worden, dessen Ziel die Optimierung der unscharfen Gütebewertung von Entwurfsmustern ist, die zuvor durch ein Mustererkennungssystem gefunden worden sind. Die durch die Gütebewertung berechneten Fuzzy-Werte helfen dabei, sicher erkannte Muster von solchen zu unterscheiden, die einer weiteren Überprüfung durch den Reengineer bedürfen, und sind somit ein wichtiger Faktor für eine effektive und zuverlässige Mustererkennung. Im Vergleich zu Abbildung 1.1 aus der Einführung ist mit dem entwickelten Adaptionsverfahren somit die in Abbildung 6.1 grün hervorgehobene Rückkopplung zwischen Benutzerkorrekturen und den Vertrauens- und Schwellwerten der Musterspezifikation realisiert worden.

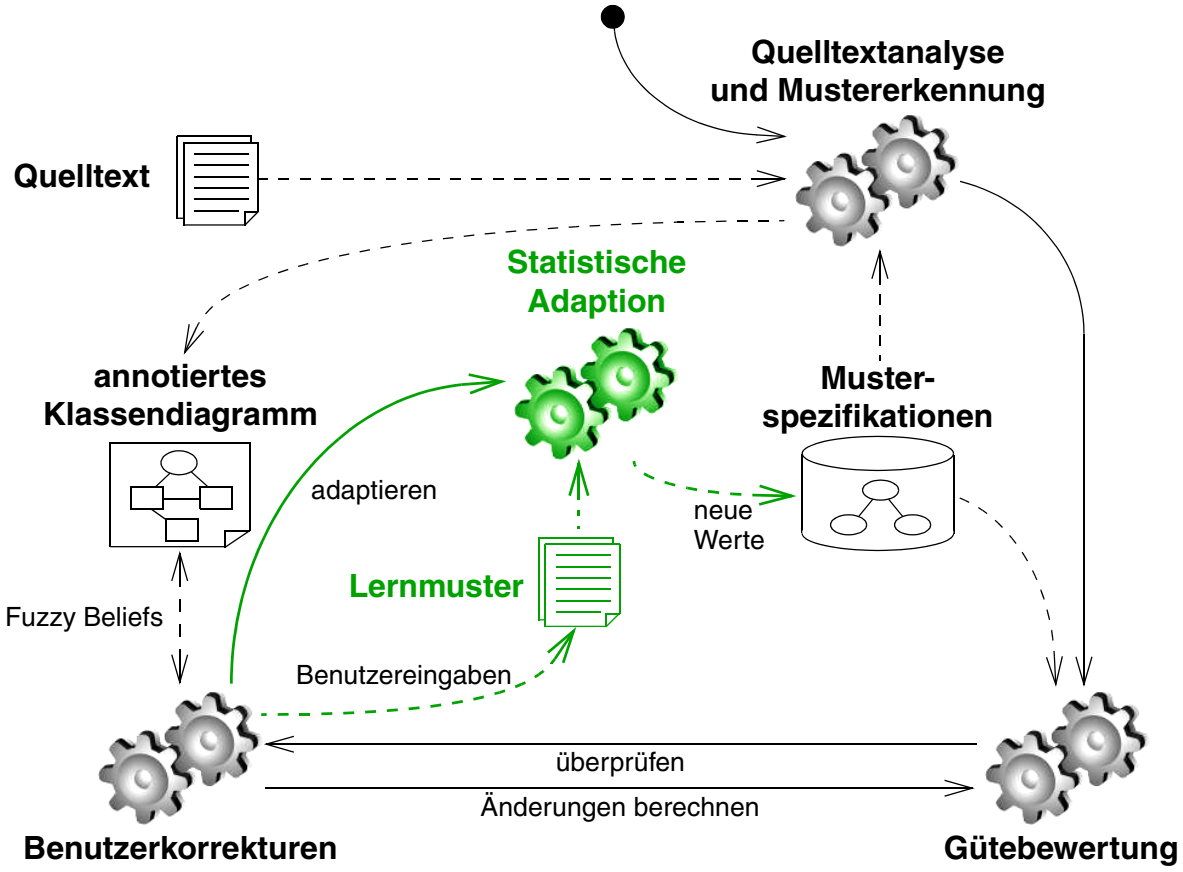


Abbildung 6.1: Das Pattern-Klassendiagramm

Die Mustererkennung erfolgt durch einen halbautomatischen Inferenzprozess, der in der Diplomarbeit von Lothar Wendehals[Wen01] vorgestellt worden ist. Die Gütebewertung der hierbei erkannten Muster geschieht durch ein Fuzzy Petrinetz auf der Basis zweier Bewertungsparameter, der Vertrauens- und Schwellwerte, die bei der Spezifikation der Erkennungsregeln festgelegt worden sind. Der Benutzer kann auf diese Bewertung Einfluß nehmen, indem er die berechneten Fuzzy-Werte auf Basis seiner eigenen Analysen korrigiert. Der in dieser Arbeit entwickelte Ansatz realisiert die automatische Anpassung der Vertrauens- und Schwellwerte anhand dieser Benutzerkorrekturen, um so zukünftige Bewertungsergebnisse zu optimieren.

Das Problem, das bei der Optimierung der Vertrauens- und Schwellwerte entsteht, besteht darin, daß es für jede vom Benutzer durchgeführte Korrektur eine Vielzahl an Möglichkeiten gibt, diese durch Anpassung der beiden Bewertungsparameter umzusetzen. Zudem können einige dieser Möglichkeiten zu weiteren Benutzerkorrekturen im Widerspruch stehen. Der hier vorgestellte Ansatz löst dieses Problem, indem die vom Benutzer vorgegebenen Fuzzy-Werte für jeden Vertrauens- und Schwellwert gesammelt werden, durch dessen Anpassung diese Werte erricht werden können. Die Entscheidung zur