Effizientes Problemlösen durch flexible Wiederverwendung von Fällen auf verschiedenen Abstraktionsebenen

Ralph Bergmann

vom Fachbereich Informatik der Universität Kaiserslautern
zur Verleihung des akademischen Grades
Doktor der Naturwissenschaften (Dr. rer. nat.)
genehmigte Dissertation

Dekan:	Prof. Dr. Hans Hagen
Berichterstatter:	Prof. Dr. Michael M. Richter
	Prof. Dr. Fritz Wysotzki

Tag der wissenschaftlichen Aussprache: 28. Juni 1996

D 386

Geleitwort des Doktorvaters

Die Suche nach Aktionsplänen, die einen vorgegebenen Anfangs- in einen geforderten Zielzustand überführen, ist bekanntermaßen komplex und Gegenstand vielfältiger Untersuchungen. Gefordert ist einerseits, daß die Pläne korrekt sind, d. h. nur zulässige Teilschritte enthalten und zum anderen, daß sie möglichst geringe Kosten und einen möglichst hohen Nutzen haben. Zu den drei wichtigsten Methoden der Suchraumeinschränkung gehören die Einbringung und Verwendung von Hintergrundwissen (vor allem für die Kontrollstrukturen), die Verwendung von früher bereits gelösten und möglichst ähnlichen Planungsfällen sowie der Übergang zu abstrakteren Ebenen. Jede dieser drei Vorgehensweisen ist geeignet, die Komplexität der Suche für spezielle Anwendungen zu reduzieren. Die bisherige Erfahrung zeigt jedoch, daß die einzelne Anwendung derartiger Vorgehensweisen nur selten zu den gewünschten Ergebnissen führt und ihre Kombination sehr wünschenswert wäre. Die vorliegende Arbeit hat sich die Aufgabe gestellt, die drei genannten Vorgehensweisen zu integrieren. Dazu mußte das vorhandene Methodenspektrum zunächst aufgearbeitet und erweitert werden und sodann mußten die nötigen Grundlagen zur Integration geschaffen werden. Darauf aufbauend hat der Verfasser dann in PARIS (Plan Abstraction and Refinement in an Integrated System) ein konkretes integriertes System konzipiert und implementiert. Dieses System ist praktisch orientiert und muß neben der effizienten Plangenerierung auch Aspekte wie Domänenunabhängigkeit und inkrementelle Erweiterbarkeit berücksichtigen.

In der Aktionsplanung ist es stets zweckmäßig, als logische Basis eine Form der Modallogik zu wählen und so wird auch hier vorgegangen. Bei der Verwendung von Fällen erfordern Planungsaufgaben stets die Adaption der Lösungen (also der früheren Pläne). Im Prinzip erfolgt die Vorgehensweise nach dem bekannten Schema von Aamodt und Plaza. Im Kontext der vorliegenden Arbeit muß jedoch eine wichtige Besonderheit beachtet werden. Üblicherweise werden zwar nicht identische sondern nur ähnliche Fälle betrachtet, sie liegen jedoch stets auf der gleichen Abstraktionsebene. Hier muß jedoch berücksichtigt werden, daß sich Fälle auf unterschiedlichen Abstraktionsebenen bewegen; deshalb spielen Methoden zur Fallabstraktion und -verfeinerung eine wesentliche Rolle. Die unterschiedlichen Abstraktionsebenen hängen mit dem vorhandenen Hintergrundwissen in der Weise zusammen, daß manches Wissen erst auf bestimmten Beschreibungsebenen vorhanden ist und deswegen geeignete Abstraktions- und Konkretisierungsabbildungen nötig werden. Es ist auch bekannt, daß ein Wechsel der Beschreibungsebene zu einem Wechsel der Komplexitätsklassen führen kann. Das Einbringen von Hintergrundswissen selbst und der Zusammenhang zwischen Generalisierung und Spezialisierung von Fällen wird im Prinzip mit der Methode des erklärungsbasierten Lernens angegangen.

Ich sehe den Wert der vorliegenden Arbeit vor allem darin, daß ein umfassender neuer Ansatz für die Aktionsplanung vorgestellt wird. Um eine Verbindung zwischen Grundlagenforschung und Anwendungsorientierung herzustellen, ist dieser Ansatz auf drei Ebenen behandelt worden, auf der formal-logischen Ebene, der Ebene der Algorithmen und Methoden sowie schließlich der Ebene der Datenstrukturen und der Implementierung. Das PARIS-System ist keine lose hybride Sammlung unterschiedlicher Vorgehensweisen sondern eine echte Integration, die auf einer detaillierten methodischen Analyse basiert. Ich denke, daß gerade dieser letzte Gesichtspunkt Anlaß zu weiteren Diskussionen und Arbeiten geben wird.

Prof. Dr. Michael M. Richter

Danksagungen

Mein erster Dank gilt Herrn Prof. Dr. Michael M. Richter für die Betreuung dieser Arbeit sowie Herrn Prof. Dr. Fritz Wysotzki für die Bereitschaft zur Übernahme des Korreferats. Prof. Richter hat mir durch die Aufnahme in seine Arbeitsgruppe die Anfertigung dieser Arbeit überhaupt erst ermöglicht und mich in vielen Diskussionen auf wichtige Zusammenhänge aufmerksam gemacht und mein Verständis von fallbasiertem Schließen und maschinellen Lernen deutlich mitgeprägt. Prof. Wysotzki hat durch seine großes Interesse und seine Diskussionsbereitschaft zur Präzisierung wichtiger Sachverhalte beigetragen.

Dank gebührt allen meinen Kollegen, die immer offen für Diskussionen waren und für Fragen zur Verfügung standen. Insbesondere möchte ich danken: Klaus-Dieter Althoff, Stefan Weß und Wolfgang Wilke für viele anregende Diskussionen zum fallbasierten Schließen; Christoph Globig für Diskussionen über theoretische Fragen und ganz praktische Dinge; Frank Maurer für Diskussionen zum Thema Knowledge Engineering; Héctor Muñoz für einen regen Gedankenaustausch über viele Fragen des fallbasierten Planens; Jürgen Paulokat und Frank Weberskirch für ihre Fachkompetentz im Bereich der Planung.

Dank gebührt auch denjenigen, mit denen ich während der letzten Jahre über viele verschiedene Aspekte meiner Arbeit diskutieren konnte und deren Anmerkungen immer eine Bereicherung für mich waren. Hier sind zu nennen: Aagnar Aamodt, David Aha, Katy Börner, Stefan Boschert, Karl Branting, Jaime Carbonell, Padraig Cunningham, Fausto Giunchiglia, Robert Holte, Alexander Horz, Subbarau Kambhampati, Gabriele Schmidt, Barry Smyth und Manuela Veloso.

Des weiteren möchte ich mich bei all denjenigen Studenten bedanken, die im Rahmen von Projekt- oder Diplomarbeiten sowie durch Tätigkeiten als wissenschaftliche Hilfskräfte diese Arbeit ganz wesentlich unterstützt haben: Wolfgang Wilke hat die Lernkomponenten des in dieser Arbeit beschriebenen PARIS-Systems implementiert und darüber hinaus in vielen Diskussionen zur Detaillierung meiner Ideen wesentlich beigetragen; Andreas Dannenmann, Wolfgang Haupenthal, Gerd Pews, Thomas Reinartz, und Dagmar Surmann haben durch ihre Arbeiten ebenfalls wichtige Beiträge geleistet.

Wichtig für das Zustandekommen dieser Arbeit war auch die außerordentlich gute und kooperative Arbeitsatmosphäre innerhalb unserer Arbeitsgruppe. Hierfür danke ich Prof. Michael M. Richter, Klaus-Dieter Althoff, Martin Bräuer, Cesar Carranza, Barbara Dellen, Stefan Dittrich, Christoph Globig, Edith Hüttel, Willi Klein, Gunnar Krug, Wolfgang Lenski, Frank Maurer, Héctor Muñoz, Jürgen Paulokat, Gerd Pews, Reinhard Praeger, Jürgen Rahmel, Werner Schirp, Ilka Steuerwald, Aldo von Wangenheim, Oliver Wendel, Stefan Weß, Elisabeth Wette-Roch und Wolfgang Wilke.

Für das Korrekturlesen von Rohfassungen dieser Arbeit möchte ich mich auch ganz besonders bedanken bei Christoph Globig, Edith Hüttel, Manuela Mohr, Héctor Muñoz, Frank Weberskirch und Wolfgang Wilke.

Auch meinen Eltern danke ich an dieser Stelle dafür, daß sie mir die Ausbildung ermöglicht und mich stets nach allen Kräften unterstützt haben. Besonderer Dank gebührt auch Manuela Mohr, für ihr Verständnis, ihre Hilfe und die notwendigen Ablenkungen von wissenschaftlichen und sonstigen Problemen.

Kaiserslautern, im Januar 1996

Ralph Bergmann

Zusammenfassung

Generische wissensbasierte Systeme zur Lösung konstruktiver (synthetischer) Aufgaben erfordern das Durchsuchen großer, meist exponentiell großer Lösungsräume und haben somit einen sehr großen Rechenzeitbedarf. Dieses Problem tritt typischerweise bei Aktionsplanungsaufgaben auf, die in dieser Arbeit eingehend betrachtet werden. Hierbei geht es darum, eine Folge von Aktionen, genannt Plan, zu finden, deren Ausführung, ausgehend von einer Ausgangssituation, ein bestimmtes Ziel erreicht.

Der Gegenstand dieser Arbeit ist, Methoden zu entwickeln und zu analysieren, die in der Lage sind, durch die *Wiederverwendung von Fällen* – das sind Beschreibungen bekannter Problemlösungen – neue Probleme effizient zu lösen. Hierzu werden die bekannten Fälle in einer *Fallbasis* gespeichert. Ein neues Problem wird dadurch gelöst, daß ein geeigneter Fall aus der Fallbasis ausgewählt und die im Fall enthaltene Lösung im Kontext des neuen Problems wiederverwendet wird. Eine wesentliche Schwierigkeit, die sich insbesondere in synthetischen Bereichen ergibt, ist, daß das Spektrum der vom System zu lösenden Probleme sehr groß ist und daß jedes Problem in der Regel eine individuelle Lösung erfordert. Daher müssen Fälle auf vielfältige Weise angepaßt werden. Die entwickelten Methoden setzen Abstraktions- und Generalisierungstechniken ein, um Fälle auf möglichst flexible Weise für neue Probleme wiederverwendbar zu machen. Durch die Abstraktion und nachfolgende Konkretisierung von Fällen läßt sich das in den Fällen enthaltene Wissen auf verschiedenen Abstraktionsebenen wiederverwenden, wobei unterschiedliche Beschreibungssprachen für verschiedene Abstraktionen eingesetzt werden können. Die Generalisierung von Fällen erlaubt es, einen vorliegenden Fall zu einer ganzen Klasse von Problemen und Lösungen zu erweitern und durch das Einbeziehen von Hintergrundwissen über den Anwendungsbereich die Korrektheit der entstehenden Lösungen zu garantieren.

Um die Methoden zu testen und detailliert bewerten zu können, wurden sie in einem System, PARIS (**P**lan **A**bstraction and **R**efinement in an **I**ntegrated **S**ystem) genannt, vollständig implementiert. Die realisierten Wiederverwendungsmethoden sind domänenunabhängig, setzen jedoch eine hierarchische Modellierung des betrachteten Gegenstandsbereichs voraus. Vorliegende Fälle können vollkommen automatisch zur Lösung neuer Planungsprobleme wiederverwendet werden. Durch die exemplarische Modellierung eines realen Anwendungsbereiches (Arbeitsplanung für die Fertigung von Drehteilen im Maschinenbau) wurde die Tragfähigkeit der entwickelten Methoden demonstriert. In einer Reihe von empirischen Untersuchungen, bei denen verschiedene Bedingungen systematisch variiert wurden, ließen sich wichtige Eigenschaften der Wiederverwendungsmethoden belegen. So konnte im Mittel eine signifikant höhere Leistungsfähigkeit (erforderliche Rechenzeit zur Problemlösung) im Vergleich zu generischem Problemlösen nachgewiesen werden. Außerdem zeigte sich eine hohe Flexibilität der Wiederverwendung, d. h. ein vorliegender Fall konnte für eine Vielzahl sehr unterschiedlicher Probleme effizient wiederverwendet werden. Die Arbeit diskutiert detailliert die Vor- und Nachteile der beschriebenen Methoden und liefert Kriterien für ihre erfolgreiche Anwendung.

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Gegenstand der Arbeit 1

1.1.1 Zielsetzung 2

1.1.2 Anforderungen 3

1.1.3 Methodik 5

1.1.4 Voraussetzungen 6

1.1.5 Bewertungskriterien 7

1.2 Vorbemerkungen zur Arbeit 8

1.2.1 Voraussetzungen an den Leser 8

1.2.2 Darstellungsebenen 8

1.3 Gliederung der Arbeit 9

I Grundlagen und Basismethoden **11**

2 Grundlagen aus dem Bereich des Problemlösens **13**

2.1 Problemlösen 14

2.1.1 Veränderliche Welten 14

2.1.2 Eine Beispieldomäne: Arbeitsplanung für rotationssymmetrische Drehteile 15

2.2 Formale Repräsentation (Syntax) 16

2.2.1 Relevante Grundbegriffe der Prädikatenlogik 17

2.2.2 Repräsentation von Welten 18

2.2.3 Repräsentation von Aktionen 22

2.2.4 Problemlösedomäne 24

2.3 Formale Semantik 25
2.4 Beschränkungen dieses Problemlösebegriffes 28
2.5 Der Problemlöseprozeß 30
2.5.1 Suche im Zustandsraum 31
2.5.2 Suche im Lösungsraums 32
2.5.3 Suche im Zustandsraum vs. Suche im Lösungsraum 32
2.5.4 Komplexität des Problemlösens 33

3 Leistungssteigerung von Problemlösern 35
3.1 Fallbasiertes Schließen 35
3.1.1 Die Grundidee des fallbasierten Schließens 36
3.1.2 Fallbasis und verwendete Repräsentationsformen 38
3.1.3 Retrieve 38
3.1.4 Reuse 40
3.1.5 Revise 41
3.1.6 Retain 41
3.1.7 Fallbasiertes Schließen in der Aktionsplanung 42
3.1.8 Einige Aspekte und Probleme bei Derivational Analogy 43
3.2 Erklärungsbasiertes Lernen 44
3.2.1 Die Grundidee des erklärungsbasierten Lernens 44
3.2.2 Die Lernphase 46
3.2.3 Eine allgemeine Prozedur zum erklärungsbasierten Lernen 48
3.2.4 Die Anwendungsphase 50
3.2.5 Nützlichkeitsbetrachtungen 50
3.2.6 Einige Aspekte und Probleme des erklärungsbasierten Lernens . . . 53
3.2.7 Hintergrund: Ursprünge des erklärungsbasierten Lernens 55
3.2.8 Vergleich von erklärungsbasiertem Lernen und Derivational Analogy 56
3.3 Abstraktion 57
3.3.1 Abstraktion und Konkretisierung 57
3.3.2 Hierarchisches Problemlösen 59
3.3.3 Vorgehensweise beim hierarchischen Problemlösen 61
3.3.4 Kosten beim hierarchischen Problemlösen 63

3.3.5 Bildung von Abstraktionsebenen 65
3.3.6 Einige Probleme beim Einsatz von Abstraktion 66

II PARIS: Flexible Wiederverwendung von Fällen durch Abstraktion und Generalisierung 69

4 Überblick 71
4.1 Fälle auf verschiedenen Abstraktionsebenen 71
4.1.1 Fallbasiertes Schließen und Abstraktion 71
4.1.2 Fälle auf verschiedenen Abstraktionsebenen 72
4.1.3 Die Vorgehensweise 73
4.1.4 Ein Beispiel für die Fallabstraktion 74
4.2 Erklärungsbasiertes Generalisieren abstrakter Fälle 79
4.2.1 Generalisierte Fälle 79
4.2.2 Die Vorgehensweise 81
4.2.3 Integration von Abstraktion und Generalisierung 82
4.2.4 Ein Beispiel für die Fallgeneralisierung 83
4.3 Fallauswahl 84
4.3.1 Aufwand zur Lösungsverfeinerung 84
4.3.2 Aufwand durch die Lösungsspezialisierung 85
4.3.3 Gesamter Wiederverwendungsaufwand 85
4.3.4 Retrievalaufwand und Wiederverwendungsaufwand 86
4.3.5 Indexierung der Fallbasis 86
4.3.6 Ein Beispiel für eine Klassifikationshierarchie 88
4.4 Die PARIS-Architektur 89
4.4.1 Die Lernphase 89
4.4.2 Die Problemlösephase 90

5 Fallabstraktion und Verfeinerung 91
5.1 Modell der Fallabstraktion 92
5.1.1 Abstraktion durch Wechsel der Repräsentationssprache 92
5.1.2 Voraussetzung zum Wechsel der Repräsentationsebenen 93

5.1.3 Fallabstraktion 94
5.2 Ein Algorithmus zur Fallabstraktion 100
5.2.1 Überblick über den Algorithmus 101
5.2.2 Vorbemerkung: Beweise in der Hornlogik 103
5.2.3 Phase-I: Bestimmung der konkreten Zustandssequenz 103
5.2.4 Phase-II: Bestimmung aller ableitbaren abstrakten Sätze 104
5.2.5 Phase-III: Suche nach anwendbaren abstrakten Operatoren 104
5.2.6 Phase-IV: Suche nach gerichtetem Pfad im Graph 106
5.2.7 Korrektheit und Vollständigkeit des PABS-Algorithmus 108
5.2.8 Komplexität des PABS-Algorithmus 108
5.3 Bestimmung der Wiederverwendbarkeit abstrakter Fälle 108
5.4 Verfahren zur Lösungsverfeinerung 110
5.4.1 Grundlegende Vorgehensweise 111
5.4.2 Suchverfahren zur Lösung der Teilprobleme 113
5.4.3 Backtracking zwischen der Lösung von Teilproblemen 114
5.4.4 Lösungsverfeinerung mit Backtracking und Tiefensuche mit schrittweiser Tiefenerhöhung 115
5.4.5 Aufwand zur Lösungsverfeinerung 115
5.4.6 Bemerkungen über die Länge von Plänen 117

6 Fallgeneralisierung und Spezialisierung 119
6.1 Spezifikation der Fallgeneralisierung 120
6.2 Erklärungsbasiertes Lernen zur Fallgeneralisierung 123
6.3 Phase-I: Generierung einer Fallerklärung 124
6.3.1 Der Begriff der Erklärung 124
6.3.2 Die Fallerklärung 126
6.3.3 Verschiedene Knoten und Wahrheitsfunktionen 127
6.3.4 Aufbau der Operatorerklärung 128
6.3.5 Aufbau der Vorbedingungserklärung 132
6.3.6 Algorithmus zur Erzeugung der Fallerklärung 134
6.4 Phase-II: Generalisierung der Fallerklärung 136
6.4.1 Generalisierte Erklärung 136

6.4.2 Grundalgorithmus zur Generalisierung von Erklärungen 138
6.4.3 Anwendung des Grundalgorithmus zur Generalisierung der Fallerklärung 141
6.5 Phase-III: Bildung des generalisierten Falles 143
6.6 Eigenschaften der erklärungsbasierten Fallgeneralisierung 144
6.6.1 Korrektheit des Verfahrens 144
6.6.2 Komplexität des Verfahrens 145
6.7 Lösungsspezialisierung 145
6.7.1 Matching von Zuständen 146
6.7.2 Algorithmus zum Test der Wiederverwendbarkeit und zur Durchführung der Lösungsspezialisierung 147
6.7.3 Aufwand zur Lösungsspezialisierung 149

7 Organisation der Fallbasis **151**
7.1 Strukturierung der Fallbasis 152
7.1.1 Ausgangssituation 152
7.1.2 Wiederverwendbarkeitsrelationen für generalisierte abstrakte Fälle . 152
7.2 Lernen der Wiederverwendbarkeitshierarchie 155
7.2.1 Trainingsbeispiele zum Lernen der Wiederverwendbarkeitsrelation . 155
7.2.2 Lösungserhaltende Wiederverwendbarkeit 156
7.2.3 Der Hypothesenraum des Lernalgorithmus 157
7.2.4 Darstellung der Hypothesen in einer Wiederverwendbarkeitshierarchie 158
7.2.5 Der Lernalgorithmus 158
7.2.6 Der Algorithmus für den Insert-Schritt 161
7.2.7 Ein Beispiel 162
7.3 Ein Kostenmodell 164
7.3.1 Abhängigkeit des Wiederverwendungsaufwandes von der Flexibilität der Wiederverwendbarkeit 164
7.3.2 Kostenmodell 166
7.3.3 Retrievalstrategie 166
7.3.4 Ermittlung des Erwartungswertes der Problemlösezeit 168
7.3.5 Schätzung des Erwartungswertes 170
7.4 Ein Beispiel 171