Anpassung der Bewertungsparameter wird dadurch von einem globalen Problem mit allen Vertrauens- und Schwellwerten des Musterkataloges auf eine lokale Entscheidung für jeden einzelnen Wert reduziert. Die verbleibende lokale Bestimmung des optimalen Wertes erfolgt durch statistische Auswertung der gesammelten Daten.

Es ist gezeigt worden, wie die Benutzerkorrekturen mit Hilfe gewichteter Änderungsanforderungen durch Propagation in dem Fuzzy Petrinetz der aktuellen Mustererkennung zu den anzupassenden Stellen für die Vertrauens- und Schwellwerte transportiert werden können. Die hierbei verwendete Gewichtung bevorzugt dabei die wahrscheinlichsten Anpassungsvarianten auf Basis der im Petrinetz berechneten Werte und verbessert damit den Lernerfolg. Eine weitere Verbesserung des Lernerfolges ist durch die Gewichtserhöhung von zueinander konsistenten Änderungsanforderungen durch Endorsement-Gewichte erreicht worden.

Zur Berechnung der optimierten Bewertungsparameter ist ein einfaches statistisches Verfahren vorgestellt worden, bei dem die gesammelten Änderungsdaten mit Hilfe statistischer Standardverfahren zunächst bereinigt werden können, bevor durch Minimierung des lokalen Fehlers die neuen Vertrauens- und Schwellwerte bestimmt werden. Im Falle der Vertrauenswerte erfolgt die Fehlerminimierung durch gewichtetes Mitteln der gesammelten Daten, im Falle der Schwellwerte durch lineare Suche des optimalen Wertes.

Besonderes Augenmerk hat bei der Entwicklung dieses statistischen Adaptionsverfahrens auf der Laufzeit- und Speichereffizienz gelegen, da die Mustererkennung insbesondere für das Reverse-Engineering sehr großer Softwaresysteme von Bedeutung ist. Daß die statistische Adaption auch für Fuzzy Petrinetze mit mehreren zehn- bis hunderttausend Stellen noch skaliert und hinreichend genaue Ergebnisse liefert, ist in der Evaluation gezeigt worden. Es hat sich auch erwartungsgemäß gezeigt, daß das Verfahren insbesondere für solche hohen Netzgrößen gute Ergebnisse liefert, während es für Netze mit bis zu wenigen hundert Stellen nicht gut geeignet scheint.