7.5 Einsatz des Kostenmodells 174
7.5.1 Optimierung der Knotenordnung 174
7.5.2 Optimierung der Selektionsprädikate 175

III Anwendung, Bewertung und Diskussion 177

8 Anwendung und empirische Untersuchungen 179
8.1 Kriterien 180
8.1.1 Vollständigkeit der abstrakten Domänentheorie 180
8.1.2 Unabhängigkeit der abstrakten Operatoren 181
8.1.3 Teilzieldistanz abstrakter Operatoren 181
8.1.4 Generalität von Operatordefinitionen 181
8.1.5 Komplexität der generischen Abstraktionstheorie 182
8.2 Arbeitsplanung von Werkstücken 183
8.2.1 Die konkrete Problemlösedomäne 183
8.2.2 Die abstrakte Problemlösedomäne 186
8.2.3 Beurteilung der Kriterien 188
8.3 Ein Beispiel 189
8.3.1 Fallabstraktion 189
8.3.2 Fallgeneralisierung 189
8.3.3 Erweiterung der Fallbasis 192
8.3.4 Retrieval eines Falles zur Lösung eines neuen Problems 194
8.3.5 Lösungsspezialisierung 194
8.3.6 Lösungsverfeinerung 195
8.3.7 Mehrfach iteriertes Lernen und Problemlösen 196
8.4 Experimentelle Untersuchungen 197
8.4.1 Hypothesen 198
8.4.2 Durchführung der Experimente 200
8.4.3 Statistische Aufbereitung der Daten 202
8.4.4 Experiment 1: Problemlösen ohne Wiederverwendung 203
8.4.5 Experiment 2: Flexibilität der Wiederverwendung 204
8.4.6 Experiment 3: Vergleich von Kompetenz und Effizienz 206

8.4.7 Experiment 4: Detaillierte Untersuchung von Kompetenz- und Effizienzsteigerungen . . . 207
8.4.8 Experiment 5: Abstraktion ohne Wechsel der Repräsentationssprache 210
8.4.9 Experiment 6: Vergleich der Retrievalverfahren . . . 211
8.4.10 Experiment 7: Einfluß der Problemlösedomäne auf die Effizienz der Wiederverwendung . . . 213

9 Diskussion und Literaturvergleich 215
9.1 Einordnung ins fallbasierte Schließen . . . 215
9.1.1 Wissen in fallbasierten Systemen . . . 215
9.1.2 Retrieval: Auswahl ähnlicher Fälle . . . 220
9.1.3 Reuse: Anpassung der Lösung . . . 221
9.1.4 Revise: Überprüfung der Lösung . . . 223
9.1.5 Retain: Die Lernphase . . . 223
9.2 Fallbasierte Planungssysteme . . . 224
9.2.1 PRODIGY/Analogy . . . 224
9.2.2 CAPlan/CbC . . . 227
9.2.3 PRIAR . . . 228
9.3 Einordnung ins erklärungsbasierte Lernen . . . 231
9.3.1 Aufwandsanalyse . . . 231
9.3.2 Prodigy/EBL . . . 232
9.3.3 Lernen von Makrooperatoren . . . 233
9.3.4 Lernen von Plangeneralisierungen . . . 233
9.3.5 Fallbasierte Ansätze im Vergleich . . . 234
9.3.6 PARIS . . . 234
9.4 Einordnung in Abstraktionsansätze . . . 235
9.4.1 ABSTRIPS und ABTWEAK . . . 235
9.4.2 ALPINE und HIGHPOINT . . . 236
9.4.3 Graphenbasierte Verfahren . . . 237

10 Zusammenfassung, Bewertung und Ausblick 239
10.1 Zusammenfassung . . . 239
10.2 Gesamtbewertung . . . 242

10.2.1 Einsatzvoraussetzungen 242
10.2.2 Diskussion der Anforderungen 244
10.2.3 Grenzen dieser Arbeit 246
10.3 Ausblick 247

Literaturverzeichnis **249**

Abbildungsverzeichnis **269**

Tabellenverzeichnis **273**

Index **274**

Kapitel 1

Einleitung

1.1 Gegenstand der Arbeit

Generische wissensbasierte Systeme zur Lösung konstruktiver (synthetischer) Aufgaben, wie etwa Entwurfs- oder Planungsaufgaben, erfordern das Durchsuchen großer, meist exponentiell großer Lösungsräume. Dies kann zwar für stark idealisierte Bereichsmodellierungen (sogenannte „Spielzeugwelten“) oder sehr einfache Problemstellungen praktikabel sein, stellt aber für praktisch relevante Probleme in der Regel ein unüberwindbares Hindernis dar. Oft verfügt ein problemlösendes System zwar über genügend Wissen, um die Korrektheit einer Lösung zu entscheiden, jedoch verhindert die hohe Komplexität der Suche das Auffinden der Lösung innerhalb tolerierbarer Zeitgrenzen. Dieses Problem tritt selbst dann auf, wenn man die Qualität der Lösung außer acht läßt und sich darauf beschränkt, korrekte Lösungen zu bestimmen.

Generell besteht zwar sowohl die Möglichkeit, den Suchprozeß durch bereichsspezifisches Kontrollwissen zu steuern, als auch effizientere Algorithmen, speziell für den gerade vorliegenden Anwendungsbereich oder für einzelne Problemklassen, zu entwickeln. Jedoch sind beide Varianten mit einem erheblichen Entwicklungsaufwand verbunden, da entweder Kontrollwissen von einem Bereichsexperten akquiriert und formalisiert werden muß oder gar vollständig neue Algorithmen entwickelt werden müssen. Selbst das Finden von Teilproblemklassen, die effiziente Spezialalgorithmen zur Problemlösung ermöglichen, ist bereits oft sehr schwierig und aufwendig. Darüber hinaus ist dieser immense Entwicklungsaufwand für jede einzelne Anwendung erneut zu leisten. Diese Vorgehensweisen sind daher für die Entwicklung wissensbasierter Systeme nicht rentabel.

Der Gegenstand dieser Arbeit ist daher, Methoden zu entwickeln und zu analysieren, die die Lösung von synthetischen Aufgaben effizienter gestalten und einen geringeren Entwicklungsaufwand erfordern als generische Ansätze. Die Arbeit konzentriert sich hierbei auf einen speziellen Typ von synthetischen Aufgaben: die Aktionsplanung. Diese ist von anderen Entwurfsaufgaben, die gewöhnlich auch als Planung bezeichnet werden (z. B.

Stadtplanung), zu unterscheiden. Bei der Aktionsplanung geht es darum, eine Folge von Aktionen, genannt Plan, zu finden, deren Ausführung, ausgehend von einer Ausgangssituation, ein bestimmtes Ziel erreicht. Ausgangssituation und Zielsituation zusammen bilden die *Problemstellung*; die Aktionsfolge ist die *Lösung*. Viele praktische Probleme (z. B. Arbeitsplanung für die Fertigung im Maschinenbau) lassen sich als Aktionsplanungsprobleme formulieren. Daher ist dies ein wichtiges Teilgebiet der synthetischen Aufgaben. Der Gegenstand dieser Arbeit ist außerdem eingeschränkt auf das Finden korrekter Lösungen; die Lösungsqualität (Optimalität oder Suboptimalität) wird nicht thematisiert. Der Grund für diese Einschränkung liegt darin, daß generische Methoden zum Finden korrekter Lösungen bereits eine hohe Zeitkomplexität besitzen. Somit ergibt sich trotz dieser Einschränkung genügend Raum für relevante Verbesserungen. Außerdem ist die Problematik der Formulierung von Qualitätsanforderungen an Pläne noch nicht befriedigend gelöst. Weitere umfassende Vorarbeiten auf diesem Gebiet sind vonnöten, bevor man sich dieser erweiterten Fragestellung sinnvoll widmen kann.

1.1.1 Zielsetzung

In dieser Arbeit wird ein Weg vorgeschlagen, der die Wiederverwendung von *Fällen* in den Mittelpunkt des Problemlöseprozesses stellt. Ein Fall beschreibt eine frühere Problemlösesituationen durch die Angabe einer Problemstellung zusammen mit einer dazugehörenden korrekten Lösung. Fälle entstehen oft als Nebenprodukt einer vom Menschen durchgeführten Problemlösung und werden meist aus Dokumentationsgründen aufbewahrt. Sind diese Dokumentationen bereits in elektronischer Form archiviert, so können sie als Fall aufbereitet und verwendet werden. Damit sind Fälle eine leicht verfügbare Wissensform, die man versuchen kann, nutzbringend für zukünftiges Problemlösen einzusetzen. Ziel dieser Arbeit ist es daher, Methoden zu entwickeln, die Problemlösen durch Auswahl und Wiederverwendung von Erfahrungswissen aus Fällen ermöglichen. Hierbei ergeben sich eine Reihe von allgemeinen Fragestellungen, die zum Erreichen dieses Ziels geklärt werden müssen.

Insbesondere in synthetischen Bereichen ergibt sich die Schwierigkeit, daß das Spektrum der vom System zu lösenden Probleme sehr groß ist und daß jedes Problem in der Regel eine individuelle Lösung erfordert. Im Vergleich zur Größe des Problem- und Lösungsspektrums ist die Menge der verfügbaren Fälle jedoch gering. Daher ist es praktisch ausgeschlossen, daß ein Fall vorhanden ist, dessen Lösung unverändert übernommen werden kann. Aus dieser Problematik ergibt sich die zentrale Fragestellung dieser Arbeit: wie kann die in einem Fall enthaltene Lösung wiederverwendet werden, um Probleme zu lösen, die von dem im Fall behandelten Problem mehr oder weniger stark abweichen? Dies ist die Frage nach der *Flexibilität* der Wiederverwendung. Während die Wiederverwendung von Fällen für nicht identische sondern nur ähnliche Probleme die Grundidee des *fallbasierten Schließens* ansich ist, bringen wir hier eine neue Idee ein. Diese besteht darin, Fälle auf anderen Abstraktionsebenen wiederzuverwenden als die, auf der der Fall ursprünglich vorlag.

Damit eng verbunden ist die Frage nach einem *Kostenmodell* für die Berechnungskosten, die die Wiederverwendung verursacht. Hierbei sind nicht die Kosten gemeint, die die Umsetzung der gefundenen Lösung (z. B. die Herstellungskosten bei der Ausführung eines Fertigungsplans) verursachen, sondern die Kosten der Lösungsberechnung durch Wiederverwendung. Ein solches Kostenmodell kann dazu beitragen, bei einem neuen Problem zu entscheiden, ob sich die Wiederverwendung eines vorhandenen Falls aus Effizienzgründen überhaupt lohnt, oder ob ein rein generisches Vorgehen vorzuziehen ist.

Eine weitere zentrale Frage stellt sich bei der Betrachtung der Fallauswahl. Sind mehrere Fälle wiederverwendbar, so ist ein Kriterium erforderlich, daß einem Fall den Vorzug gibt. Hierbei kann das Kostenmodell eine Grundlage für die Fallauswahl darstellen. Es sollte ein Fall ausgewählt werden, dessen Wiederverwendungskosten möglichst gering sind.

Ein weiteres Problem bei der Fallauswahl ergibt sich, wenn eine große Zahl von Fällen vorliegt. Die Auswahl eines wiederzuverwendenden Falls ist dann mit einem gewissen Aufwand verbunden. Daher stellt sich die Frage, wie die Fallbasis organisiert werden muß, damit der Aufwand zur Fallauswahl gering gehalten werden kann.

Um diese Ziele hier bereits kurz gegen verwandte Fragestellungen abzugrenzen, soll noch erwähnt werden, daß es nicht das Ziel dieser Arbeit ist, z. B. durch effizientere Suchalgorithmen, einen an sich leistungsfähigeren generischen Problemlöser zu entwickeln. Ebensowenig ist es das Ziel dieser Arbeit, die Qualität von Problemlösungen mit Hilfe von Fällen zu steigern. Auch wenn diese beiden Punkte für den praktischen Einsatz eines wissensbasierten Systems von wichtiger Bedeutung sind, können sie im Rahmen dieser Arbeit nicht behandelt werden.

1.1.2 Anforderungen

Bei der Verfolgung der oben genannten Ziele und Fragestellungen müssen eine Reihe von allgemeinen Anforderungen berücksichtigt werden, damit sich der Einsatz einer fallverarbeitenden Methodik im Vergleich zu herkömmlichen generischen Methoden lohnt. Diese Anforderungen wollen wir nachfolgend kurz charakterisieren.

Höhere Leistungsfähigkeit: Ein Problemlöser, der neue Probleme durch Wiederverwendung von Fällen löst, muß eine deutlich höhere Leistungsfähigkeit besitzen als ein generischer Problemlöser, da ansonsten der Einsatz von Fällen keinen Vorteil bieten würde. Hierbei bezieht sich der Begriff der Leistungsfähigkeit auf den Problemlöseprozeß und nicht auf die berechnete Lösung. Leistungsfähigkeit umfaßt sowohl die Effizienz als auch die Kompetenz des Problemlöseprozesses. Verbesserung der Effizienz bedeutet, daß im Mittel die zu lösenden Probleme mit geringerem Zeitaufwand gelöst werden können als durch einen generischen Problemlöser; Verbesserung der Kompetenz zielt darauf ab, Probleme zu lösen, die durch einen generischen Problemlöser beim Einsatz gleicher Rechenzeitressourcen nicht

gelöst werden können. Wichtig ist, daß diese Verbesserungen in einem Bereich liegen, der von praktischer Relevanz ist.[1]

Korrektheit: Die Wiederverwendung von Fällen muß zu einer korrekten Lösung des neuen Problems führen. Eine Verletzung dieser Anforderung würde einen Nachteil gegenüber generischen Methoden bedeuten, da diese die Korrektheit der Lösung zusichern.

Flexibilität: Zur Problemlösung muß ein Fall flexibel einsetzbar sein, d. h. für möglichst viele neue Problemstellungen zu einer höheren Leistungsfähigkeit im Vergleich zu generischem Problemlösen führen. Dies ist besonders dann wichtig, wenn nur eine geringe Anzahl von Fällen verfügbar ist und die zu lösenden Probleme stark variieren. Hierbei sollte sich die Leistungsfähigkeit des fallverarbeitenden Problemlösers an der *Ähnlichkeit* zwischen der neuen Problemstellung und einem vorliegenden Fall orientieren. Ist eine aktuelle Problemstellung einem bereits bekannten Fall sehr ähnlich, so wird eine hohe Effizienz des Problemlösers bei der Lösung des neuen Problemes erwartet. Je unähnlicher die neue Problemstellung zu einem verfügbaren Fall ist, desto eher kann eine geringere Effizienzsteigerung akzeptiert werden. Ist im Extremfall die neue Problemstellung vollkommen von den vorliegenden Fällen verschieden, so kann der Problemlöseprozeß auch rein generisch sein, ohne daß eine Leistungssteigerung bewirkt wird.

Automatische Fallverarbeitung: Fälle sollten ohne manuelle Aufbereitung durch einen Experten wiederverwendbar sein, da dies, insbesondere bei großen Fallmengen, einen erheblichen Aufwand verursachen würde. Daher wollen wir auch auf eine Beschreibung des Problemlöseprozesses, aus dem der Fall hervorgegangen ist, verzichten. Fälle, nur bestehend aus Problemstellung und Lösung, sollen daher vollautomatisch wiederverwendet werden können.

Inkrementelle Erweiterbarkeit: Die zu entwickelnde Methodik zur Verarbeitung von Fällen sollte außerdem in der Lage sein, das Problemlöseverhalten aufgrund von Erfahrungen aus dem eigenen Problemlöseprozeß zu verbessern. Daher muß die inkrementelle Erweiterung durch neue Fälle unterstützt werden.

Domänenunabhängigkeit: Die Methodik muß unabhängig von einer speziellen Anwendungsdomäne verwendbar sein. Das erforderliche domänenspezifische Wissen muß in einer deklarativen Form repräsentiert und verarbeitet werden. Andernfalls wäre das Einsatzspektrum der Methodik zu stark eingeschränkt.

Wichtig ist, daß alle diese Anforderungen gleichzeitig erfüllt werden können. Selbst der Verzicht auf eine dieser Anforderungen wird in der Regel dazu führen, daß der Einsatz eines fallbasierten Problemlösers unrentabel ist.

[1] Weder eine Verkürzung der Problemlösezeit von einer Millisekunde auf eine Mikrosekunde, noch eine Verkürzung von 1000 Jahren auf ein Jahr ist von praktischer Relevanz. Bedeutsam ist jedoch z. B. eine Verkürzung der Problemlösezeit von einem Tag auf zwei Minuten.

1.1.3 Methodik

An dieser Stelle sollen bereits die in dieser Arbeit entwickelten Methoden kurz skizziert und motiviert werden. Die Grundidee besteht darin, *Abstraktions-* und *Generalisierungsmethoden* bei der Wiederverwendung von Fällen auszunutzen. Während Abstraktion und Generalisierung oft synonym benutzt werden, unterscheiden wir in dieser Arbeit genau zwischen beiden Begriffen. Kernpunkt der Abstraktion ist eine Reduktion des Detailliertheitsgrades einer Beschreibung, was häufig mit dem Wechsel der Beschreibungssprache verbunden ist. Eine Generalisierung hingegen faßt die Beschreibung mehrerer Objekte zu einer gemeinsamen, allgemeineren Beschreibung zusammen, ohne den Detailliertheitsgrad oder die Beschreibungssprache zu wechseln.

Im Verlauf dieser Arbeit werden zwei neue Methoden zur Wiederverwendung von Planungsfällen vorgestellt: die Wiederverwendung von *abstrakten Fällen* und die Wiederverwendung von *generalisierten Fällen*. Darüber hinaus lassen sich beide Methoden kombinieren. Wir sprechen in diesem Fall von der Wiederverwendung *generalisierter abstrakter Fälle*. Diesen Methoden (und ihrer Kombination) ist gemeinsam, daß vorliegende Fälle zunächst in einer *Lernphase* vorverarbeitet („kompiliert") werden. Hierdurch werden sie in eine Form gebracht, die ihre spätere effiziente Wiederverwendung in der *Problemlösephase* ermöglicht.

Wiederverwendung abstrakter Fälle: *Abstraktion* ist eine bekannte Technik, die z. B. innerhalb der Künstlichen Intelligenz beim hierarchischen Problemlösen eingesetzt wird. Grundlegend ist hierbei die Idee, Probleme und Lösungen auf verschiedenen Abstraktionsebenen zu betrachten. Abstraktionsebenen können nach ihrem Abstraktionsgrad (bzw. Detailliertheitsgrad) angeordnet werden. Beim klassischen hierarchischen Problemlösen werden Probleme „top-down" gelöst. Beginnend bei der höchsten Abstraktionsebene wird eine abstrakte Problemlösung generiert, die dann schrittweise auf den nächst tieferen Abstraktionsebenen konkretisiert (verfeinert) wird. Hierbei bearbeitet der Problemlöser auf jeder Abstraktionsebene nur das jeweils Relevante. Hierdurch wird vermieden, daß zu früh ein unnötiger Rechenaufwand für die Bestimmung irrelevanter Details betrieben wird.

In dieser Arbeit wird eine Methode vorgeschlagen, die es ermöglicht, die in einem Fall enthaltene Problemlösung, je nach Erfordernis, auf verschiedenen Abstraktionsebenen wiederzuverwenden. Wir sprechen daher von der Wiederverwendung abstrakter Fälle. Ist eine neue Problemstellung zu einem vorliegenden Fall sehr ähnlich, so kann die im Fall enthaltene Lösung auf einer niedrigen Abstraktionsebene, mit vielen Details, direkt und ohne Modifikation wiederverwendet werden. Weicht die neue Problemstellung jedoch stärker von dem vorliegenden Fall ab, so muß die im Fall enthaltene Lösung auf einer höheren Abstraktionsebene, d. h. mit weniger Details, wiederverwendet werden. Hierbei verbleibt ein gewisser Aufwand, um die fehlenden Details durch Verfeinerung der abstrakten Lösung zu generieren.

Wiederverwendung generalisierter Fälle: Die zweite Methode zur Wiederverwendung von Fällen basiert auf der expliziten Generalisierung von Fällen. Während ein Fall eine spezielle Problemstellung und eine spezielle Lösung beschreibt, enthält ein *generalisierter Fall* eine *Problemklasse* und eine *Lösungsklasse*. Ein neues Problem kann mit einem generalisierten Fall gelöst werden, wenn das Problem in die Problemklasse des generalisierten Falls fällt. Diesem neuen Problem wird dann eine spezielle Lösung aus der Lösungsklasse zugeordnet.