Für den zum Vergleich herangezogenen klassischen Regressionsansatz mit neuronalen Netzen und Backpropagation hat sich im Gegensatz dazu gezeigt, daß zwar eine bessere Fehlerreduktion erreicht wird, das Verfahren jedoch aufgrund der mit der Größe der analysierten Systeme stark ansteigenden Lerndauer für den praktischen Einsatz mit großen Netzen ungeeignet ist.

6.2 Ausblick

6.2.1 Statistische Adaption

Ein wichtiger Faktor für den Lernerfolg des statistischen Adaptionsansatzes ist die Gewichtung der Änderungsanforderungen. In Kapitel 4 ist hierfür eine Gewichtsfunktion auf der Basis der logistischen Kurve vorgestellt worden. Durch eine bessere Parameterwahl der Gewichtsfunktion oder alternative Funktionsverläufe kann an dieser Stelle der Lernerfolg unter Umständen noch weiter verbessert werden. Hierzu bieten sich genauere Versuche zur Auswirkung unterschiedlicher Gewichtsfunktionen auf die Lernergebnisse an.

Bei der Evaluierung ist gezeigt worden, daß der statistische Ansatz besonders geeignet ist für sehr große Netze, während er für kleine Netze zum Teil schlechte Ergebnisse liefert. Im Gegensatz dazu liefert der Neuronale Netze Ansatz im Bereich der kleinen Netze, wo dessen Laufzeit noch hinreichend niedrig ist, bessere Ergebnisse. Hier kann zukünftig eventuell eine Kombinationsstrategie anvisiert werden, bei der das Training kleiner Netze durch FNNs erfolgt, während große Netze mit dem hier vorgestellten Ansatz trainiert werden. Hierzu ist jedoch zunächst eine geeignete Zusammenführung der Lernergebnisse der beiden Ansätze zu entwickeln.

Der statistische Ansatz ist bisher ausschließlich im Rahmen der hier betrachteten Mustererkennung unetrsucht worden. Es wäre wünschenswert, weitere Erfahrungswerte über den Einsatz dieses Verfahrens mit anderen Rechennetzen, als den hier verwendeten Fuzzy Petrinetzen, zu sammeln. Von besonderem Interesse hierbei ist, ob eine Ausweitung des Ansatzes auf Netze möglich ist, die Zyklen aufweisen. Die Terminierung der Propagierungsalgorithmus kann in solchen Fällen durch das Setzen einer unteren Schranke für die Änderungsgewichte und eine Gewichtung der Zyklen, durch die eine Gewichtsreduktion bei jedem Zyklendurchlauf sichergestellt ist, erreicht werden. Ob hierbei sinnvolle Lernergebnisse zu erreichen sind, bleibt zu untersuchen.

Das vorgestellte Adaptionsverfahren kann in zwei Teile untergliedert werden, die Aufbereitung der Korrekturdaten im FPN und die statistische Ermittlung der neuen Werte auf Basis der gesammelten Lerndaten. Für den ersten Teil ist hier eine einfache, gewichtete Propagierungsheuristik verwendet worden. In Umgebungen mit weniger starken Anforderungen hinsichtlich der Skalierbarkeit kann dieses Verfahren unter Umständen durch ein optimaleres, gegebenenfalls auch iteratives Verfahren ersetzt werden, das für ein gegebenes Lernmuster die optimalen Parameter bestimmt. Die Bestimmung des optimalen Wertes über alle Lernmuster kann dann wiederum durch statistische Auswertung der Einzelwerte der Lernmuster erfolgen. So kann

unter Umständen der Lernerfolg durch eine etwas laufzeitintensivere Aufbereitung in Kombination mit der einfachen und effizienten Zusammenführung der aufbereiteten Daten gesteigert werden. Durch einen solchen Kombinationsansatz kann unter Umständen die Laufzeit des Lernprozesses gegenüber einer rein iterativen Lösung auf Kosten der Fehlerreduktion stark reduziert werden, da im Allgemeinen die iterative Optimierung jedes einzelnen Lernmusters deutlich einfacher ist, als die Iteration über alle Lernmuster.

6.2.2 Optimierungsmöglichkeiten im Inferenzprozess

Neben der hier betrachteten Optimierung der Parameter der Fuzzy-Bewertung existieren noch mindestens zwei weitere Bereiche im vorgestellten Inferenzprozess, in denen eine computergestützte Optimierung möglich erscheint.

Zum einen ist hier die Anwendungsreihenfolge der Musterregeln während der Inferenz zu nennen. Diese Reihenfolge ist durch zwei Parameter bestimmt. In der Bottom-Up-Phase wird die Auswertung einer Musterregel durch das Finden einer Prämisse der Regel ausgelöst, die für diese Regel als Trigger spezifiziert ist. Durch Auswertung der statistischen Häufigkeit des Auftretens der Prämissen einer Regel kann in einem automatischen Ansatz mit unsupervised learning die Selektion der Trigger-Prämisse optimiert werden. In der Top-Down-Phase wird die Ausführungsreihenfolge durch das Level der getriggerten Regeln bestimmt. Durch Einführen eines zusätzlichen, den Kanten des PDN zugeordnetes Level-Attribut kann die Ausführungsreihenfolge in Abhängigkeit von dem Nachbereichsprädikat kontrolliert werden, das die Top-Down-Analyse veranlaßt hat. Zur Optimierung dieser Reihenfolge kann die Zahl der erfolgreichen und erfolglosen Top-Down-Läufe statistisch ausgewertet werden, um die Prädikate mit der niedrigsten Erfolgsquote zuerst auszuwerten. Hierdurch wird ein frühestmögliches Scheitern nicht erfüllbarer Musterregeln in der Top-Down-Phase gewährleistet.

Ein weiterer Bereich, in dem Lernen in den Inferenzprozess eingehen kann, sind die Musterspezifikationen. Es ist zu untersuchen, inwieweit es möglich ist, mit Hilfe von Lernansätzen Musterspezifikationen zu verbessern, um vom Benutzer manuell angelegte Musterannotationen zukünftig zu berücksichtigen. Hierzu könnten etwa strukturelle Ähnlichkeiten zwischen den Elementen des abstrakten Syntaxgraphen im Bereich der Annotationen untersucht werden.

ANHANG A

Literaturverzeichnis

[AGG] Technical University of Berlin. *AGG, the Attributed Graph Grammar system. Online at http://www.tfs.cs.tu-berlin/agg*.

[And97] U. Anders. *Statistische neuronale Netze*. Verlag Franz Vahlen GmbH, 1997.

[CM98] V.S. Cherkassky and F. Mulier. *Learning from Data: Concepts, Theory, and Methods*. John Wiley and Sons, Inc., 1998.

[FNT98] T. Fischer, J. Niere, and L. Torunski. *Konzeption und Realisierung einer integrierten Entwicklungsumgebung für UML, Java und Story-Driven-Modeling. Master's thesis,* University of Paderborn, Department of Mathematics and Computer Science, Paderborn, Germany, July 1998.

[Gal93] S.I. Gallant. *Neural Network Learning and Expert Systems*. MIT Press, Massachusetts Institute of Technology, 1993.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object Oriented Software*. Addison-Wesley, Reading, MA, 1995.

[Jah99] J.H. Jahnke. *Management of Uncertainty and Inconsistency in Database Reengineering Processes*. PhD thesis, University of Paderborn, Paderborn, Germany, September 1999.

[NSW+02] J. Niere, W. Schäfer, J.P. Wadsack, L. Wendehals, and J. Welsh. Towards pattern-based design recovery. In *Proc. of the 24th International Conference on Software Engineering (ICSE), Orlando, Florida, USA*, pages 338–348, May 2002.

[Pal01] M. Palasdies. *Design-Pattern Spezifikation und Erkennung auf Basis von Storydiagrammen.* Master's thesis, University of Paderborn, Department of Mathematics and Computer Science, Paderborn, Germany, May 2001.