Bei dieser Wiederverwendungsmethode spielt die im maschinellen Lernen entwickelte Technik des *erklärungsbasierten Lernens* eine wichtige Rolle. Diese Technik kann dazu eingesetzt werden, generalisierte Fälle automatisch aus speziellen Fällen zu bilden. Im Gegensatz zu induktiven Generalisierungsverfahren kann durch das erklärungsbasierte Generalisieren die Korrektheit des generalisierten Falls sichergestellt werden, d. h. ein generalisierter Fall weist einem Problem der Problemklasse immer eine korrekte Lösung zu.

Wiederverwendung generalisierter abstrakter Fälle: Die dritte Methode, die Wiederverwendung generalisierter abstrakter Fälle, ist eine Kombination der beiden zuvor beschriebenen Methoden. Hierbei werden generalisierte Fälle, d. h. Problem- und Lösungsklassen, auf verschiedenen Abstraktionsebenen wiederverwendet. Technisch läßt sich diese Art der Wiederverwendung durch Hintereinanderausführung der Algorithmen, die für die Wiederverwendung generalisierter bzw. abstrakter Fälle erforderlich sind, realisieren.

Effizientes Fallretrieval: Bei allen fallverarbeitenden Ansätzen stellt sich das Problem der Organistation der Fallbasis, insbesondere dann, wenn große Fallmengen vorliegen. Hierzu verwenden wir eine Methode, bei der auf die Fallbasis mittels einer *Wiederverwendbarkeitshierarchie* zugegriffen wird. In dieser Hierarchie sind Fälle mit einem höheren Abstraktions- oder Generalisierungsgrad „oberhalb“ von Fällen mit einem niedrigen Abstraktions- oder Generalisierungsgrad angeordnet. Beim Fallretrieval kann diese Hierarchie „top-down“ durchlaufen werden, bis ein Fall gefunden ist, der mit möglichst geringem Aufwand wiederverwendet werden kann. Ein Kostenmodell, das die Wiederverwendungskosten und die Retrievalkosten gleichermaßen berücksichtigt, erlaubt es darüber hinaus das Retrieval durch eine geeignete Wahl der Reihenfolge von Nachfolgerknoten in der Hierarchie sowie durch „vergessen“ von ungünstigen Fällen weiter zu verbessern. Dies geschieht durch den Einsatz von Optimierungsverfahren.

1.1.4 Voraussetzungen

Für den Einsatz der entwickelten Methodik müssen einige Voraussetzungen erfüllt sein. Die Grundvoraussetzung dafür, daß sich der Einsatz einer fallverarbeitenden Methodik überhaupt lohnt ist, daß keine Spezialalgorithmen exisitieren, mit denen Probleme des vorliegenden Problembereichs effizient gelöst werden können. Sind solche effizienten Al-

gorithmen vorhanden, so sind diese immer vorzuziehen, da dann auf den Aufwand zur Fallakquisition verzichtet werden kann.

Darüber hinaus müssen einige wichtige, zum Teil offensichtliche, allgemeine Voraussetzungen erfüllt sein.

- Für den vorliegenden Problembereich müssen Fälle und ihre Beschreibung bereits verfügbar sein bzw. aus vorhandenen Aufzeichnungen mit geringem Aufwand konstruiert werden können.
- Das Spektrum der erwarteten zukünftigen Probleme sollte annähernd bekannt sein und es sollten überwiegend Probleme vorkommen, die sich nicht stark von den vorliegenden Fällen unterscheiden.
- Bei starken Änderungen im Spektrum der zu lösenden Probleme sollten Bereichsexperten in der Lage sein, neuartige Probleme exemplarisch ohne Systemunterstützung zu lösen und als neue Fälle dem System verfügbar zu machen.

Außerdem stellt die in dieser Arbeit entwickelte Methodik weitere, zum Teil technische Anforderungen, die wir bereits an dieser Stelle kurz erwähnen möchten. Diese Anforderungen werden im weiteren Verlauf der Arbeit noch präzisiert.

- Der vorliegende Problembereich muß als Aktionsplanungsaufgabe formulierbar sein. Dies erfordert die Definition einer Grundmenge von Situationen (Zuständen) sowie einer Menge von Aktionen, die Situationen verändern können. Was dies genau bedeutet, wird in Kapitel 2 beschrieben.
- Die vorliegenden Fälle müssen im Rahmen der definierten Aktionsplanungsaufgabe formalisierbar sein, d. h. die Problemstellung muß als Paar, bestehend aus Anfangs- und Zielsituation repräsentiert werden, und die Lösung als Sequenz verfügbarer Aktionen.
- Der vorliegende Problembereich muß auf verschiedenen Abstraktionsebenen beschrieben werden können. Hierbei muß es möglich sein, die verschiedenen Abstraktionsebenen als eigenständige Handlungsplanungsaufgaben zu beschreiben. Einige technische Bedingungen, die Repräsentation der Abstraktionsebenen betreffend, müssen zusätzlich erfüllt sein. Diese werden in Abschnitt 8.1 ausführlich beschrieben.

1.1.5 Bewertungskriterien

Um die entwickelten Methoden zu testen und detailliert bewerten zu können, wurden sie in einem System, PARIS genannt, vollständig implementiert. PARIS ist ein Akronym für **P**lan **A**bstraction and **R**efinement in an **I**ntegrated **S**ystem. Mit diesem System läßt sich die Funktionsweise der entwickelten Methodik beispielhaft demonstrieren. Der wesentliche

Zweck der Implementierung besteht jedoch darin, verschiedene empirische Untersuchungen durchzuführen. Dies ist für eine umfassende Beurteilung des Ansatzes erforderlich, da einige der gestellten Anforderungen – hier ist insbesondere die Leistungsfähigkeit und die Flexibilität gemeint – nicht durch rein theoretische Betrachtungen beurteilt werden können. Zur Durchführung der empirischen Untersuchungen wurde daher ein Teilbereich aus dem Maschinenbau, nämlich die Arbeitsplanung von rotationssymmetrischen Werkstücken auf einer Drehbank, ausgewählt und exemplarisch modelliert. Hierbei ist es jedoch nicht das Ziel, das Problem der Arbeitsplanung in allen seinen Details vollständig zu lösen oder gar ein einsatzfähiges System zu entwickeln. Hierzu hätten eine Reihe weiterer Probleme gelöst werden müssen, wie z. B. die Integration in eine CAD/CAM Umgebung. Dies wird von der entwickelten Anwendung nicht abgedeckt. Die Modellierung umfaßt jedoch einen realistischen und komplexen Teilbereich, bei dem ein generischer Problemlöser typischerweise bei der Problemlösung scheitert. Dies ermöglicht die Auswahl von realitätsnahen Testszenarien, so daß verhindert werden kann, daß die sich ergebenden experimentellen Befunde fern jeglicher praktischer Bedeutung und für reale Anwendungen nicht übertragbar sind.

1.2 Vorbemerkungen zur Arbeit

Es folgen einige Vorbemerkungen zur Arbeit, die die Anforderungen an den Leser, sowie die in der Arbeit vorkommenden Beschreibungsebenen betreffen.

1.2.1 Voraussetzungen an den Leser

Im weiteren Verlauf der Arbeit wird angenommen, daß der Leser zumindest Grundkenntnisse aus dem Bereich der Künstlichen Intelligenz besitzt. Insbesondere Grundkenntnisse in den klassischen KI-Gebieten der Logik, des Problemlösens im Sinne von GPS oder STRIPS und der Suche erforderlich. Darüber hinaus wird eine grundsätzliche Vertrautheit mit den Gebieten des maschinellen Lernens – insbesondere des erklärungsbasierten Lernens – und des fallbasierten Schließens vorausgesetzt. Literaturverweise auf einführende Überblicksarbeiten zu diesen Gebieten werden jedoch an geeigneter Stelle gegeben und ermöglichen es dem fachfremden Leser sich die erforderlichen Grundkenntnisse anzueignen.

Kenntnisse im Anwendungsbereich der Arbeitsplanung werden nicht vorausgesetzt. Die Beispiele, die zur Verdeutlichung aus diesem Bereich herangezogen werden, sind ausführlich erklärt und sollten allgemein verständlich sein.

1.2.2 Darstellungsebenen

Im Verlauf dieser Arbeit werden die vorgestellten Methoden auf drei verschiedenen Ebenen diskutiert.

- Auf einer informellen Ebene werden zunächst allgemeine Fragestellungen, Motivationen und Lösungsansätze erörtert. Hierdurch soll dem Leser der erste Zugang zu den vorgestellten Methoden erleichtert werden.

- Wichtige Grundbegriffe werden dann – ebenfalls noch informell – durch *Begriffsbestimmungen* eingeführt. Die hierdurch entstehende Begriffsbildung strukturiert die Beschreibung der Methoden wesentlich.

- Schließlich werden die zentralen Begriffe durch *Definitionen* formalisiert. Diese werden dann zur algorithmischen Beschreibung der Methoden eingesetzt. *Lemmata* und *Sätze* drücken wichtige Eigenschaften der Algorithmen aus.

1.3 Gliederung der Arbeit

Die weitere Arbeit ist in drei Hauptteile gegliedert, wobei jeder Hauptteil wieder in verschiedene Kapitel unterteilt ist.

Teil I: Grundlagen und Basismethoden (Kapitel 2 und 3)

Dieser Teil enthält eine detaillierte Beschreibung der Grundlagen der Arbeit sowie eine kurze Darstellung der grundlegenden Methoden, die im weiteren Verlauf der Arbeit eine Rolle spielen. *Kapitel 2* führt in die Begriffswelt des Problemlösens ein und stellt die in dieser Arbeit verwendeten Repräsentationsformalismen für Problemlösedomänen dar. Danach werden die Grenzen dieses Problemlösebegriffes diskutiert und die grundlegendsten Ansätze aus dem Bereich der Suche und Planung besprochen. In *Kapitel 3* führen wir die für diese Arbeit relevanten Begriffe und Ideen aus den Gebieten des erklärungsbasierten Lernens, des fallbasierten Schließens und der Abstraktion ein.

Teil II: PARIS - Flexible Wiederverwendung von Fällen durch Abstraktion und Generalisierung (Kapitel 4 bis 7)

Dieser Teil beinhaltet den wesentlichen Teil der vorliegenden Arbeit. *Kapitel 4* gibt einen vollständigen Überblick über den gesamten Ansatz und das Zusammenwirken der einzelnen Methoden. An einem einfachen Beispiel aus dem Bereich der Arbeitsplanung wird dabei die grundsätzliche Vorgehensweise veranschaulicht. Die folgenden drei Kapitel beschreiben dann die einzelnen Methoden im Detail. In *Kapitel 5* wird der Ansatz zur Wiederverwendung von abstrakten Fällen dargestellt. *Kapitel 6* beschreibt die Wiederverwendung von generalisierten Fällen und zeigt den Einsatz von erklärungsbasiertem Generalisieren. Der Ansatz zur Organisation der Fallbasis wird in *Kapitel 7* beschrieben.

Teil III: Anwendung, Bewertung und Diskussion (Kapitel 8 bis 10)

Der abschließende Teil dieser Arbeit befaßt sich mit der Bewertung und Diskussion des Ansatzes. In *Kapitel 8* wird eine Anwendung auf den Bereich der Arbeitsplanung beschrieben. In einer Reihe empirischer Untersuchungen wird die Effizienz, Kompetenz und Flexibilität des Problemlösens durch Wiederverwendung unter verschiedenen Bedingungen untersucht. In *Kapitel 9* werden verwandte Arbeiten diskutiert und verglichen. *Kapitel 10* faßt die wesentlichen Teile dieser Arbeit zusammen und erörtert, inwieweit die ursprünglichen Anforderungen erfüllt werden konnten. Die Arbeit endet mit einem Ausblick auf zukünftige Fragestellungen im Lichte der hier erzielten Ergebnisse.

Teil I

Grundlagen und Basismethoden

Kapitel 2

Grundlagen aus dem Bereich des Problemlösens

Die allgemeine Sicht auf Problemlösen, die in dieser Arbeit zugrunde gelegt wird, geht zurück auf frühe Arbeiten aus den Anfängen der Künstlichen Intelligenz. Zu dieser Zeit wurde der Begriff des Problemlösens gebildet, überwiegend geprägt von der Vorstellung eines universellen Problemlösers (engl. general problem solver, GPS) [Ernst & Newell, 1969; Newell & Simon, 1972]. Hierbei lag die Hypothese zugrunde, daß es einen universellen Mechanismus menschlichen Problemlösens gibt, der als berechenbarer Algorithmus auf einem Computer ablauffähig gemacht werden kann und so intelligentes Verhalten realisiert. Diese Arbeiten waren überwiegend durch das Ziel geprägt, kognitive Modelle menschlichen Problemlösens zu finden und diese als Computerprogramme zu operationalisieren. Primäres Ziel war hierbei jedoch nicht, effizientes Problemlösen auf einem Rechner zu realisieren. Obwohl Letzteres heute überwiegend im Mittelpunkt der KI-Forschung steht, wurde an der ursprünglichen Sicht auf Problemlösen bis heute festgehalten, auch wenn sich zwischenzeitlich viele eigenständige Spezialisierungen, insbesondere in den Forschungsgebieten der Suche (z. B. [Korf, 1980; Korf, 1987; Korf, 1993]) und der Aktionsplanung (z. B. [Georgeff, 1987; Herzberg, 1989; Weld, 1994]) herausgebildet haben.

In Abschnitt 2.1 wird der Begriff des *Problemlösens* zunächst informell eingeführt. Hierdurch wird zunächst ein Eindruck davon vermittelt, welche Problembereiche überhaupt behandelt werden sollen. Danach wird in Abschnitt 2.2 eine Syntax zur formalen Repräsentation von Problemlösedomänen in Anlehnung an STRIPS [Fikes & Nilsson, 1971] eingeführt. Zur Motivation der syntaktischen Konstrukte, wird deren Semantik bereits kurz informell eingeführt. Wir werden ebenfalls darlegen, wie syntaktische Schlußfolgerungen in Problemlösedomänen gezogen werden können. Die formale Semantik dieser Repräsentation wird separat in Abschnitt 2.3 erörtert. In Abschnitt 2.4 werden dann die Grenzen dieses Problemlösebegriffes diskutiert. Abschließend geben wir in Abschnitt 2.5 noch einen kurzen Überblick über generische Verfahren zur Problemlösung.

2.1 Problemlösen

2.1.1 Veränderliche Welten

Beim Problemlösen in dem hier verwendeten Sinne steht die Betrachtung eines Systems verschiedener möglicher *Welten* oder *Situationen* im Vordergrund. Diese verschiedenen Welten stehen dadurch im Zusammenhang, daß es bestimmte Möglichkeiten gibt, um von einer Welt in eine andere zu gelangen. Solche, rein diskreten Übergänge werden durch *Aktionen* ausgelöst. Eine Problemlösedomäne legt nun die betrachteten Welten und die Aktionen fest.

Begriffsbestimmung 2.1 (Problemlösedomäne, Domäne) Eine *Problemlösedomäne* (oder kurz *Domäne*) besteht aus einem System verschiedener Welten sowie einer Menge von Aktionen, die Übergänge von einer Welt in eine andere Welt darstellen.[1]

Zur Klarheit soll bereits an dieser Stelle ein Beispiel erwähnt werden, welches wir im Verlauf dieser Arbeit weiter verfolgen werden.

Beispiel 2.1 Im Bereich der Arbeitsplanung geht es darum, ein bestimmtes Teil (z. B. ein Werkstück) zu fertigen. Eine Welt ist hierbei ein solches Teil in einem bestimmten Fertigungszustand. Das System verschiedener Welten ist die Menge aller möglichen Teile in den möglichen verschiedenen Fertigungszuständen. Die Aktionen stellen Fertigungsschritte dar die das Werkstück verändern und somit eine Welt in eine andere überführen.

Diese verschiedenen Welten mit den möglichen Übergängen kann man sich als einen Graphen vorstellen. Die Knoten dieses Graphen sind Welten, und die Kanten sind Übergänge, die mit einer Aktion markiert sind.[2]

Beim Problemlösen sind wir nun nicht daran interessiert, Aussagen innerhalb einer solchen Welt zu machen, sondern vielmehr Aussagen darüber, mit welcher *Aktionsfolge* eine Welt in eine andere überführt werden kann.[3] Daher betrachten wir Probleme und Lösungen der folgenden Art:

Begriffsbestimmung 2.2 (Problem) Ein *Problem* ist ein Paar von Welten (initiale Welt, Zielwelt).

[1] Die Bedeutung des Begriffs „Welt“ entspricht hier der in der Modallogik. Eine Einführung in die Modallogik findet man in KI-Lehrbüchern wie z. B. in [Richter, 1992b].

[2] Solche Graphen kommen auch in der Modallogik in Form von Graphenmodellen vor.

[3] Dies ist ein wesentlicher Unterschied zu analytischen Aufgaben wie z. B. Klassifikation. Hierbei ist man in der Regel an Aussagen innerhalb einer einzigen Welt interessiert.

Begriffsbestimmung 2.3 (Lösung, Plan) Eine *Lösung*, auch *Plan* genannt, ist eine endliche zusammenhängende Folge von Aktionen.

Der Korrektheitsbegriff für eine Lösung präzisiert sich wie folgt:

Begriffsbestimmung 2.4 (Korrekte Problemlösung) Ein Plan ist eine korrekte Lösung eines Problems, wenn er die initiale Welt in die Zielwelt überführt. Das *Problemlösen* besteht dann in der Erstellung eines solchen Plans für ein gegebenes Problem.

2.1.2 Eine Beispieldomäne: Arbeitsplanung für rotationssymmetrische Drehteile

Im folgenden wollen wir die bereits in Beispiel 2.1 skizzierte Problemlösedomäne etwas genauer vorstellen, da wir diese im weiteren Verlauf dieser Arbeit zur Demonstration verschiedener Techniken einsetzen werden. Hierbei handelt es sich um einen Ausschnitt einer praktisch relevanten Anwendung, nämlich der Arbeitsplanung für die Fertigung rotationssymmetrischer Drehteile auf einer Drehbank.

Rotationssymmetrische Drehteile werden üblicherweise auf einer computergesteuerten Drehbank (CNC-Maschine) gefertigt. Hierbei wird aus einem meist zylindrischen Rohling das gewünschte Werkstück hergestellt. Der Rohling wird in das Spannfutter der Drehmaschine eingespannt und in Rotation versetzt. Während sich der Rohling mit hoher Geschwindigkeit um seine Rotationsachse dreht, wird durch verschiedene Schneidwerkzeuge ein Teil des Materials des Rohlings abgetragen, bis letzlich die gewünschte Kontur des Zielwerkstücks entsteht. Bei diesem Prozeß ist die Reihenfolge, in der die verschiedenen Bereiche des Rohlings abgetragen werden, sowie die hierzu verwendeten Schneidwerkzeuge von entscheidender Bedeutung. Das Auffinden einer Bearbeitungsreihenfolge, die es erlaubt ein gewünschtes Zielwerkstück zu fertigen, ist somit die zentrale Aufgabenstellung.

Diese Domäne kann wie folgt als Problemlösedomäne beschrieben werden: Eine Welt ist ein Werkstück, das sich in einem bestimmten Bearbeitungsstadium befindet. Aktionen stellen die durch die Drehmaschine durchführbaren möglichen Bearbeitungen dar. Dies können z. B. Aktionen sein, die ein Werkstück (z. B. den Rohling) in das Spannfutter der Drehmaschine einspannen oder Aktionen, die einen ganz speziellen Schnitt durchführen, also einen bestimmten Bereich des Materials des Werkstückes abtragen. Abbildung 2.1 zeigt einen Ausschnitt möglicher Welten und Übergänge in einem Graphen.

Ein Problem in dieser Domäne wird durch die Angabe des Rohlings (initiale Welt) und des gewünschten Werkstücks (Zielwelt) spezifiziert. In Abbildung 2.1 ist eine typische initiale Welt (zylindrischer Rohling) auf der linken Seite des Graphen dargestellt; mögliche Zielwelten (verschieden geformte Werkstücke) sind am rechten Rand des dargestellten Ausschnitts des Graphen zu sehen. Die einzigen beiden Aktionen, die ausgehend von der initialen Welt möglich sind, sind das Aufspannen des Werkstückes auf der linken oder der rechten Seite. Ist das Werkstück aufgespannt, so sind verschiedene Schnittaktionen durchführbar, von

denen einige im dargestellten Graphen angezeigt sind. Außerdem kann ein eingespanntes Werkstück auch wieder ausgespannt werden.

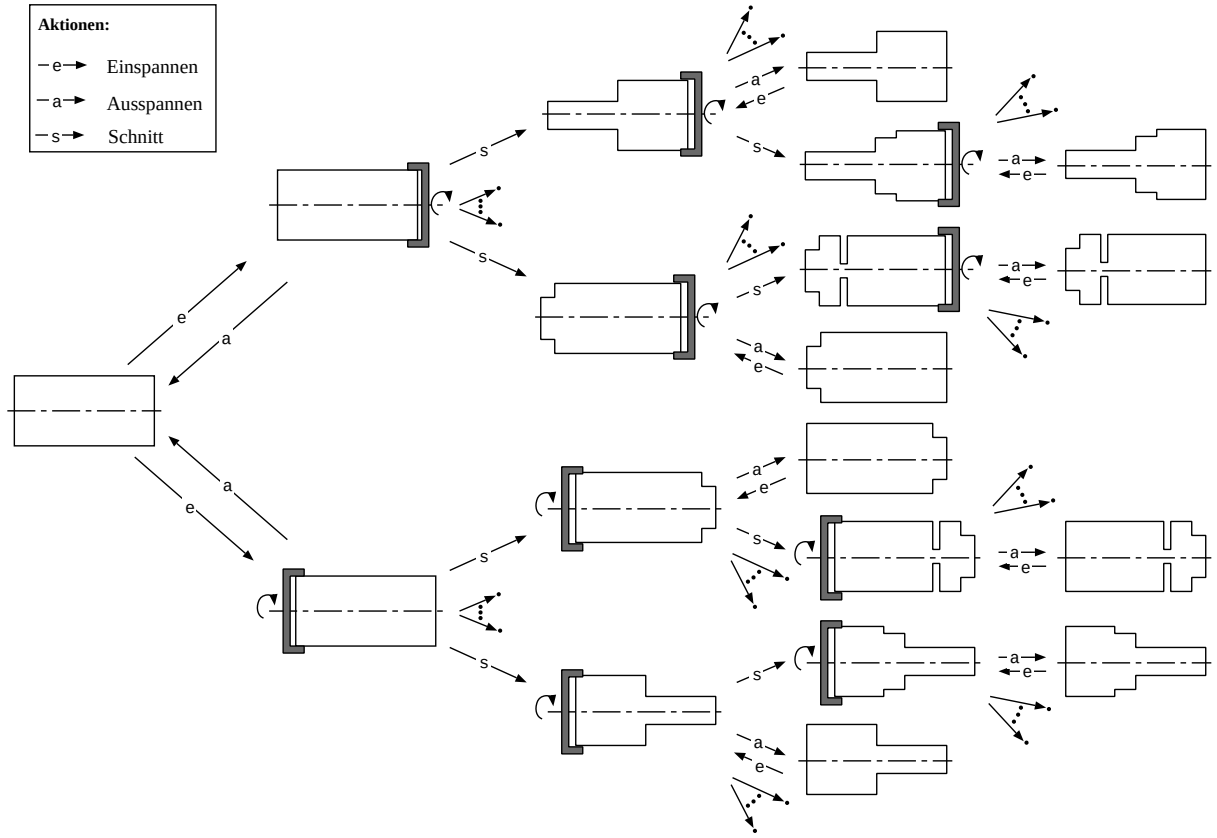

Abbildung 2.1: Beispiel für einen Ausschnitt möglicher Welten und Aktionen für die Domäne der Arbeitsplanung rotationssymmetrischer Drehteile, dargestellt in Form eines Graphen. Die Knoten des Graphen (Welten) sind durch verschieden geformte und verschieden eingespannte Werkstücke repräsentiert. Die Kanten geben mögliche Aktionen an.

2.2 Formale Repräsentation (Syntax)

Zur weiteren Formalisierung dieses Problemlösebegriffes bieten sich prinzipiell mehrere Möglichkeiten an. Die Modallogik ist eine spezielle Erweiterung der Prädikatenlogik erster Stufe zur Behandlung verschiedener Welten. Diese eignet sich prinzipiell zur Formalisierung von Problemlösen; ihre Mächtigkeit wird hier jedoch in vielen Fällen nicht benötigt. Daher wählen wir einen anderen Formalisierungsansatz, der unter dem Namen STRIPS (*ST*anford *R*esearch *I*nstitute *P*roblem *S*olver) [Fikes & Nilsson, 1971] bekannt geworden

ist.[4] In STRIPS werden verschiedene Welten durch verschiedene Zustände – das sind Mengen von aussagenlogischen oder prädikatenlogischen Formeln – repräsentiert. Aktionen werden durch Operatoren, die Veränderungen der Zustände durch Hinzunahme und Entfernen von Formeln unter bestimmten Voraussetzungen beschreiben, dargestellt. Aussagen, die sich auf eine bestimmte Welt beziehen, können hierbei durch aussagenlogische oder prädikatenlogische Ableitungen auf der Grundlage der Formeln eines Zustandes realisiert werden; bestimmte Aussagen über die Erreichbarkeit von Welten durch Aktionen werden durch Operatoranwendungen, die neue Zustände bestimmen, realisiert.

Nachfolgend werden wir eine Repräsentation von Problemlösedomänen (Syntax) einführen, die auf einer prädikatenlogischen Sprache erster Stufe aufbaut. Wir werden hierbei keine Erweiterung der Prädikatenlogik selbst vornehmen, sondern eine Metasprache einführen, die die Prädikatenlogik verwendet.

2.2.1 Relevante Grundbegriffe der Prädikatenlogik

Wir gehen von den in der Prädikatenlogik üblichen Begriffen und Bezeichnungen aus, die wir hier kurz zusammengefassen. Für eine ausführliche Einführung in die Prädikatenlogik sei an dieser Stelle auf die einschlägige Literatur verwiesen [Richter, 1978; Richter, 1992b; Lloyd, 1984].

Eine prädikatenlogische Sprache erster Stufe $\mathcal{L}$ wird definiert über einem *Alphabet*, bestehend aus disjunkten Mengen von Symbolen für

- *Konstanten* (z. B. a, b, c),
- *Variablen* (z. B. x, y, z),
- *Funktionen* (z. B. f, g, h) und
- *Prädikaten* (z. B. p, q, r).

Darüber hinaus beinhaltet das Alphabet die üblichen logischen Verknüpfungsoperatoren $\wedge, \vee, \neg, \rightarrow, \forall$ und $\exists$, sowie die üblichen Klammersymbole (und). Jedes Funktionssymbol besitzt eine *Stelligkeit* $n > 0$[5] und jedes Prädikatssymbol hat eine *Stelligkeit* $n \geq 0$.

Für eine pradikatenlogische Sprache ist die Menge der *Terme* wie folgt definiert: Jede Konstante und jede Variable ist ein Term. Falls f ein Funktionssymbol mit der Stelligkeit n ist und falls $t_1, \ldots, t_n$ Terme sind, dann ist auch $f(t_1, \ldots, t_n)$ ein Term. Terme, die keine Variablen enthalten, werden auch als *Grundterme* bezeichnet.

[4] STRIPS ist auch heute die Grundlage vieler Repräsentationsformen für Problemlösedomänen wie z. B. die der Sprache PDL (*P*rodigy *D*escription *L*anguage) [Minton, 1988; Blythe *et al.*, 1992] des PRODIGY Systems [Minton, 1988; Carbonell *et al.*, 1991; Veloso & Carbonell, 1993]. In PDL wurden als wesentliche Erweiterungen von STRIPS sogenannte Inferenzregeln, das sind im wesentlichen Implikationen, und in der neuesten Version ein Typkonzept für Variablen eingeführt.

[5] 0-stellige Funktionen sind Konstanten, die als eigene Symbole eingeführt sind.

Aus Prädikaten und Termen können nun *atomare Formeln*, auch *Atome* genannt, aufgebaut werden. Ist p ein n-stelliges Prädikatssymbol und sind $t_1, \ldots, t_n$ Terme, dann ist $p(t_1, \ldots, t_n)$ eine atomare Formel. Atomare Formeln können mit Hilfe von logischen Verknüpfungsoperatoren zu *Formeln* kombiniert werden. Hierbei ist jede atomare Formel auch eine Formel. Sind F und G Formel, so sind auch $(F \wedge G)$, $(F \vee G)$, $(\neg F)$ und $(F \rightarrow G)$ Formeln. Ist F eine Formel und x eine Variable, dann sind auch $(\forall x F)$ und $(\exists x F)$ Formeln. Als *Grundformeln* bzw. *Grundatome* werden Formeln bzw. Atome bezeichnet, in denen keine Variablen vorkommen.

Hornklauseln sind spezielle Formeln der Gestalt $\forall x_1, \ldots, \forall x_n(A_1 \wedge \ldots \wedge A_m \rightarrow B)$, wobei $A_1 \ldots A_m$ und B Atome sind. In den Atomen $A_1 \ldots A_m$ und B kommen genau die Variablen $x_1, \ldots, x_n$ vor. Hornklauseln mit $m = 0$, also $\forall x_1, \ldots, \forall x_n B$, werden auch *Fakten* genannt. In der logischen Programmierung hat sich für Hornklauseln eine spezielle Schreibweise etabliert, bei der die Quantoren weggelassen, die Konjunktionszeichen durch Kommas ersetzt werden und die gesamte Implikation in umgekehrter Reihenfolge aufgeschrieben wird. Eine Hornklausel wird notiert als $B \leftarrow A_1, \ldots, A_m$. Fakten werden einfach durch weglassen der Quantoren notiert, also z. B. B.

Eine *Substitution* σ ist eine Menge von Variablenbindungen der Form $\{x_1/t_1, \ldots, x_n/t_n\}$. Ist t ein Term, so bezeichnet $t\sigma$ den Term, der durch die gleichzeitige Ersetzung aller Variablen x_i durch t_i (für $i = 1 \ldots n$) entsteht. Der Term $t\sigma$ wird auch *Instanz* von t genannt. Ist F eine Formel, so bezeichnet $F\sigma$ die Formel, die durch die gleichzeitige Ersetzung aller Variablen x_i durch t_i (für $i = 1 \ldots n$) entsteht. Die Formel $F\sigma$ wird auch *Instanz* von F genannt. Analog läßt sich die Anwendung einer Substitution auch auf eine Menge von Termen oder Formeln definieren. Eine Substitution wird *Grundsubstitution* genannt, falls alle t_i $(i = 1 \ldots n)$ Grundterme sind.

Für die Prädikatenlogik erster Stufe gibt es vollständige und korrekte Beweiskalküle. Ein syntaktischer Ableitungsoperator wird mit $\vdash$ notiert, und $\Sigma \vdash F$ besagt, daß die Formel F aus der Formelmenge Σ syntaktisch folgt.

2.2.2 Repräsentation von Welten

Beim Problemlösen ist man daran interessiert, bestimmte Aktionsfolgen zu finden, deren Ausführung eine Welt auf bestimmte Weise verändert; man ist nicht daran interessiert, bestimmte Aussagen über eine Welt selbst zu machen. Daher ist es nicht erforderlich, die betrachteten Welten so vollständig wie möglich zu formalisieren. Es müssen nur die Eigenschaften formalisiert werden, die bekannt sein müssen, um zwischen verschiedenen möglichen Aktionen und deren Ausgang differenzieren zu können.

Bei der Beschreibung einer Welt unterscheiden wir im folgenden zwischen einer dynamischen und einer statischen Beschreibung. Die dynamische Beschreibung umfaßt Eigenschaften, die sich in verschiedenen Welten unterscheiden können. Die statische Beschreibung enthält Eigenschaften, die in allen Welten gelten.

Dynamischer Beschreibungsanteil

Wir drücken bestimmte Eigenschaften, in denen sich verschiedene Welten voneinander unterscheiden können, durch Grundatome der zugrunde liegenden Sprache $\mathcal{L}$ aus. Diese werden nach Lifschitz [1987] *essentielle Sätze* genannt werden. Nicht alle Eigenschaften, in denen sich verschiedene Welten unterscheiden können, müssen durch essentielle Sätze ausgedrückt werden oder ausdrückbar sein. Hierauf werden wir genauer in den Abschnitten 2.2.2 und 2.2.3 eingehen.

Bezeichnung 2.1 (Essentielle Sätze, essentielle Satzformen)

(i) Wir zeichnen eine Menge von Grundatomen der Sprache $\mathcal{L}$ aus, die wir *essentielle Sätze* nennen und mit $\mathcal{E}^*$ bezeichnen.
(ii) Eine Menge von essentiellen Sätzen wird durch eine endliche Menge von *essentiellen Satzformen*, die mit $\mathcal{E}$ bezeichnet wird, wie folgt bestimmt:
$\mathcal{E}^* = \{e\sigma \mid e \in \mathcal{E}$ und σ ist eine Substitution, und $e\sigma$ ist ein Grundatom $\}$.

Bemerkung 2.1 Durch eine endliche Menge von essentiellen Satzformen $\mathcal{E}$ kann man ggf. eine unendliche Menge von essentiellen Sätzen $\mathcal{E}^*$ beschreiben. Enthält $\mathcal{E}$ beispielsweise das Atom $p(a, x)$ [6], so sind alle Grundatome, die durch Substitution der Variablen x durch einen beliebigen Grundterm der Sprache $\mathcal{L}$ entstehen, essentielle Sätze, so z. B. $p(a, a)$ oder $p(a, f(g(a, b)))$.

Im folgenden werden essentielle Sätze verwendet, um *Zustände* daraus aufzubauen:

Definition 2.1 (Zustandsbeschreibung (Zustand), Zustandsraum)

(i) Eine *Zustandsbeschreibung* (oder kurz *Zustand*) s ist eine endliche Teilmenge der essentiellen Sätze, also $s \subseteq \mathcal{E}^*$.
(ii) Der *Zustandsraum* S ist die Menge aller Zustände, d. h. $S = \wp_{FIN}(\mathcal{E}^*)$.[7]

Bemerkung 2.2 Inhaltlich werden die Atome, aus denen ein Zustand besteht, als konjunktiv verknüpft betrachtet.

Ein Zustand dient dazu, eine Welt zu beschreiben. Diese Beschreibung erfolgt dadurch, daß in einem Zustand die Eigenschaften in Form von essentiellen Sätzen notiert werden, die in einer Welt wahr sind. Eine Welt ist also ein endliches Modell eines Zustands im Sinne der Prädikatenlogik. Im Rahmen der formalen Semantik (siehe Abschnitt 2.3) werden wir daher davon sprechen, daß ein Zustand in einer Welt wahr ist.

[6] Hierbei sei a eine Konstante und x eine Variable.
[7] Hierbei bezeichnet $\wp_{FIN}(M)$ die Menge der endlichen Teilmengen von M.

Statischer Beschreibungsanteil

Die essentiellen Sätze lassen nur Grundatome zur Beschreibung von Welten zu. Nun sollen komplexere prädikatenlogische Formeln, die es erlauben, Zusammenhänge innerhalb einer Welt auszudrücken, eingeführt werden. Hierbei machen wir zwei Einschränkungen:

(i) Ausdrückbare komplexere Zusammenhänge gelten in allen betrachteten Welten.
(ii) Zusammenhänge werden in Form von Hornklauseln beschrieben.

Bemerkung 2.3 Die erste Einschränkung stellt hierbei nur eine syntaktische Einschränkung dar. Wie wir sehen werden, können komplexere Zusammenhänge, die nur in einigen Welten gelten, trotzdem dargestellt werden. Hierzu muß man lediglich der Prämisse der betreffenden Hornklausel ein zusätzliches 0-stelliges Prädikat, z. B. Q, hinzufügen, das als essentieller Satz deklariert werden muß. In Welten, in denen diese Hornklausel einen gültigen Zusammenhang ausdrückt, wird dies durch die Gültigkeit von Q repräsentiert; in den Welten in denen der Zusammenhang nicht gilt, ist Q nicht in der Zustandsbeschreibung enthalten.

Bemerkung 2.4 Die zweite Einschränkung, die Repräsentation in Hornklauselform betreffend, motiviert sich daraus, daß die Hornlogik ein relativ ausdrucksstarker Ausschnitt der Prädikatenlogik ist, der außerdem effizient verarbeitet werden kann.[8] Die im Vergleich zur Prädikatenlogik verminderte Ausdrucksstärke können wir hierbei in Kauf nehmen, da der Schwerpunkt des Interesses beim Problemlösen auf der Bestimmung von Weltübergängen liegt und nicht auf Aussagen innerhalb einer Welt.

Aufgrund dieser Einschränkungen werden die verwendbaren Formeln als *universelle Implikationen* bezeichnen.

Bezeichnung 2.2 (Universelle Implikation) Wir zeichnen eine Menge von Hornklauseln der prädikatenlogischen Sprache $\mathcal{L}$ aus, die wir als *universelle Implikationen* bezeichnen und mit $\mathcal{R}$ notieren.

Bemerkung 2.5 Gemäß dieser Bezeichnung, werden auch Grundatome in Form von Fakten als universelle Implikation (mit leerer Vorbedingung) zugelassen. Im Gegensatz zu den Grundatomen in Form von essentiellen Sätzen müssen die hier definierten Grundatome in allen betrachteten Welten gelten.

[8] Ein effizienter Inferenzmechanismus für die Hornlogik ist z. B. die SLD-Resolution (linear resolution with selection function for definite clauses) [Apt & Emden, 1982] die effiziente Implementierungen auf der Basis von WAM-Code (*W*arren *A*bstract *M*achine) [Warren, 1983] erlaubt. Diese sind die Grundlage der Programmiersprache PROLOG, die den heutigen Standard der *Logischen Programmierung* darstellt.

Schlußfolgerungen innerhalb von Welten

Aussagen über Eigenschaften, die in einer Welt gelten, sind bei dieser Repräsentation sowohl extensional als auch intensional gegeben. Ein Zustand ist dabei eine extensionale Beschreibung von Eigenschaften einer Welt; gültige Eigenschaften werden einfach aufgezählt und als Zustand notiert. Weitere Eigenschaften sind intensional dadurch bestimmt, daß ihre Gültigkeit in einer Welt durch Schlußfolgerungen mittels universeller Implikationen bestimmt werden kann. Solche Schlußfolgerungen innerhalb von Welten können mit Hilfe bekannter Kalküle wie z. B. dem Resolutionskalkül gezogen werden. Diese ermöglichen es festzustellen, ob eine Formel F eine Konsequenz aus den in einem Zustand s gegebenen essentiellen Sätzen zusammen mit den universellen Implikationen $\mathcal{R}$ ist. Dies notieren wir in der üblichen Form durch $s \cup \mathcal{R} \vdash F$. Da die universellen Implikationen in allen Welten gelten, sind alle intensional bestimmten Eigenschaften durch die Angabe eines Zustandes bereits vollständig determiniert. Welten, die unterschieden werden müssen (z. B. weil von ihnen aus verschiedene Aktionen möglich sind), müssen folglich durch unterschiedliche Zustände repräsentiert werden. Dies bedingt, daß die essentiellen Sätze hierfür das notwendige Vokabular zur Verfügung stellen müssen. Betrachten wir hierzu folgendes Beispiel:

Beispiel 2.2 In der Domäne der Arbeitsplanung für rotationssymmetrische Werkstücke (siehe Abschnitt 2.1.2) muß z. B. die aktuelle Form des Werkstückes sowie die Art der Aufspannung durch essentielle Sätze dargestellt werden, da verschieden geformte und verschieden aufgespannte Werkstücke unterschieden werden müssen. Die Repräsentation der Aufspannung kann beispielsweise durch essentielle Sätze der Form `chuck_pos(x)` erfolgen, wobei die Variable `x` entweder mit den Konstanten `none`, `left` oder `right` belegt werden kann. Die Interpretation, die wir hierbei zugrunde legen, ist die folgende: `chuck_pos(none)` bedeutet, daß das Werkstück nicht eingespannt ist; `chuck_pos(left)` und `chuck_pos(right)` bezeichnen die Einspannung des Werkstück auf der linken bzw. auf der rechten Seite. Ein möglicher Zustand kann nun z. B. durch $s_1 = \{$ `chuck_pos(left)` $\}$ gegeben sein. Dieser Zustand steht für jede Welt, in der ein nicht näher bestimmtes Werkstück auf der linken Seite aufgespannt ist. Welten, die sich in der Form des Werkstücks unterscheiden, können hierbei also nicht unterschieden werden.

Wir wollen nun zwei universelle Implikationen einführen, um eine weitere Eigenschaft, aufgrund eines in jeder Welt gültigen Zusammenhangs, implizit zu definieren.

Universelle Implikation:
`chucked` $\leftarrow$ `chuck_pos(left)`
`chucked` $\leftarrow$ `chuck_pos(right)`.

Hierbei soll das neue vorkommende Prädikat `chucked` so interpretiert werden, daß es genau dann zutrifft, wenn ein Werkstück eingespannt ist, egal von welcher Seite. Zu beachten ist hierbei, daß `chucked` kein essentieller Satz ist und folglich auch nicht in einem Zustand

vorkommen kann. Diese Implikationen können jedoch dazu genutzt werden, um die neue Eigenschaft abzuleiten. Sie ist eine gültige Eigenschaft für die betrachtete Welt.

2.2.3 Repräsentation von Aktionen

Aktionen werden durch *Operatoren* repräsentiert. Operatoren manipulieren Zustände, um Übergänge von einer Welt in eine andere darzustellen. Zur *Beschreibung* eines Operators, hat sich die in STRIPS [Fikes & Nilsson, 1971] eingeführte Repräsentationsweise bis heute durchgesetzt.