[Sac04] L. Sachs. *Angewandte Statistik - Anwendung statistischer Methoden*. Springer Verlag, Berlin, 2004.

[Sar99] Warren S. Sarle. *Ill-conditioning in Neural Network Learning.* Online unter ftp://ftp.sas.com/pub/neural/illcond/illcond.html, SAS Institute Inc., September 1999.

[Str99] C. Strebin. *Adaption unsicheren Reverse-Engineering-Wissens auf Basis konnektionistischer Methoden.* Master's thesis, University of Paderborn, Department of Mathematics and Computer Science, Paderborn, Germany, 1999.

[vdSH98] Patrick van der Smagt and Gerd Hirzinger. *Solving the Ill-Conditioning in Neural Network Learning.* In *Neural Networks: Tricks of the Trade, G. Orr and K.-R. Mller (eds), Lecture Notes in Computer Science,* 1524:193–206, 1998.

[UML] Object Management Group. *UML documentation version 1.4 (2002). Online at http://www.uml.org*.

[Wen01] L. Wendehals. *Cliché- und Mustererkennung auf Basis von Generic Fuzzy Reasoning Nets.* Master's thesis, University of Paderborn, Department of Mathematics and Computer Science, Paderborn, Germany, October 2001.

[Wer94] P.J. Werbos. *The Roots Of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting.* John Wiley and Sons, Inc., 1994.

[Zün96] A. Zündorf. *Graph Pattern Matching in Progres.* In *Proc. of the 5th International Workshop on Graph-Grammars and their Application to Computer Science*, LNCS 1073. Springer Verlag, 1996.

[Zün01] A. Zündorf. *Rigorous Object Oriented Software Development*. University of Paderborn, 2001.

ANHANG B

Abbildungsverzeichnis

Abbildung 1.1:Überblick über den Mustererkennungsprozess 3
Abbildung 2.1:ASG-Repräsentation von Quellcode 6
Abbildung 2.2:Das Composite-Pattern 7
Abbildung 2.3:Eine Variante des Composite-Patterns 8
Abbildung 2.4:Implementierung von zu-n-Assoziationen 9
Abbildung 2.5:Formale Spezifikation des Composite-Patterns 9
Abbildung 2.6:Das Pattern-Klassendiagramm 11
Abbildung 2.7:Ausschnitt eines Pattern Dependency Nets 14
Abbildung 2.8:Beispielausführung des Erkennungsprozesses 19
Abbildung 2.9:Ausschnitt eines Fuzzy Petrinetzes 22
Abbildung 2.10:Fuzzy Petrinetz nach der Auswertung 25
Abbildung 2.11:Inferenzprozess Statechart 25
Abbildung 3.1:Lernen eines Regressionsproblems 31
Abbildung 3.2:Die Lernmaschine im Detail 32
Abbildung 3.3:Ein Neuron 36
Abbildung 3.4:Implikation im FNN 37
Abbildung 3.5:Umsetzung von Vererbung im FNN 38
Abbildung 3.6:Fuzzy Petrinetz nach der Auswertung 41
Abbildung 3.7:Backpropagation als Pseudo-Code 42
Abbildung 4.1:Realisierungsmöglichkeiten 47
Abbildung 4.2:Vertrauenswerte als Bias-Stellen 49
Abbildung 4.3:Realisierungsmöglichkeiten 50
Abbildung 4.4:Gewichtete Propagation der Vertrauenswerte 52
Abbildung 4.5:Propagierung der Änderungsanforderungen 53
Abbildung 4.6:Propagierung mit Endorsements 55
Abbildung 4.7:Realisierungsmöglichkeiten 56
Abbildung 4.8:Bestätigen von Fuzzy-Werten 58
Abbildung 4.9:Propagierung mit zusätzlichen Stellen 65
Abbildung 4.10:Benutzereingaben für Schwellwerte 67
Abbildung 5.1:Generieren von Lernmustern 71
Abbildung 6.1:Das Pattern-Klassendiagramm 79