Definition 2.2 (Operatorbeschreibung, Operator) Eine *Operatorbeschreibung* ist ein 4-Tupel (o, Pre_o, Add_o, Del_o). Hier bei ist o ein 0-stelliges Funktionssymbol (Konstante), das *Operator* genannt wird, Pre_o ist eine endliche Konjunktion von Grundatomen aus $\mathcal{L}$, die *Vorbedingung* genannt wird, Add_o und Del_o sind endliche Mengen von essentiellen Sätzen aus $\mathcal{E}^*$, die *Addliste* bzw. *Deleteliste* genannt werden.

Bemerkung 2.6 Operatoren werden hierbei nicht als Konstanten im üblichen Sinne der Prädikatenlogik interpretiert.

Zur besseren Lesbarkeit von Operatorbeschreibungen möchten wir diese jedoch nicht als Tripel notieren, sondern verwenden hierzu die folgende frame-artige Darstellung, in der die Vorbedingung sowie die Add- und Deletelisten als separate Slots dargestellt sind.

$$
\begin{array}{l}
\mathit{Operator:} < \mathit{Operatorname} > \\
\quad \mathit{Pre:} \{< \mathit{Vorbed}_1 >, \ldots, < \mathit{Vorbed}_n >\} \\
\quad \mathit{Del:} \{< \mathit{Delete}_1 >, \ldots, < \mathit{Delete}_k >\} \\
\quad \mathit{Add:} \{< \mathit{Add}_1 >, \ldots, < \mathit{Add}_l >\}
\end{array}
$$

Schlußfolgerungen mit Operatoren

Ein Operator definiert eine Veränderung eines Zustandes gemäß folgender Definition.

Definition 2.3 (Zustandsübergang) Ein Operatorbeschreibung (o, Pre_o, Add_o, Del_o) definiert eine partielle Funktion $f_o : S \longrightarrow S$, *Zustandsübergang* genannt, durch:

$$
f_o(s) = \begin{cases} (s \setminus Del_o) \cup Add_o & \text{falls} \quad s \cup \mathcal{R} \vdash Pre_o \\ \text{undefiniert} & \text{sonst} \end{cases}
$$

Anstelle von $s_2 = f_o(s_1)$ schreiben wir auch $s_1 \stackrel{o}{\longrightarrow} s_2$.

Ist in einem Zustand die Vorbedingung Pre_o erfüllt, so ergibt sich ein Nachfolgezustand durch das Entfernen der essentiellen Sätze der Deleteliste und das Hinzufügen der essentiellen Sätze der Addliste. Dies entspricht der ursprünglichen Operatordefinition in STRIPS.

Diese Definition erlaubt es, auf einfache Weise Schlußfolgerungen über das Ergebnis von Operatoranwendungen auf Zustände zu ziehen. Hierbei muß lediglich die Operatorvorbedingung überprüft werden. Dies ist eine einfache Schlußfolgerung innerhalb der Welt, in der der Operator angewendet werden soll. Der Folgezustand kann dann durch effektive Berechnung der Funktion f_o bestimmt werden.

Beispiel 2.3 In Fortführung von Beispiel 2.2 kann ein einfacher Operator `unchuck` zur Repräsentation einer Aktion zum Ausspannen eines Werkstück wie folgt definiert werden:

Operator: `unchuck`
Pre: { `chucked` }
Del: { `chuck_pos(left)`, `chuck_pos(right)` }
Add: { `chuck_pos(none)` }

Dieser Operator ist genau dann anwendbar, falls `chucked` gilt, d. h. falls das Werkstück im aktuellen Zustand eingespannt ist. Diese Einschaft kann mit Hilfe der in Beispiel 2.2 eingeführten universellen Implikationen abgeleitet werden. Durch die Operatorausführung wird `chuck_pos(left)` bzw. `chuck_pos(right)` aus der Zustandsbeschreibung gelöscht und `chuck_pos(none)` hinzugefügt. Wir müssen in diesem Fall beide Möglichkeiten `chuck_pos(left)` und `chuck_pos(right)` in der Deleteliste aufnehmen. Das Prädikat `chucked` selbst können wir nicht löschen, da es nicht als essentieller Satz eingeführt ist. Die Konsequenz des Löschens von `chuck_pos(left)` und `chuck_pos(right)` ist natürlich auch, daß `chucked` nicht mehr abgeleitet werden kann.

Parametrisierte Operatoren

Im folgenden werden wir eine Erweiterung der Operatordefinition einführen, die es erlaubt Problemlösedomänen für praktisch relevante Problembereiche komfortabler zu definieren, als dies durch den bislang vorgestellten Rahmen möglich ist. Für viele Anwendungen sind *parametrisierte Operatoren* wünschenswert. Hierbei kann jedem Operator zusätzlich eine Liste von Parametern mitgegeben werden. Nachfolgend wird eine erweiterte Definition für Operatorbeschreibungen gegeben, die die vorherige Definition ersetzt.

Definition 2.4 (Parametrisiertere Operatorbeschreibung) Eine *parametrisierte Operatorbeschreibung* ist ein 4-Tupel $(o(x_1, \ldots, x_n), Pre_o, Add_o, Del_o)$. Hierbei ist o ein n-stelliges Funktionssymbol, das Operator genannt wird, und x_i sind Variablen, die *Parameter* des Operators genannt werden. Weiter ist Pre_o (Vorbedingung) eine endliche Konjunktion von Atomen aus $\mathcal{L}$, und Add_o (Addliste) und Del_o (Deleteliste) sind endliche

Mengen von Instanzen von essentiellen Satzformen. In Pre_o, Add_o und Del_o kommen hierbei nur die Variablen $x_1, \ldots, x_n$ vor.

Bezeichnung 2.3 (Instantiierter Operator) Sind $t_1, \ldots, t_n$ Grundterme und ist o ein Operator, so wird $o(t_1, \ldots, t_n)$ *instantiierter Operator* genannt.

Bemerkung 2.7 Operatoren werden hierbei nicht als Funktionen im üblichen Sinne der Prädikatenlogik interpretiert.

Schlußfolgerungen mit instantiierten Operatoren

Parametrisierte Operatoren werden nicht in Schlußfolgerungen benutzt, sondern lediglich instantiierte Operatoren. Diese können jedoch vollkommen analog zu gewöhnlichen Operatoren behandelt werden, was in der folgenden Definition zum Ausdruck kommt.

Definition 2.5 (Zustandsübergang eines instantiierten Operators)
Sei $(o(x_1, \ldots, x_n), Pre_o, Add_o, Del_o)$ eine parametrisierte Operatorbeschreibung, dann definiert der instantiierter Operator $o(t_1, \ldots, t_n)$ eine partielle Funktion $f_{o(t_1,\ldots,t_n)} : S \longrightarrow S$, durch:

$$f_{o(t_1,\ldots,t_n)}(s) = \begin{cases} (s \setminus (Del_o\sigma)) \cup (Add_o\sigma) & \text{falls} \quad s \cup \mathcal{R} \vdash (Pre_o\sigma) \\ \text{undefiniert} & \text{sonst} \end{cases}$$

Hierbei ergibt sich die Substitution σ durch $\sigma = \{x_1/t_1, \ldots, x_n/t_n\}$. Anstelle von $f_{o(t_1,\ldots,t_n)}(s_1) = s_2$ schreiben wir auch $s_1 \overset{o(x_1,\ldots,x_n)}{\longrightarrow} s_2$.

Beispiel 2.4 In Fortführung von Beispiel 2.3 kann ein parametrisierbarer Operator `chuck(x_pos)` eingeführt werden, der die Aufspannung eines Werkstückes an einer beliebigen Seite (rechts oder links) beschreibt. Hierbei gibt der Parameter `x_pos` die entsprechende Seite an und kann entweder mit der Konstanten `left` oder `right` belegt werden.[9] Die Beschreibung dieses Operators sieht dann wie folgt aus:

Operator: `chuck(x_pos)`
Pre: `chuck_pos(none)`
Del: { `chuck_pos(none)` }
Add: { `chuck_pos(x_pos)` }

2.2.4 Problemlösedomäne

Aufbauend auf den zuvor eingeführten Begriffen kann nun der Begriff der Problemlösedomäne formal einführt werden.

[9] Hierbei ist `x_pos` eine Variable und `left`, `right` und `none` sind Konstanten.

Definition 2.6 (Problemlösedomäne, Domäne) Eine Problemlösedomäne (Domäne) $\mathcal{D}$ wird durch das 4-Tupel $\mathcal{D} = (\mathcal{L}, \mathcal{E}, \mathcal{O}, \mathcal{R})$ beschrieben. Hierbei ist

(i) $\mathcal{L}$ eine prädikatenlogische Sprache erster Ordnung,
(ii) $\mathcal{E}$ eine endliche Menge von Atomen aus $\mathcal{L}$ (essentielle Satzformen),
(iii) $\mathcal{O}$ eine Menge von parametrisierten Operatorbeschreibungen, deren Operatoren voneinander verschieden sind und
(iv) $\mathcal{R}$ eine Menge Hornklauseln (universelle Implikationen).

Eine Problemlösedomäne ist also eine syntaktische Repräsentation eines Gegenstandsbereichs. Probleme des Gegenstandsbereichs müssen auf der Ebene dieser Repräsentation formuliert und durch entsprechende Schlußfolgerungen auf dieser Ebene gelöst werden. Daher definieren wir nachfolgend die Begriffe *Problem* und *Lösung*.

Definition 2.7 (Problem) Ein Problem p in einer Problemlösedomäne ist ein Paar von Zuständen $(s_I, s_G) \in S \times S$. Hierbei wird s_I als Initialzustand und s_G als Zielzustand bezeichnet.

Definition 2.8 (Lösung, Plan) Eine Lösung (Plan) ist eine endliche Folge von instantiierten Operatoren $\bar{o} = \langle o_1(t_{1,1}, \ldots, t_{1,k_1}), \ldots, o_n(t_{n,1}, \ldots, t_{n,k_n})\rangle$ der Problemlösedomäne.

Die wesentliche Schlußfolgerung, die wir in Problemlösedomänen ziehen, betrifft die Korrektheit einer Lösung für ein Problem. Mit Hilfe der durch Operatoren definierten Zustandsübergänge ergibt sich der Korrektheitsbegriff für Lösungen wie folgt:

Definition 2.9 (Korrektheit einer Lösung) Eine Lösung $\bar{o}$ ist eine *korrekte Lösung* eines Problems $p = (s_I, s_G)$, falls gilt:

$$s_I = s_o \overset{o_1(t_{1,1},\ldots,t_{1,k_1})}{\longrightarrow} s_1 \longrightarrow, \cdots, \longrightarrow s_{n-1} \overset{o_n(t_{n,1},\ldots,t_{n,k_n})}{\longrightarrow} s_n = s_G$$

2.3 Formale Semantik

Hier erörtern wir die Semantik der zuvor eingeführten Repräsentation und orientieren uns sowohl an der Kripkesemantik [Kripke, 1963] für die Modallogik, als auch an einer Semantik für STRIPS, die von Lifschitz [1987] eingeführt wurde. Die Kenntnis dieser beiden Arbeiten wird jedoch nicht vorausgesetzt, wohl aber Grundkenntnisse über die Semantik der Prädikatenlogik.

Wir beginnen mit der Definition der *Interpretation* für Problemlösedomänen.

Definition 2.10 (Interpretation) Eine Interpretation für eine Problemlösedomäne $\mathcal{D} = (\mathcal{L}, \mathcal{E}, \mathcal{O}, \mathcal{R})$ ist von der Form $I = (U, W, A, (IW_w)_{w \in W}, IA)$. Hierbei ist:

(i) U eine Menge, die *Universum* genannt wird,
(ii) W eine Menge, die *System von Welten* $w \in W$ genannt wird,
(iii) A eine endliche Menge von partiellen Funktionen $\alpha : W \longrightarrow W$, die *Aktionen* genannt werden,
(iv) IW_w für $w \in W$ eine gewöhnliche prädikatenlogische Interpretation für $\mathcal{L}$ über dem Universum U,
(v) IA eine totale Abbildung $IA : \mathcal{O} \longrightarrow A$ von Operatoren auf Aktionen.

Dieser Interpretationsbegriff erweitert die prädikatenlogische Interpretation um die Betrachtung verschiedener Welten und deren Übergänge. Aussagen, die sich auf eine spezielle Welt w beziehen, werden durch die dieser Welt zugehörige Interpretation IW_w Wahrheitswerte im üblichen Sinne zugeordnet. Operatoren werden durch die Abbildung IA als Aktionen interpretiert.

Wir können nun zunächst definieren, wann eine Formel *wahr* in einer Welt einer Interpretation ist.

Definition 2.11 (Gültigkeit einer Formel in einer Welt einer Interpretation) Eine *Formel* F aus $\mathcal{L}$ ist *wahr* in der Welt $w \in W$ einer Interpretation I wenn F wahr ist in der Interpretation IW_w im Sinne der Prädikatenlogik. Wir schreiben $I, w \models F$.

Diesen Wahrheitsbegriff wollen wir jetzt auf Zustände ausdehnen. Hierzu wird die Menge der essentiellen Sätze als konjunktiv verknüpft betrachtet.

Definition 2.12 (Gültigkeit eines Zustandes) Ein *Zustand* $s \in S$ ist *wahr* in der Welt $w \in W$ einer Interpretation I falls für alle $e \in s$ gilt: $I, w \models e$. Wir schreiben $I, w \models s$.

Es verbleibt noch die Definition des Wahrheitsbegriffes für eine Operatorbeschreibung:

Definition 2.13 (Gültigkeit einer Operatorbeschreibung) Eine *Operatorbeschreibung* (o, Pre_o, Add_o, Del_o) eines Operators o ist *wahr* in einer Interpretation I, wenn für jede Welt $w \in W$ mit $I, w \models Pre_o$ gilt:

(i) $IA(o)(w)$ ist definiert und
(ii) jeder essentielle Satz $e \in \mathcal{E}^* \setminus Del_o$, der in der Welt w der Interpretation I wahr ist, ist auch in der Welt $IA(o)(w)$ der Interpretation I wahr und
(iii) jeder Satz $e \in Add_o$ ist wahr in der Welt $IA(o)(w)$ der Interpretation I.

Hierbei wird also gefordert, daß immer, wenn ein Operator in einer Welt anwendbar ist, d. h. seine Vorbedingung in der Welt wahr ist, die Aktion, als die der Operator interpretiert wird, tatsächlich zu einer definierten neuen Welt führt (i). Außerdem müssen durch den Operator alle essentiellen Sätze, die in der Ausgangswelt wahr sind, nicht aber in der Welt, die durch die Aktion erreicht wird, gelöscht werden (ii). Schließlich darf ein Operator nur

essentiellen Sätze hinzufügen, die in der neuen Welt gelten (iii). Instantiierte Operatoren können ähnlich behandelt werden wie gewöhnliche Operatoren. Dies werden wir jedoch nicht näher erörtern.

Aufbauend auf den zuvor eingeführten Definitionen können wir jetzt definieren, wann eine Interpretation ein *Modell* für eine Problemlösedomäne ist.

Definition 2.14 (Modell einer Problemlösedomäne) Eine Interpretation I ist ein *Modell* für eine Problemlösedomäne $\mathcal{D} = (\mathcal{L}, \mathcal{E}, \mathcal{O}, \mathcal{R})$ wenn

(i) die Operatorbeschreibung jedes Operators aus $\mathcal{O}$ in I wahr ist und
(ii) jede universelle Implikation aus $\mathcal{R}$ wahr ist in jeder prädikatenlogischen Interpretation $(IW_w)_{w \in W}$.

Die Bedeutung von Modellen für Problemlösedomänen ergibt sich aus der Tatsache, daß es in diesen Modellen möglich ist, den Korrekheitsbegriff gemäß Definition 2.9 sinnvoll zu interpretieren. Dies wird durch den folgenden Satz ausgedrückt.

Satz 2.1 (Korrektheitssatz) Ist eine Interpretation I ein Modell der Problemlösedomäne $\mathcal{D}$ so gilt: ist der Plan $\bar{o} = \langle o_1, \ldots, o_n \rangle$ eine korrekte Lösung des Problems $p = (s_I, s_G)$ und gilt $I, w_I \models s_I$ für eine Welt w_I aus I, so ist $w_G = IA(o_n) \circ \cdots \circ IA(o_1)(w_I)$ definiert und es gilt $I, w_G \models s_G$.[10]

Diesen Satz wollen wir zunächst beweisen.

Beweis Wir führen die folgenden Abkürzungen für $i \in \{0, \ldots, n\}$ ein:

- s_i sei wie folgt gegeben durch : $s_I = s_o \xrightarrow{o_1} s_1 \longrightarrow, \cdots, \longrightarrow s_{n-1} \xrightarrow{o_n} s_n = s_G$,
- $\alpha_i = IA(o_i)$ und
- $\alpha^{(i,0)} = \alpha_i \circ \alpha_{i-1} \circ \cdots \circ \alpha_0$. Desweiteren ist $\alpha^{(0,0)}$ die Identität.

Wir zeigen nun durch Induktion nach i, daß für alle $i \in \{0, \ldots n\}$ gilt:

(a) $\alpha^{(i,0)}(w_0)$ ist definiert und
(b) $I, \alpha^{(i,0)}(w_0) \models s_i$.

Der Induktionsanfang ($i = 0$) ist offensichtlich.

Für $i > 0$ gilt: $\mathcal{R} \cup s_{i-1} \vdash Pre_{o_i}$ (wegen $s_{i-1} \xrightarrow{o_i} s_i$). Weiter gilt $I, \alpha^{(i-1,0)} \models \mathcal{R}$, da I Modell für D ist. Nach Induktionsvoraussetzung gilt außerdem $I, \alpha^{(i-1,0)}(w_0) \models s_{i-1}$. Da $\vdash$ ein korrekter Ableitungsoperator für die Hornlogik ist, folgt somit $I, \alpha^{(i-1,0)}(w_0) \models Pre_{o_i}$. Hieraus folgt aus Definition 2.13 (i) daß $\alpha_i(\alpha^{(i-1,0)}(w_0))$ definiert ist. Somit ist (a) bewiesen.

Nach Definition 2.13 (ii) gilt für alle $e \in \mathcal{E}^* \setminus Del_{o_i}$, daß aus $I, \alpha^{(i-1,0)}(w_0) \models e$ folgt, daß $I, \alpha_i(\alpha^{(i-1,0)}(w_0)) \models e$ gilt. Da nach Induktionsvoraussetzung $I, \alpha^{(i-1,0)}(w_0) \models s_{i-1}$ gilt, folgt direkt $I, \alpha(\alpha^{(i-1,0)}(w_0)) \models s_{i-1} \setminus Del_{o_i}$. Weiter folgt nach Teil (iii) von Definition

[10] Hierbei bezeichnet $\circ$ die Komposition von Funktionen. $f \circ g(x)$ steht für $f(g(x))$.

2.13, daß für alle $e \in Add_{o_i}$ gilt $I, \alpha_i(\alpha^{(i-1,0)}(w_0)) \models e$. Somit ist $I, \alpha(\alpha^{(i-1,0)}(w_0)) \models (s_{i-1} \setminus Del_{o_i}) \cup Add_{o_i}$ nachgewiesen. Nach Definition 2.2 ist $s_i = (s_{i-1} \setminus Del_{o_i}) \cup Add_{o_i}$, so daß (b) ebenfalls bewiesen ist.

Eine Interpretation setzt die Problemlösedomäne mit einer Gegenstandswelt in Beziehung. Ist diese Interpretation ein Modell der Problemlösedomäne, so besagt dieser Satz, daß eine korrekte Problemlösung im Gegenstandsbereich einer Aktionsfolge entspricht, die jede Welt, für die der Initialzustand wahr ist, in eine Welt überführt, in der der Zielzustand wahr ist.

Eine Problemlösedomäne muß immer zusammen mit einer Interpretation betrachtet werden, da diese den Zusammenhang zwischen den formal repräsentierten Problemen und Lösungen und dem Gegenstandsbereich herstellt. Die Wahl der essentiellen Sätze ist hierbei von entscheidender Bedeutung, da verschiedene Welten des Gegenstandsbereichs nur durch essentielle Sätze unterschieden werden können. Die essentiellen Sätze müssen also so gewählt werden, daß zwei Welten, die im Gegenstandsbereich unterschieden werden müssen, sich auch durch die Gültigkeit der essentiellen Sätze unterscheiden.

2.4 Beschränkungen dieses Problemlösebegriffes

Auch wenn die bislang eingeführte Sichtweise auf Problemlösen recht allgemein zu sein scheint, so ergeben sich dennoch ein Reihe von Einschränkungen, die auch im Rahmen dieser Arbeit nicht überwunden werden, da es nicht das Ziel dieser Arbeit ist, ein neues, verbessertes Modell von Problemlösen bzw. Planen zu entwickeln. Diese generellen Einschränkungen sollen an dieser Stelle genannt werden, da sie natürlich auch Einschränkungen für die Anwendbarkeit der in dieser Arbeit im Vordergrund stehenden Wiederverwendungsmethoden darstellen.

Qualifikationsproblem: Als Qualifikationsproblem [McCarthy, 1980; Shoham, 1986] bezeichnet man die Notwendigkeit, Vorbedingungen für die Anwendbarkeit von Aktionen vollständig zu kennen und als Vorbedingungen von Operatoren zu repräsentieren. In den bislang vorgestellten Beispielen haben wir nur die wichtigsten Vorbedingungen der jeweiligen Operatoren aufgeführt. Einige Vorbedingungen haben wir dabei außer acht gelassen, z. B. daß während des Drehprozeßes die Stromversorgung der Drehmaschine nicht ausfällt, das verwendete Schneidwerkzeug nicht stumpf ist und gerade kein Erdbeben stattfindet. Das Qualifikationsproblem ist eine spezielle Ausprägung des allgemeinen Problems der Modellierung eines realen Weltausschnittes durch eine formale Repräsentation in einem Computer. Es ist prinzipiell unmöglich, alle Details, die in der realen Welt vorkommen, zu kennen und geeignet abzubilden. Die Aufgabe eines Wissensingenieurs ist es daher, eine geeignete Modellierung auszuwählen, in der die wesentlichen Aspekte einer Domäne abgebildet werden.

Unvollständig bekannte Welt: Ein ähnliches Problem ist dann gegeben, wenn die Welt nicht von Anfang an vollständig bekannt ist. In der beschriebenen Sichtweise des Problemlösens, wird jedoch davon ausgegangen, daß vollständige Information über die Objekte der Welt (kodiert in den Zustandsbeschreibungen) und über die verfügbaren Operatoren vorliegt. Unvollständiges Weltwissen kann in dem vorliegenden Ansatz prinzipiell nicht behandelt werden [Schoppers, 1987; Georgeff & Lansky, 1987; Etzioni *et al.*, 1992]. Dies kann dazu führen, daß z. B. ein neues Problem überhaupt nicht im Rahmen des Problemlösemodells darstellbar ist, weil z. B. die erforderlichen Objekte (essentielle Sätze der Zustandsbeschreibung) nicht definiert wurden. Es kann auch passieren, daß zwar das Problem darstellbar ist, jedoch eine Lösung, die in der realen Welt existiert, nicht durch das System dargestellt und gefunden werden kann, da die erforderlichen Operatoren nicht vorhanden sind. In der von uns betrachteten Domäne der Arbeitsplanung für rotationssymmetrische Drehteile kann das z. B. dann auftreten, wenn die Herstellung eines Werkstückes Werkzeuge erfordert, die in der vorliegenden Problemlösedomäne nicht enthalten sind.

Unsicherer Ausgang von Operationen: Ein weiteres Problem liegt vor, wenn Operatoren in einer Domäne ein nicht-deterministisches Verhalten aufweisen. So kann es z. B. sein, daß der Ausgang einer Operation nicht mit absoluter Sicherheit vorhergesehen werden kann [Kushmerick *et al.*, 1994]. Ein typisches Beispiel hierfür ist eine Würfeloperation, bei der es 6 mögliche Ausgänge mit einer Wahrscheinlichkeit von jeweils 1/6 gibt. Solche Operationen können in unserem Problemlösemodell nicht repräsentiert werden.

Dynamische Welt: Unserem Modell von Problemlösen liegt die Annahme zugrunde, daß sich die Welt nur durch die Ausführung von Aktionen verändert. Bereiche, in denen sich die Welt selbständig verändert, ohne daß Aktionen ausgeführt werden, können nicht behandelt werden [Georgeff & Lansky, 1987; Schoppers, 1987; Penberthy & Weld, 1994].

Kontinuierliche Veränderungen: Aktionen beschreiben diskrete Weltveränderungen. Kontinuierliche Veränderungen, wie z. B. das langsame Trocknen der Farbe nach dem Lackieren eines Werkstücks, sind nicht repräsentierbar. Es fehlt auch eine explizite Repräsentation von Zeit.

Nebenläufige Aktionen: In einigen Gegenstandsbereichen kann es erforderlich sein, bestimmte Aktionen parallel durchzuführen, um z. B. eine erhöhte Ausführungsgeschwindigkeit zu erzielen oder, weil z. B. die parallele Ausführung bestimmter Aktionen zu anderen Resultaten führt als die serielle Ausführung. Unser Problemlösemodell ist jedoch auf die sequentielle Ausführung von Aktionen beschränkt.

Komplexere Kontrollstrukturen: In vielen Bereichen, so z. B. beim Programmieren in einer prozeduralen Programmiersprache, sind weitere Kontrollstrukturen (z. B. Verzwei-

gungen oder Schleifen) neben der rein sequentiellen Ausführung von Aktionen die Regel. Solche Kontrollstrukturen können in unserem Problemlösemodell nicht dargestellt werden [Chapman, 1987; Weld, 1994].

Wenn auch die oben beschriebenen Einschränkungen bereits seit langem bekannt sind und in der Literatur diskutiert werden [Georgeff, 1987; Herzberg, 1989; Weld, 1994], ist bis heute keine allgemein akzeptierte und verbreitete alternative Sicht auf Problemlösen entstanden. Vereinzelt sind erste Ansätze entwickelt worden, auf die wir jedoch hier nicht weiter eingehen können.

Allgemeine Annahmen

Zur Vermeidung der zuvor beschriebenen Probleme müssen wir die folgenden allgemeinen Annahmen machen.

- Die im Gegenstandsbereich zu betrachtenden Welten und Aktionen sind vollständig bekannt.
- Der Gegenstandsbereich selbst verändert sich nicht, d. h. die vorkommenden Welten und Aktionen sowie ihre Definitionen bleiben konstant.
- Die Welt wird nur durch die Ausführung von Aktionen verändert.
- Die Veränderungen der Welt sind diskreter Natur.
- Aktionen verändern die Welt deterministisch.
- Aktionen werden nur sequentiell ausgeführt.

2.5 Der Problemlöseprozeß

Wie bereits durch Chapman [1987] nachgewiesen wurde, ist die Frage, ob ein Problem eine Lösung besitzt, bereits bei wesentlich einfacheren Repräsentationen von Problemlösedomänen unentscheidbar! Daher kann es keinen Algorithmus geben, der bei Eingabe einer Problemlösedomäne und eines Problems nach der hier betrachteten Definition entscheidet, ob das Problem lösbar ist.

Im Teilgebiet der KI, das sich mit Aktionsplanung beschäftigt, sind eine Reihe von verschiedenen Algorithmen und Systemen entwickelt worden, die versuchen, Lösungen von Planungsproblemen zu bestimmen. Diese können dann als „Black Box“ verwendet werden, und ein Wissensingenieur muß zur Realisierung einer Applikation lediglich die erforderliche Problemlösedomäne formulieren. Folglich hat ein solcher universeller Problemlösemechanismus eine besonders hohe Attraktivität.

Jedoch ist inzwischen nachgewiesen, daß es Domänen gibt, bei denen Problemlösen NP-vollständig oder sogar PSPACE-vollständig ist [Bylander, 1991; Bylander, 1992]. Folglich kann es nach dem aktuellen Kenntnisstand der Komplexitätstheorie keinen universellen Problemlösealgorithmus mit nur polynomialem Zeitbedarf geben. Für spezielle Domänen, die weniger harte Probleme beschreiben, ist es natürlich durchaus möglich, effiziente Algorithmen zu finden, was jedoch in der Regel eine aufwendige Analyse der Domäne und die Entwicklung eines auf diese Domäne zugeschnittenen Algorithmus erfordert.

Die bislang in den Forschungsgebieten der Planung und der Suche entwickelten domänenunabhängigen Problemlöser nutzen typischerweise als einzige Information die Domänenbeschreibung in Form von Zuständen und Operatoren aus, da diese minimal und hinreichend ist, um eine Domäne vollständig zu spezifizieren. Man sagt auch, Probleme werden durch Rückgriff auf *erste Prinzipien* gelöst. Hierbei ist zwischen der Suche im Zustandsraum und der Suche im Lösungsraum zu unterscheiden. Dieser Unterschied soll an dieser Stelle kurz erläutert werden.

2.5.1 Suche im Zustandsraum

Der einfachste und chronologisch erste Ansatz zum Problemlösen ist die Suche im Zustandsraum. Hierbei wird im Zustandsgraph nach einem Pfad vom Ausgangszustand zum Zielzustand gesucht. Diese Suche kann vom Anfangszustand ausgehend arts oder vom Zielzustand ausgehend rückwärtsgerichtet durchgeführt werden. Beim Durchsuchen des Suchraumes muß ein Verfahren zu einem bestimmten Zeitpunkt entscheiden, welcher Zustand als nächstes betrachtet werden soll. Hierfür bieten sich verschiedene *Suchstrategienen* an, z. B. die Breitensuche, Tiefensuche oder Bestensuche [Dijkstra, 1959; Korf, 1987; Korf, 1988; Korf, 1993]. Diese Suchverfahren können hierbei nicht von einer expliziten Repräsentation des Zustandsgraphen ausgehen, sondern müssen die erforderlichen Zustände, bzw. ihre Beschreibung in jedem Schritt *berechnen*. Bei einer vorwärtsgerichteten Suche ist das besonders einfach, da sich ein Nachfolgezustand direkt aus der Vorschrift für eine Operatoranwendung ergibt. Bei der rückwärtsgerichteten Suche ist es jedoch erforderlich, die schwächste Vorbedingung zu bestimmen, die hinreichend ist, um die Anwendbarkeit eines Operators zu garantieren. Dieser Schritt wird *Regression* genannt [Waldinger, 1977]. Da alle diese Suchverfahren den durch die Probemlösedomäne definierten Suchraum vollständig durchsuchen, sind solche Algorithmen *vollständig* in dem Sinne, daß sie eine Lösung eines Problems finden, falls eine solche existiert und dem Suchalgorithmus genügend Rechenzeit zur Verfügung steht. Ebenso sind solche Algorithmen natürlich auch korrekt, d. h. sie liefern nur Ausgaben, die tatsächlich Lösungen des Problems sind.

An dieser Stelle soll auch nochmal eines der ersten Verfahren zur Suche im Zustandsraum – die Mittel-Zweck-Analyse (engl. means end analysis) – genannt werden. Die Mittel-Zweck-Analyse wurde sowohl in GPS [Newell & Simon, 1962; Ernst & Newell, 1969] als auch in STRIPS [Fikes & Nilsson, 1971] eingesetzt. Hierbei wird eine spezielle Suchstrategie verwendet, bei der die Differenzen der Zustandsbeschreibung von Anfangs- und Zielzustand als

Ziele angesehen, die *der Reihe nach* durch eine rückwärtsgerichtete Tiefensuche bearbeitet werden. Dieser Vorgehensweise liegt jedoch die *Linearitätsannahme* [Sussman, 1975] zugrunde, die besagt, daß alle Ziele *unabhängig* von anderen Zielen durch nicht-interagierende Teilpläne gelöst werden können. Diese Annahme führt jedoch oft zu Problemen. Ein klassisches Beispiel hierfür taucht bereits in der Blockswelt auf und ist unter dem Begriff *Sussmann-Anomalie* [Sussman, 1975] bekannt geworden. Die Verletzung der Linearitätsannahme führt dazu, daß die Mittel-Zweck-Analyse nicht mehr vollständig ist, d. h. zu lösbaren Problemen keine Lösung findet. Diese Einschränkung ist spezifisch für lineare Problemlösemethoden, wie die Mittel-Zweck-Analyse, jedoch keineswegs zutreffend für das gesamte Spektrum der Problemlöser, die im Zustandsraum suchen. Bei der vorwärtsgerichteten Suche werden Zielinteraktionen bereits in der Vorschrift zur Operatoranwendung (explizites Löschen ungültiger Sätze) berücksichtig und bei der rückwärsgerichteten Suche werden diese im Regressionsschritt behandelt.

2.5.2 Suche im Lösungsraums

Ein alternativer Ansatz zum Problemlösen wird seit Mitte der 70'er Jahre verfolgt und ist unter dem Begriff *nicht-lineares Planen* eingeführt worden [Sacerdoti, 1975; Sacerdoti, 1977; Tate, 1977]. Hierunter verbirgt sich eine Klasse von Problemlösern, die, im Gegensatz zur Suche im Zustandsraum, im *Lösungsraum* suchen [Weld, 1994]. In diesem Raum repräsentieren die Knoten teilweise spezifizierte Lösungen und die Kanten stellen Verfeinerungsschritte für Lösungen dar. Der Initialzustand dieses Suchraumes besteht aus der leeren Lösung und der Zielzustand ist eine vollständige Lösung für das zu lösende Problem. Der Vorteil dieses Lösungsansatzes besteht zunächst darin, daß verschiedene Verfeinerungsschritte für Lösungen betrachtet werden können, wie z. B. das Einfügen eines Operators an einer beliebigen Stelle im Plan. Wesentlich zentraler jedoch ist die Möglichkeit, auf eine totale Ordnung der Operatoren während des Problemlösens ganz zu verzichten und Lösungen mit partiell geordneten Operatoren zu betrachten (engl. partial-order planning).

Auch diese Art der Suche im Lösungsraum hat zu einer Reihe von vollständigen und korrekten Problemlösern geführt. Hierbei sind z. B. NONLIN [Tate, 1977], TWEAK [Chapman, 1987], SNLP [McAllester & Rosenblitt, 1991], UCPOP [Weld, 1994] zu nennen.

2.5.3 Suche im Zustandsraum vs. Suche im Lösungsraum

Ursprünglich vermutete man, daß Problemlösen durch Suche im Lösungsraum grundsätzlich leistungsfähiger ist als Problemlösen durch Suche im Zustandsraum [Minton *et al.*, 1991; Kamphampati & Chen, 1993; Barrett & Weld, 1994]. Jedoch konnte inzwischen gezeigt werden, daß die Suche im Zustandsraum und die Suche im Lösungsraum verschiedene Heuristiken repräsentieren, die in Abhängigkeit der gerade vorliegenden *Domäne* mehr oder weniger geeignet sein können [Veloso & Blythe, 1994; Knoblock & Yang, 1994; Minton *et al.*, 1994; Kambhampati & Srivastava, 1995]. Dadurch wurde die zuvor vermutete allgemeine

Überlegenheit der Suche im Lösungsraum widerlegt und ist der Erkenntnis gewichen, daß man im Prinzip versuchen muß, für jede Domäne – oder eigentlich für jedes Problem in einer Domäne – den am besten geeigneten Suchmechanismus auszuwählen.

2.5.4 Komplexität des Problemlösens

Problemlösen gehört zu den komplexitätsmäßig schwierigsten Problemen in der Informatik. Die Frage nach der Existenz einer Lösung eines Problems, bei gegebener Problemlösedomäne, ist für die meisten Problemlösedomänen NP-vollständig, ja sogar PSPACE-vollständig [Bylander, 1991; Bylander, 1992].

Die vorgestellten Ansätze zum Problemlösen durch vollständige Suche haben alle eine exponentielle Zeitkomplexität. Für die Suche im Zustandsraum kann diese Komplexität mit $O(b^n)$ angegeben werden, wobei b der maximale Verzweigungsgrad (engl. branching factor) ist und n die Länge der Lösung [Korf, 1987]. Je nachdem ob mit vorwärtsgerichteter oder rückwärtsgerichteter Suche vorgegangen wird, gibt b die Anzahl der Nachfolger eines Zustands oder die Anzahl der eines Zustandes an. Für die Suche im Lösungsraum ist für eine Reihe von Verfahren gezeigt worden, daß sich die Zeitkomplexität zu $O(c \cdot b^d)$ ergibt, wobei c die Komplexität zur Expansion eines Knotens im Lösungsraum ist, b der effektive Verzweigungsgrad der durch den Suchalgorithmus bedingt ist und d die effektive Suchtiefe [Knoblock & Yang, 1994]. Die genaue Bestimmung der Parameter c, b und d ist dabei vom speziellen Verfahren abhängig. Wichtig ist aber, daß die Suchtiefe d in jedem Fall ein Vielfaches der Anzahl n der in der Lösung vorkommenden Operatoren ist. Folglich kann die Komplexität jedes bekannten Algorithmus zum Problemlösen durch $O(c \cdot b^{nk})$ angegeben werden, wobei c, b und k algorithmen- und domänenspezifische Parameter sind und n die Länge der Lösung ist. Die Komplexität hängt also ganz entscheidend von der Länge der gesuchten Lösung ab.

Für den praktischen Einsatz eines Problemlösers ist der exponentielle Zeitbedarf ein schwerwiegendes Problem. Wenn ein Problemlöser in der Lage ist, einfache Probleme zu lösen, ist das in der Regel für eine Anwendung wenig nützlich, wenn nicht auch komplexere Probleme, die auch für menschliche Experten schwierig sind, gelöst werden können.

Hierbei ist zu beachten, daß generische Problemlöser, die nur aufgund erster Prinzipien arbeiten, in der Regel auch dann einen exponentiellen Zeitbedarf haben, wenn Problemlösen in der vorliegenden Problemlösedomäne selbst nicht NP-vollständig ist. Der Grund dafür ist die spezielle Kodierung des vorliegenden Problembereiches durch Zustände und Operatoren. Um hier zu effizienteren Problemlösern zu kommen, ist es erforderlich, weiteres Wissen während des Problemlöseprozesses zu benutzen. Zusätzliches Wissen muß dem Problemlöser zur Verfügung gestellt und von diesem verarbeitet werden können, damit zumindest teilweise auf Suche verzichtet werden kann. Solches *Kontrollwissen* ist Wissen, das einen Suchprozeß im Zustands- oder Lösungsraum so steuert, daß Teile des Suchraumes, die von vornherein als aussichtslos oder ungünstig erkannt werden, abgeschnitten werden.

Dieses Wissen zusätzlich zur Modellierung der Problemlösedomäne zu akquirieren und zu formalisieren ist jedoch ein erheblicher Aufwand, der von einem Wissensingenieur zu leisten ist, und der die Attraktivität eines domänenunabhängigen Problemlösers erheblich reduziert. Daher wird seit etwa Mitte der 80'er Jahre versucht, weiteres Wissen, das zur Effizienzsteigerung eines Problemlösers eingesetzt werden kann, automatisch mit Hilfe von Methoden des *Maschinellen Lernens* aus Beispielen, auch Fälle genannt, zu akquirieren.

Kapitel 3

Methoden zur Leistungssteigerung von Problemlösern

In diesem Kapitel werden einige grundlegende Methoden vorgestellt, die es erlauben, Problemlösen leistungsfähiger zu gestalten. Zunächst wird das *fallbasierte Schließen* (Abschnitt 3.1) und das *erklärungsbasierte Lernen* (Abschnitt 3.2) eingeführt. Hierbei handelt es sich um zwei Methoden, die es erlauben, Fallwissen auszunutzen, um den Suchaufwand beim Problemlösen zu reduzieren. Das erklärungsbasierte Lernen verfolgt hierzu einen „compilierenden" Ansatz, d. h. die Fälle werden zunächst in eine effizienter zu verarbeitende Datenstruktur transformiert, die dann während des Problemlösens eingesetzt wird. Das fallbasierte Schließen realisiert hingegen einen „interpretierenden" Ansatz und benutzt die vorliegenden Fälle ohne wesentliche Vorverarbeitung direkt zur Problemlösung. Der dritte Teil dieses Kapitels (siehe Abschnitt 3.3) erörtert die Verwendung von *Abstraktionsmechanismen* beim Problemlösen und analysiert die sich daraus ergebenden Leistungsvorteile.

Ziel dieses Kapitels ist es nicht, einen vollständigen Überblick über diese Forschungsrichtungen zu geben. Solche Überblicksarbeiten sind bereits in der einschlägigen Fachliteratur zu finden (zum erklärungsbasierten Lernen siehe [Ellman, 1989; DeJong, 1993], zum fallbasierten Schließen siehe [Riesbeck & Schank, 1989; Kolodner, 1993; Aamodt & Plaza, 1994; KI - 1/96, 1996], zur Abstraktion siehe [Giunchiglia & Walsh, 1992]). Hier soll vielmehr kurz in die genannten Gebiete eingeführt und die relevanten Begriffe und Mechanismen als Grundlage für die weitere Arbeit erörtert werden.

3.1 Fallbasiertes Schließen

Im folgenden werden wir in das Gebiet des fallbasierten Schließens einführen. Dabei werden wir uns im wesentlichen an der in [Wess, 1995] und [Aamodt & Plaza, 1994] eingeführten Terminologie orientieren. Da die Formalisierung von fallbasiertem Schließen stark von dem betroffenen Aufgabentyp (z. B. Klassifikation, Konfiguration, Design, und Aktionsplanung)

abhängt, werden die nachfolgenden Betrachtungen auf die informelle Ebene beschränkt. Die angegebenen Verweise ermöglichen jedoch bei Bedarf eine weitere Vertiefung.

3.1.1 Die Grundidee des fallbasierten Schließens

Der Begriff *fallbasiertes Schließen* (engl. Case-Based Reasoning, CBR) geht ursprünglich zurück auf die kognitionswissenschaftlichen Arbeiten von Roger Schank [Schank, 1982] und Janet Kolodner [Kolodner, 1983a; Kolodner, 1983b]. Fallbasiertes Schließen bedeutet die Lösung eines neuen Problems durch Erinnerung an eine vergangene Problemsituation und die Wiederverwendung der dabei gemachten Erfahrung im Kontext des neuen Problems [Wess, 1995]. Problemsituationen und dabei gemachte Erfahrungen stehen hierbei also im Mittelpunkt des Problemlösens. Diese werden als *Fälle* bezeichnet.

Begriffsbestimmung 3.1 (Fall) Ein Fall ist die Beschreibung einer Problemsituation zusammen mit den Erfahrungen, die während der Bearbeitung des Problems gewonnen werden konnten.

Bemerkung 3.1 Häufig wird ein Fall nur als Paar (Problem, Lösung) angesehen. Wir fassen diesen Begriff hier bewußt etwas weiter (vgl. auch Definition 3.1 und Begriffsbestimmung 3.8).

Damit Erfahrungen zur Lösung neuer Probleme wiederverwendet werden können, müssen sie gespeichert werden. Hierbei kommt es darauf an, daß die Fälle so gespeichert werden, daß die für die Lösung neuer Probleme relevanten Fälle in angemessener Zeit wiedergefunden werden. Hierzu bedarf es einer geeigneten Organisation der Fälle in einer *Fallbasis*.

Begriffsbestimmung 3.2 (Fallbasis) Eine Fallbasis ist eine geeignet organsierte Sammlung von Fällen.

Zusammenfassend läßt sich der Begriff fallbasiertes Schließen wie folgt beschreiben:

Begriffsbestimmung 3.3 (Fallbasiertes Schließen) Fallbasiertes Schließen ist ein Ansatz zum Problemlösen durch Auswahl von Fällen und Wiederverwendung der enthaltenen Erfahrungen.

Beim Problemlösen durch fallbasiertes Schließen lassen sich mehrere Teilprozesse identifizieren, die üblicherweise vorkommen. Im Laufe der Zeit sind in der Literatur eine Reihe von unterschiedlichen Modellen vorgeschlagen worden [Kolodner *et al.*, 1985; Hammond, 1989; Riesbeck & Schank, 1989; Althoff & Wess, 1992; Aamodt & Plaza, 1994], in denen verschiedene Phasen unterschieden werden. Wir wollen im folgenden das Prozeßmodell

von Aamodt und Plaza [1994] kurz rekapitulieren, da auf dieses Modell inzwischen im europäischen Raum überwiegend Bezug genommen wird. In diesem Modell werden die vier Phasen *Retrieve*, *Reuse*, *Revise* und *Retain* unterschieden. In der Retrieve-Phase wird für ein neues Problem ein Fall (oder ggf. mehrere Fälle) aus der Fallbasis ausgewählt. Während der Reuse-Phase werden die in diesem Fall (oder in den Fällen) enthaltenen Erfahrungen im Kontext der neuen Problemstellung wiederverwendet, und eine Lösung für das neue Problem wird bestimmt. Die gefundene Lösung wird in der Revise-Phase überprüft und ggf. revidiert. In der Retain-Phase werden neu gewonnene Problemlöseerfahrungen gespeichert. Abbildung 3.1 zeigt, wie diese Phasen zu einem vollständigen Zyklus angeordnet sind. Nachfolgend wollen wir diese Phasen etwas genauer betrachten, ohne technische Realisierungen im Detail zu erörtern.

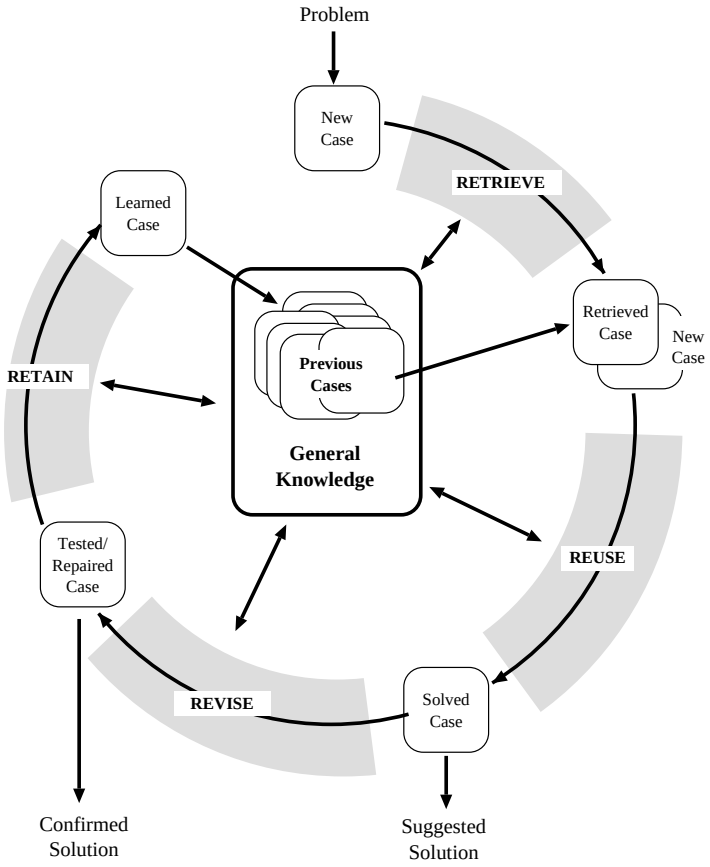

Abbildung 3.1: Der Zyklus des fallbasierten Schließens nach Aamodt und Plaza, 1994.

3.1.2 Fallbasis und verwendete Repräsentationsformen

Im Mittelpunkt dieses Modells steht die Fallbasis. Wie bereits erwähnt, ist die Fallbasis eine Sammlung von Fällen. Ein Fall enthält dabei eine Beschreibung eines Problems zusammen mit der Beschreibung der zugehörigen Lösung (Erfahrung). In einem fallbasierten System können solche Fälle auf verschiedene Weisen repräsentiert werden. Am einfachsten ist die Repräsentation durch Attribut-Wert-Paare wie sie z. B. im PATDEX-System [Stadler & Wess, 1989; Wess, 1991] verwendet wurde.[1] Ein Teil der Attribute wird zur Repräsentation der Problemsituation und der verbleibene Teil zur Repräsentation der Lösung verwendet. Attribut-Wert-Repräsentationen sind jedoch nur für einfach strukturierte analytische Problemebereiche geeignet. In komplexeren Domänen finden zwischenzeitlich objektorientierte Fallrepräsentationen Anwendung. CASUEL [Manago *et al.*, 1994] ist eine objektorientierte Repräsentationssprache für Fälle, die im fallbasierten System INRECA [Manago *et al.*, 1993; Althoff *et al.*, 1995] Verwendung findet. In fallbasierten Systemen für synthetische Aufgaben wie, z. B. Aktionsplanung, werden Fälle oft durch strukturierte Mengen von pradikatenlogischen Formeln repräsentiert. Auch in dieser Arbeit werden Fälle auf diese Weise repräsentiert. Wir werden hierauf genauer in den Abschnitten 3.1.7, 4.1.2, 5.1 und 6.1 eingehen.

3.1.3 Retrieve

Der Retrieve-Schritt ist einer der wichtigsten Phasen im Prozeßmodell des fallbasierten Schließens. Kein fallbasiertes System kommt ohne diesen Schritt aus.

Begriffsbestimmung 3.4 (Retrieve-Phase) In der Retrieve Phase wird ein Fall oder eine Menge von Fällen aus der Fallbasis ausgewählt, die geeignet sind, um ein neu vorliegendes Problem zu lösen.

Hierbei stellen sich zwei wesentliche Fragen:

- Wann ist ein Fall geeignet zur Lösung des neuen Problems?
- Wie können geeignete Fälle effizient aus der Fallbasis ausgewählt werden?

Die Eignung eines Falls wird üblicherweise aufgrund von *Ähnlichkeitsbeurteilungen* bestimmt. Hierbei wird die Ähnlichkeit zwischen dem neu vorliegenden Problem und der Problemsituation des Falls betrachtet. Dies basiert auf der grundlegenden Annahme, daß ähnliche Probleme auch ähnliche Lösungen besitzen. Zur Bestimmung der Ähnlichkeit zweier Probleme ist eine Vielzahl verschiedener Ansätze entwickelt worden. Weit verbreitet ist

[1] Attribut-Wert-Repräsentationen haben auch den Vorteil, daß sie theoretische Analysen von fallbasiertem Schließen (siehe z. B. [Globig & Wess, 1993]) vereinfachen, ohne zu stark von praktischen Problemen zu abstrahieren.

die Modellierung von Ähnlichkeit durch *Ähnlichkeitsmaße*. Hierbei liegt oft eine geometrische Interpretation zugrunde, bei der Fälle in einem n-dimensionalen Raum betrachtet werden. Jede Dimension entspricht dabei einem Attribut aus der Problembeschreibung. Die Ähnlichkeit zwischen zwei Problemen (zwei Punkte im n-dimensionalen Raum) kann auf den geometrischen Abstand zurückgeführt werden. Wir werden hierauf nicht genauer eingehen, da dieses Art der Ähnlichkeitsbestimmung überwiegend für analytische Bereiche geeignet ist. Eine ausführliche Darstellung dieser Ansätze findet sich in [Wess, 1995].

In synthetischen Bereichen werden überwiegend wissensbasierte Methoden zur Ähnlichkeitsbestimmung angewendet. Diese benutzen meist deklarativ repräsentiertes Hintergrundwissen zur Berechnung von Ähnlichkeiten zwischen Problemen. So kann die Ähnlichkeitsbestimmung auch als Erklärungsprozeß aufgefaßt werden. Ein solcher Erklärungsprozeß kann z. B. die Bedeutung von Unterschieden oder Gemeinsamkeiten von Problemstellungen wissensbasiert ermitteln [Aamodt, 1991; Aamodt, 1994]. Erklärungen können aber auch dafür eingesetzt werden, um bei einem vorliegenden Fall herauszufinden, welche Merkmale der Problembeschreibung *relevant* sind für die im Fall enthaltene Lösung [Kambhampati, 1990b; Janetzko *et al.*, 1992; Veloso, 1992a; Veloso, 1994a; Bergmann *et al.*, 1994a; Pews, 1994]. Auch in dieser Arbeit werden erklärungsbasierte Verfahren (siehe Abschnitt 3.2 und Kapitel 6) im Rahmen der Fallauswahl eingesetzt.

Eine weiterer, für diese Arbeit wichtiger Ansatz besteht darin, Ähnlichkeit aufgrund von Abstraktion zu definieren. Hierbei wird Ähnlichkeit als Gleichheit unter einer Abstraktionsabbildung verstanden. In Abhängigkeit vom Abstraktionsgrad können so verschieden hohe Ähnlichkeiten definiert werden. Ansätze hierzu finden sich in [Bergmann & Dannenmann, 1992; Horz, 1992; Bergmann *et al.*, 1994a; Pews, 1994; Wilke, 1994]. Der in dieser Arbeit entwickelte Ansatz zur Fallauswahl auf der Grundlage von Abstraktionsmechanismen wird ausführlich in Kapitel 7 dargestellt und in Abschnitt 9.1.2 diskutiert.

Nach diesem Überblick über Ansätze zur Ähnlichkeitsbeurteilung wollen wir noch kurz auf Ansätze verweisen, die es erlauben, ähnliche Fälle *effizient* aus einer Fallbasis auszuwählen. Dies ist besonders dann wichtig, wenn die Fallbasis sehr groß ist. Das Spektrum der bislang entwickelten Retrievalverfahren ist sehr groß. Zunächst ist anzumerken, daß Retrievalverfahren sehr stark von der Art der Ähnlichkeitsbeurteilung abhängen. Einzig das sequenzielle Retrieval, bei dem die Ähnlichkeit zwischen dem vorgelegten Problem und einem Fall aus der Fallbasis der Reihe nach für jeden Fall der Fallbasis berechnet wird, ist weitgehend unabhängig von der Art der Ähnlichkeitsbeurteilung. Dieser Ansatz ist jedoch aus Effizienzgründen für große Fallbasen nicht akzeptabel. Effiziente Retrievalverfahren basieren auf einer geeigneten Strukturierung der Fallbasis z. B. durch Bäume, Hashtabellen, Diskriminationsnetze. In [Wess, 1995] werden verschiedene Retrievalverfahren, wie sie für analytische Problembereiche eingesetzt werden (u. A. Retrieval mit kd-Bäumen) diskutiert und gegenübergestellt. Wie bereits erwähnt, stellt Kapitel 7 dieser Arbeit einen auf Bäumen basierenden Retrievalansatz ausführlich vor. Eine empirische Beurteilung der Effizienz dieses Verfahrens ist in Abschnitt 8.4.9 zu finden.

3.1.4 Reuse

Der Reuse-Schritt verwendet die im Retrieve-Schritt ausgewählten Fälle zur Erstellung einer Lösung des vorgelegten Problems.

Begriffsbestimmung 3.5 (Reuse-Phase) In der Reuse-Phase wird, unter Ausnutzung der in den ausgewählten Fällen enthaltenen Erfahrungen, eine Lösung des Problems bestimmt.

Dieser Schritt kann je nach Aufgabenbereich verschieden aufwendig ausfallen. Für analytische Problemstellungen besteht die Wiederverwendung meist darin, daß die Lösung, die in dem durch das Retrieval ausgewählten Fall enthalten ist, unverändert als Lösung für das neu vorgelegte Problem verwendet wird. Eine Anpassung der Lösung ist hierbei oft nicht erforderlich. In synthetischen Bereichen, wie z. B. bei der Konfiguration oder der Aktionsplanung, ist die Lösungsanpassung jedoch unumgänglich. Der Grund hierfür liegt vor allem darin, daß das Spektrum möglicher Lösungen in synthetischen Bereichen wesentlich größer ist, als in analytischen. Daher kann in analytischen Bereichen zwar davon ausgegangen werden, daß für jede mögliche Lösung ein Fall vorliegt. In synthetischen Bereichen ist diese Annahme jedoch vollkommen unrealistisch.

Wir unterscheiden zwei grundsätzlich verschiedene Ansätze zur Lösungsanpassung: *Adaption durch Transformation* und *generative Adaption* [Cunningham *et al.*, 1994]. Die Adaption durch Transformation benutzt eine Reihe domänenspezifischer Transformationsheuristiken, die eine Lösung aufgrund von Unterschieden in den Problemstellungen verändern. Solche Transformationsheuristiken werden häufig in Form von Regeln repräsentiert, deren Vorbedingung einen bestimmten Unterschied zwischen Problemstellungen überprüft und deren Aktionsteil eine gezielte Veränderung der Lösung beschreibt. Beispiele für solche Ansätze können in [Hammond, 1989; Ashley, 1990; Smyth & Cunningham, 1992; Smyth & Keane, 1994; Bergmann *et al.*, 1994b] gefunden werden.

Generative Adaption basiert wesentlich auf der Benutzung generischer Problemlösetechniken zur Lösungsanpassung. Hierbei wird vorausgesetzt, daß ein generischer Problemlöser verfügbar ist, der vorgelegte Probleme prinzipiell lösen kann, jedoch aufgrund der Rechenzeitkomplexität hieran in der Regel scheitert. Die in der Problemlösung der Retrievalphase bestimmten Fälle werden bei der generativen Anpassung benutzt, um den generativen Problemlöser bei der Lösungsfindung zu führen. Die Art der Führung, die ein Fall bieten kann, ist dabei von System zu System verschieden und hängt insbesondere von der Art des verwendeten Problemlösers ab [Bergmann *et al.*, 1996]. Ein Grundprinzip dabei ist jedoch das Wiedergeben von *Entscheidungen*, die zur Lösung des im Fall enthaltenen Problems getroffen wurden. Solche Entscheidungen betreffen die Auswahl zwischen verschiedenen alternativen Lösungswegen im generischen Problemlöseprozeß. Bei der Wiederverwendung eines Falls werden möglichst viele relevante Entscheidungen aus den ausgewählten Fällen übernommen, während die verbleibenden Entscheidungen durch den generativen Problemlöser neu getroffen werden müssen. Beispiele für diese Vorgehensweise finden sich in [Koton,

1988; Veloso, 1992a; Paulokat & Wess, 1993; Pfitzner, 1993; Muñoz *et al.*, 1994; Bergmann & Wilke, 1995a; Bergmann & Wilke, 1995b]. Wir werden hierauf noch etwas genauer in Abschnitt 3.1.7 dieses Kapitels eingehen. Die in dieser Arbeit vorgestellten Ansätze zur Wiederverwendung werden im Detail in den Kapiteln 5 und 6 dargestellt.

3.1.5 Revise

Die Revise-Phase schließt sich direkt an die Reuse Phase an.

Begriffsbestimmung 3.6 (Revise-Phase) In der Revise-Phase wird die durch die Reuse-Phase bestimmte Lösung überprüft und, falls erforderlich, modifiziert.

Dieses Überprüfung der Lösung kann entweder durch eine Simluation geschehen oder durch die Validierung in der realen Welt. Ansätze zur Simulation von Lösungen sind kaum verbreitet, da dies Simulationswissen erfordert.[2] Ist solches Wissen jedoch vorhanden, so kann es auch direkt bei in der Reuse-Phase so eingesetzt werden, daß nur korrekte Lösungen entstehen. Dies gilt z. B. für alle Systeme, die zur Wiederverwendung eine generative Lösungsanpassungsstrategie verfolgen. Da auch der in dieser Arbeit entwickelte Wiederverwendungsansatz die Korrektheit der Lösung garantiert, kann auch hier darauf verzichtet werden, die entstandene Lösung nachträglich zu simulieren.

3.1.6 Retain

Die letzte Phase im Prozeßmodell des fallbasierten Schließens ist die Retain-Phase.

Begriffsbestimmung 3.7 (Retain-Phase) In der Retain-Phase werden die bei der aktuellen Problemlösung gewonnenen Erfahrungen in der Fallbasis gespeichert.

Im einfachsten Fall geschieht dies dadurch, daß die neue Problemlöseerfahrung als Fall in in die Fallbasis aufgenommen wird. Dies wird durch nahezu alle fallbasierten Systeme unterstützt. Hierbei ist es ggf. erforderlich, die Organisation der Fallbasis so anzupassen, daß auch auf den neuen Fall effizient während des Retrievals zugegriffen werden kann. Eine Frage, die sich hierbei jedoch stellt ist, ob jede neue Problemlöseerfahrung als Fall gespeichert werden muß. Bejaht man diese Frage, so ergibt sich das Problem, daß die Fallbasis nach einiger Zeit so groß wird, daß effizentes Retrieval nicht mehr möglich ist. Verneint man sie, so müssen Kriterien gefunden werden, die es erlauben zu entscheiden, ob ein Fall aufgenommen wird oder nicht. Ansätze für solche Kriterien sind in [Aha *et al.*, 1991; Francis & Ram, 1993; Smyth & Keane, 1995] zu finden. Auch in dieser Arbeit wird hierzu ein Ansatz entwickelt, der auf der expliziten Abschätzung der Nützlichkeit (siehe hierzu auch Abschnitt 3.2.5) beruht. Hierauf werden wir in Kapitel 7 ausführlich eingehen.

[2] CHEF [Hammond, 1989] ist eines der wenigen Systeme, welches über eine Reuse-Phase verfügt, da der in CHEF verwendete Wiederverwendungsansatz die Korrektheit der gefundenen Lösung nicht garantiert.

3.1.7 Fallbasiertes Schließen in der Aktionsplanung

Fallbasiertes Schließen kann eingesetzt werden, um einen Problemlöseprozeß (im Sinne von Abschnitt 2.1) effizienter zu gestalten.

In diesem Kontext kann der Begriff des Falls wie folgt präzisiert werden.

Definition 3.1 (Fall) Ein Fall F ist ein Paar $(p, \bar{o})$, wobei p ein Problem gemäß Definition 2.7 ist und $\bar{o}$ eine korrekte Lösung des Problems gemäß Definition 2.9.

Dies ist jedoch nur eine minimale Repräsentation für einen Fall. Wir werden später sehen, daß diese Repräsentation ggf. noch erweitert werden muß.

Durch die Wiederverwendung von Erfahrungen aus früheren Problemlösesituationen soll das Lösen eines neu vorgelegten Problems effizienter ablaufen. Hierzu muß man erreichen, daß der Aufwand zum Finden eines ähnlichen Falls (Retrieve) und der Aufwand zur Lösungsanpassung (Reuse) zusammengenommen geringer ist, als der Aufwand, der erforderlich ist, um die Lösung mit generischen Problemlösemethoden (siehe Abschnitt 2.5) zu finden.

Eine der ersten Methoden die hierzu entwickelt wurden, ist die *Derivational-Analogy-Methode* [Carbonell, 1986].[3] Derivational Analogy ist ein Beispiel für generative Lösungsanpassung, d. h. die Lösungsanpassung wird mit Hilfe eines generischen Problemlöser durchgeführt. Die Grundidee besteht darin, den *Problemlöseprozeß*, d. h. die durchgeführten Inferenzen, von einem Fall auf das neu vorgelegte Problem zu übertragen. Hierbei wird der Inferenzprozeß betrachtet, der zur Lösungsfindung eingesetzt wird. Dazu wollen wir einen Inferenzprozeß betrachten, der auf der bereits eingeführten Suche im Zustandsraum (siehe Abschnitt 2.5.1) basiert. Dieser Inferenzprozeß führt einen konstruktiven Beweis dafür, daß eine korrekte Lösung des vorgelegten Problems im Sinne von Definition 2.9 existiert. Dieser Beweis enthält die zur Problemlösung anzuwendenden Operatoren.

Beim Derivational Analogy wird hierzu der Fallbegriff erweitert, und man spricht von *Problemlösefällen* [Veloso, 1992a].

Begriffsbestimmung 3.8 (Problemlösefall) Ein Problemlösefall ist eine Erweiterung des Fallbegriffs aus Definition 3.1 um den Beweis der Korrektheit der im Fall enthaltenen Lösung für das Problem.

Ein Problemlösefall ist ein Fall gemäß Begriffsbestimmung 3.1, da der Korrektheitsbeweis als eine spezielle Erfahrung angesehen werden kann.

[3] Auch wenn üblicherweise der Begriff der *Analogie* für die Übertragung von Lösungen über verschiedene Domänen hinweg verstanden wird [Kedar-Cabelli, 1988; Burstein, 1989; Hall, 1989], wird hier der Analogiebegriff synonym zum Begriff des fallbasierten Schließens, also auch für die Lösungsanpassung innerhalb einer Domäne, verwendet.

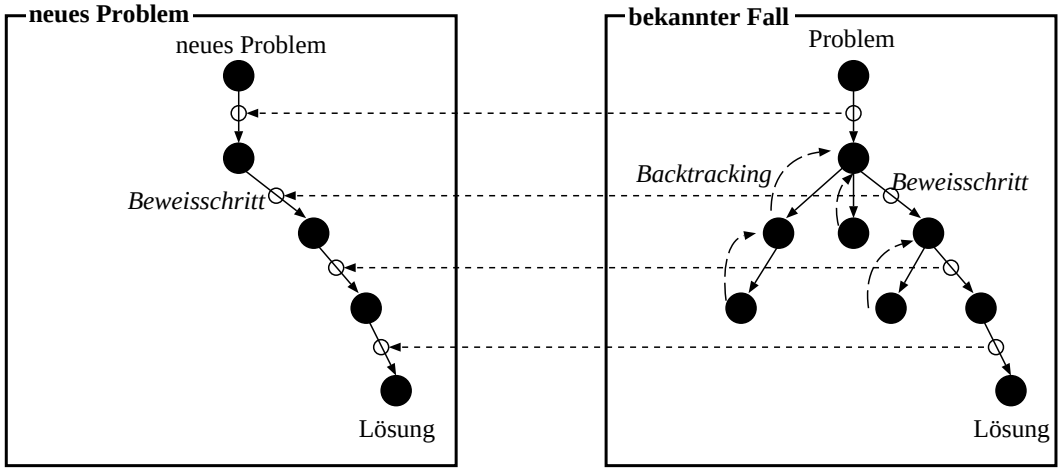

Abbildung 3.2: Übertragung von Problemlöseentscheidungen bei Derivational Analogy.

Bemerkung 3.2 Ein Problemlösefall kann im allgemeinen sämtliche Entscheidungen beinhalten, die während der Problemlösung getroffen wurden, zusammen mit einer Begründung für diese Entscheidung. Solche Entscheidungen können z. B. die Auswahl eines Ziels, die Auswahl eines Operators oder die Wahl von Operatorparameter betreffen. Auch können Entscheidungen, die sich nachträglich als falsch oder ungünstig erwiesen haben und somit ein Backtracking verursacht haben, zusammen mit einer Begründung für den Fehlschlag gespeichert werden.

Wiederverwendung bei Derivational Analogy

Um das neue Problem mit Hilfe des ausgewählten Falls zu lösen, wird nun versucht, die einzelnen Beweisschritte im Fall zur Lösung des neuen Problems nachzuvollziehen (siehe Abbildung 3.2). Dabei muß vor jedem Beweisschritt überprüft werden, ob die Voraussetzungen für die Entscheidung, die ja im Problemlösefall gespeichert sind, in der aktuellen Situation ebenfalls erfüllt sind. Dies ist erforderlich, da das neue Problem sich in der Regel von dem Problem, das im Fall vorliegt, unterscheidet. Können bestimmte Beweisschritte nicht wiederverwendet werden, so wird an dieser Stelle der herkömmliche Problemlöser verwendet, um neue Beweisschritte, ohne die Zuhilfenahme eines Falls zu finden.

3.1.8 Einige Aspekte und Probleme bei Derivational Analogy

Im folgenden sollen einige wichtige Eigenschaften und Probleme, die mit dem Einsatz von Derivational Analogy verbunden sind, erörtert werden.

Fehlende Zusicherung der Nützlichkeit wiedergegebener Entscheidungen: Da bei der Derivational Analogy nicht sichergestellt ist, daß alle Bedingungen für die vorliegende Entscheidungskette erfüllt sind, kann im allgemeinen nicht garantiert werden, daß die nachvollzogenen Entscheidungen tatsächlich zu einer Problemlösung führen. So ist es durchaus möglich, daß das Fehlen einer einzigen Vorbedingung verhindert, daß der eingeschlagene Lösungsweg auch zur einer Lösung führt. Eine kleine Änderung in der Problemstellung kann unter Umständen eine vollkommen andere Problemlösung erfordern. In diesem Fall kann ein falsch ausgewählter Fall den Problemlöseprozeß in die falsche Richtung lenken, so daß die vom Fall übernommenen Entscheidungen später wieder durch den Problemlöser zurückgenommen werden müssen. Dies kann dazu führen, daß der fallbasierte Problemlöseprozeß insgesamt mehr Zeit erfordert, als ein Problemlöser, der ohne Fälle arbeitet.

Indexierung der Fallbasis: Eine wichtige Frage beim Einsatz eines fallbasierten Ansatzes ist die nach der Indexierung der Fallbasis. Wächst die Fallbasis auf eine gewisse Größe an, so müssen Möglichkeiten gefunden werden, den ähnlichsten Fall auf effiziente Weise zu finden. Andernfalls kann die Suche nach dem ähnlichsten Fall mehr Rechenzeit beanspruchen als durch die Wiederverwendung der Lösung eingespart wird. Diesem Problem werden wir nochmal beim erklärungsbasierten Lernen begegnen und es dort ausführlicher betrachten (siehe Abschnitt 3.2.5).

Vollständigkeit und Korrektheit der Domänentheorie: Eine korrekte und vollständige Domänentheorie (Problemlösedomäne) muß vorhanden sein, damit der generische Problemlöser die generative Lösungsanpassung vornehmen kann.

3.2 Erklärungsbasiertes Lernen

3.2.1 Die Grundidee des erklärungsbasierten Lernens

Ziel des erklärungsbasierten Lernens (engl. explanation-based learning, EBL) [Mitchell *et al.*, 1986; DeJong & Mooney, 1986; Ellman, 1989; DeJong, 1993] ist es, Inferenzprozesse effizienter zu gestalten. Dies betrifft Inferenzprozesse wie Resolution in der Hornlogik [Kedar-Cabelli & McCarty, 1987; Hirsh, 1987], Produktionsregelsysteme [Rosenbloom & Newell, 1986; Rosenbloom & Laird, 1986], Deduktionsverfahren für Gleichungssysteme durch Termersetzung [Laird & Gamble, 1990; Fuchs, 1995a; Fuchs, 1995b] bzw. mathematische, vollständige Induktion [Kolbe & Walter, 1994] und Inferenzen zur Planung mit STRIPS-Operatoren [Fikes *et al.*, 1972; Tadepalli, 1991; Kambhampati & Kedar, 1994].

Beim erklärungsbasierten Lernen wird hierzu das folgende Szenario zugrunde gelegt. Die Aufgabe, die durch den betreffenden Infernzprozeß gelöst werden soll, besteht im Nachweis einer bestimmten *Konzeptzugehörigkeit* eines vorgelegten *Beispiels*. Zur Vereinfachung der

nachfolgenden Darstellung wollen wir davon ausgehen, daß Inferenzen in der Hornlogik betrachtet werden. Eine Erweiterung zur Behandlung von Inferenzen mit STRIPS-Operatoren wird in Kapitel 6 vorgestellt. Zunächst führen wir einige Begriffe ein.

Definition 3.2 (Beispiel) Ein Beispiel B ist eine endliche Menge von Gundatomen.

Definition 3.3 (Zielkonzept) Ein Zielkonzept K ist eine Formel.

Bemerkung 3.3 Das Zielkonzept ist oft von der Form $\exists x_1 \ldots \exists x_n \, p_1 \wedge \ldots \wedge p_m$, wobei $p_1, \ldots, p_m$ Atomformeln sind und $x_1, \ldots, x_n$ die darin vorkommenden Variablen bezeichnen. Bei dem Nachweis der Konzeptzugehörigkeit ist man in der Regel auch daran interessiert eine Instantiierung dieser Variablen zu finden, so daß $p_1 \wedge \ldots \wedge p_m$ gilt.

Definition 3.4 (Domänentheorie) Eine Domänentheorie Σ ist eine endliche Menge von Hornklauseln (Regeln und Fakten).

Betrachtet wird nun die Aufgabe, zu entscheiden, ob $\Sigma \cup B \vdash K$ gilt. Ist diese Bedingung erfüllt, so sagt man: das Beispiel B erfüllt das Zielkonzept K. Hierbei nehmen wir an, daß für viele verschiedene Beispiele $B_1, \ldots, B_m$ nacheinander entschieden werden soll, ob sie das Zielkonzept erfüllen. Dies erfordert für jedes Beispiel einen separaten Inferenzprozeß (in der Hornlogik), der mitunter sehr aufwendig sein kann. Dieser Inferenzprozeß kann nun wie folgt effizienter gestaltet werden: Betrachtet man die Inferenzketten zum Nachweis von $\Sigma \cup B_i \vdash K$, so liegt häufig der Fall vor, daß Teile dieser Inferenzketten für viele Beispiele B_i übereinstimmen. Daher können Inferenzen dadurch eingespart werden, daß diese wiederkehrenden Teile nur einmal durchgeführt und bei Bedarf wiederverwendet werden. Hierzu wird beim erklärungsbasierten Lernen aus einem Teil der Inferenzkette eine neue Formel gebildet, die eine Art Abkürzung darstellt.

Bemerkung 3.4 Dies entspricht der Vorhensweise bei der Bildung von Lemmas (Hilfssätze) in der Mathematik. Auch hier wird ein oft wiederkehrender Sachverhalt nur einmal bewiesen und als Lemma formuliert, das in vielen nachfolgenden Beweisen benutzt wird. Daher braucht der betreffende Sachverhalt nicht erneut bewiesen zu werden.

Abbildung 3.3 verdeutlicht diese Vorgehensweise. Hierbei stimmen die Inferenzen, die zum Nachweis der Konzeptzugehörigkeit der drei Beispiele erforderlich sind, bis zu einem Punkt S (das ist im Fall der Hornlogik eine konjunktive Formel) überein. Eine Einsparung kann dadurch erzielt werden, daß dieser Teil des Beweises nur einmal geführt wird. Technisch wird dies dadurch realisiert, daß die neue Formel $S \rightarrow K$ als Abkürzung zur Domänentheorie hinzugenommen und im Inferenzprozeß bevorzugt angewendet wird.

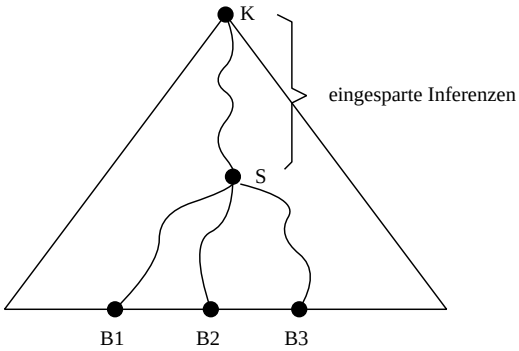

Abbildung 3.3: Einsparung von Inferenzen beim Nachweis der Konzeptzugehörigkeit dreier Beispiele

3.2.2 Die Lernphase

Beim erklärungsbasierten Lernen wird eine Implikation $S \rightarrow K$ aus einem vorgelegten *Trainingsbeispiel für das Zielkonzept* gelernt, d. h. in einem „Compilationsschritt" aus dem Trainingsbeispiel gebildet.

Definition 3.5 (Trainingsbeispiel für ein Zielkonzept) Ein Trainingsbeispiel für ein Zielkonzept ist ein Beispiel B, das das Zielkonzept K erfüllt, d. h. es gilt $\Sigma \cup B \vdash K$.

Das Ergebnis dieses Lernschrittes ist sogenannte *Konzeptspezialisierung.*

Definition 3.6 (Konzeptspezialisierung) Eine Konzeptspezialisierung eines Zielkonzeptes K ist eine Formel der Form $S \rightarrow K$, wobei gilt: $\Sigma \vdash (S \rightarrow K)$.

Bemerkung 3.5 Eine Konzeptspezialisierung ersetzt nicht die Definition des Zielkonzeptes, sondern enthält lediglich eine hinreichende Bedingung für die Konzeptzugehörigkeit. Sie stellt somit kein neues Wissen dar, da sie bereits in der deduktiven Hülle von Domänentheorie und Zielkonzept enthalten ist (siehe auch [Dietterich, 1986; Subramanian & Smith, 1988]).

Beim erklärungsbasierten Lernen wird an die gebildeten Konzeptspezialisierungen eine zusätzliche Anforderung gestellt, die zusichern soll, daß die Konzeptspezialisierung für viele neuen Beispiele ausgenutzt werden kann. Diese Anforderung wird *Operationalitätskriterium* genannt.

Definition 3.7 (Operationalitätskriterium) Ein Operationalitätskriterium Ω ist eine Relation über der Menge der Konzeptspezialisierungen.

Das Operationalitätskriterium ist eine zusätzliche Eingabe für das erklärungsbasierte Lernverfahren und wird üblicherweise dadurch definiert, daß eine Menge *operationaler Prädikate* angegeben wird [Mitchell *et al.*, 1986; Braverman & Russel, 1988]. Die Konzeptspezialisierung erfüllt genau dann das Operationalitätskriterium, wenn alle in S vorkommenden Prädikate operationale Prädikate sind.

Bemerkung 3.6 Der Begriff „Operationalität" deutet darauf hin, daß die gebildeten Konzeptspezialisierungen so formuliert sein müssen, daß die Bedingung S effizient für ein gegebenes Beispiel überprüfbar ist, d. h. daß der zum Nachweis der Konzeptzugehörigkeit verbleibende Inferenzaufwand gering ist.

Wir können nun abschließend definieren, was ein erklärungsbasiertes Lernverfahren ist.

Definition 3.8 (Erklärungsbasiertes Lernen) Erklärungsbasiertes Lernen bestimmt bei der Eingabe eines Zielkonzeptes K, eines Trainingsbeispiels B, einer Domänentheorie Σ und eines Operationalitätskriteriums Ω als Ausgabe eine operationale Konzeptspezialisierung $S \rightarrow K$ von K, die

(i) das Trainingsbeispiel umfaßt, d. h. $\Sigma \cup B \vdash S$ gilt, und
(ii) das Operationalitätskriterium erfüllt, d. h. $\Omega(S \rightarrow K)$ gilt.

Bemerkung 3.7 Die durch das erklärungsbasierte Lernen erzeugte operationale Konzeptspezialisierunge ist

(a) eine *Generalisierung* des Trainingsbeispiels und
(b) eine *Spezialisierung* des Zielkonzepts.

Durch das Lernverfahren ist sichergestellt, daß das Trainingsbeispiel immer die operationale Konzeptspezialisierung erfüllt ($\Sigma \cup B \vdash S$). In der Regel erfüllt aber nicht nur das Trainingsbeispiel die Konzeptspezialisierung, sondern auch andere Beispiele. Daher wird sie als Generalisierung bezeichnet. Das erklärungsbasiertes Lernen wählt eine spezielle Möglichkeit aus, um das Zielkonzept zu erreichen. Diese wird durch die ausgewählte Inferenzkette, festgelegt. Folglich ist die operationale Konzeptspezialisierung immer auch eine Spezialisierung des Zielkonzeptes, denn es gilt: $\Sigma \vdash S \rightarrow K$.

Lernen aus mehreren Trainingsbeispielen

Erklärungsbasiertes Lernen bildet aus *einem* Trainingsbeispiel *eine* operationale Konzeptspezialisierung. Liegen mehrere Trainingsbeispiele $B_1, \ldots, B_m$ vor, so wird aus jedem Trainingsbeispiel B_i eine operationale Konzeptspezialisierung $S_i \rightarrow K$ gebildet. Es ergibt sich folglich eine *Menge operationaler Konzeptspezialisierungen* $\{S_1 \rightarrow K, \ldots, S_m \rightarrow K\}$.

3.2.3 Eine allgemeine Prozedur zum erklärungsbasierten Lernen

Nachdem wir spezifiziert haben, was unter erklärungsbasiertem Lernen zu verstehen ist, wird nun das Grundprinzip vorgestellt, nach dem die Verfahren zur Durchführung dieses Lernschrittes arbeiten. Hierbei beschränken wir uns auf eine informelle Beschreibung; eine formale Darstellung wird in Kapitel 6 erfolgen.

Die allgemeine Vorgehensweise beim erklärungsbasierten Lernen läßt sich grob in zwei Schritte zerlegen:

1. Erzeugung einer Erklärung (Beweis), die unter Ausnutzung der Domänentheorie zeigt, daß das Trainingsbeispiel das Zielkonzept erfüllt.

2. Bestimmung einer hinreichenden Menge von Bedingungen, die das Operationalitätskriterium erfüllen und zusichern, daß die vorliegende Erklärung weiterhin eine Erklärung für die Gültigkeit des Zielkonzeptes ist.

Der hierbei verwendete Begriff der *Erklärung* bedarf noch einer weiteren Detaillierung.

Begriffsbestimmung 3.9 (Erklärung) Eine Erklärung ist ein Beweis dafür, daß ein Trainingsbeispiel das Zielkonzept erfüllt.

Bemerkung 3.8 Der Begriff der Erklärung, so wie er im erklärungsbasierten Lernen verwendet wird, hat eine leicht andere Bedeutung als z. B. im Bereich der Expertensysteme. Beim erklärungsbasierten Lernen spielen Aspekte wie z. B. die Verständlichkeit der Erklärung für einen Benutzer keine Rolle. Erklärungen sind hier rein intern verwendete Datenstrukturen und stellen keine Schnittstelle zur Außenwelt dar.

Ursprünglich wurden Erklärungen als Bäume repräsentiert (siehe z. B. [Mitchell *et al.*, 1986]). Wir wollen hier diese Repräsentation etwas erweitern und eine Erklärung als einen zusammenhängenden, azyklischen, gerichteten Graphen darstellen.[4] Diese Erweiterung wird insbesondere zur Behandlung von Inferenzen mit STRIPS-Operatoren erforderlich sein (siehe Abschnitt 6.3.4). Der Graph, der eine Erklärung repräsentiert, besitzt genau einen Knoten, von dem keine weiteren gerichteten Kanten ausgehen. Dieser ausgezeichnete Knoten repräsentiert das Zielkonzept. Knoten, von denen nur gerichtete Kanten ausgehen, repräsentieren Fakten, die entweder in der Domänentheorie vorkommen oder durch das Trainingsbeispiel gegeben sind. Die Menge aller gerichteten Kanten, die zu einem Knoten hinführen, repräsentiert eine Regel aus der Domänentheorie. Die Knoten, die durch solche gerichtete Kanten erreicht werden, stellen die Konklusion der Regel dar, während die Knoten, von denen die Kanten ausgehen, die Prämissen der Regel repräsentieren. Eine formale Definition hierzu wird in Abschnitt 6.3 gegeben.

[4] Eine ähnliche Erweiterung des Erklärungsbegriffes wird in [Mooney, 1993] gegeben.

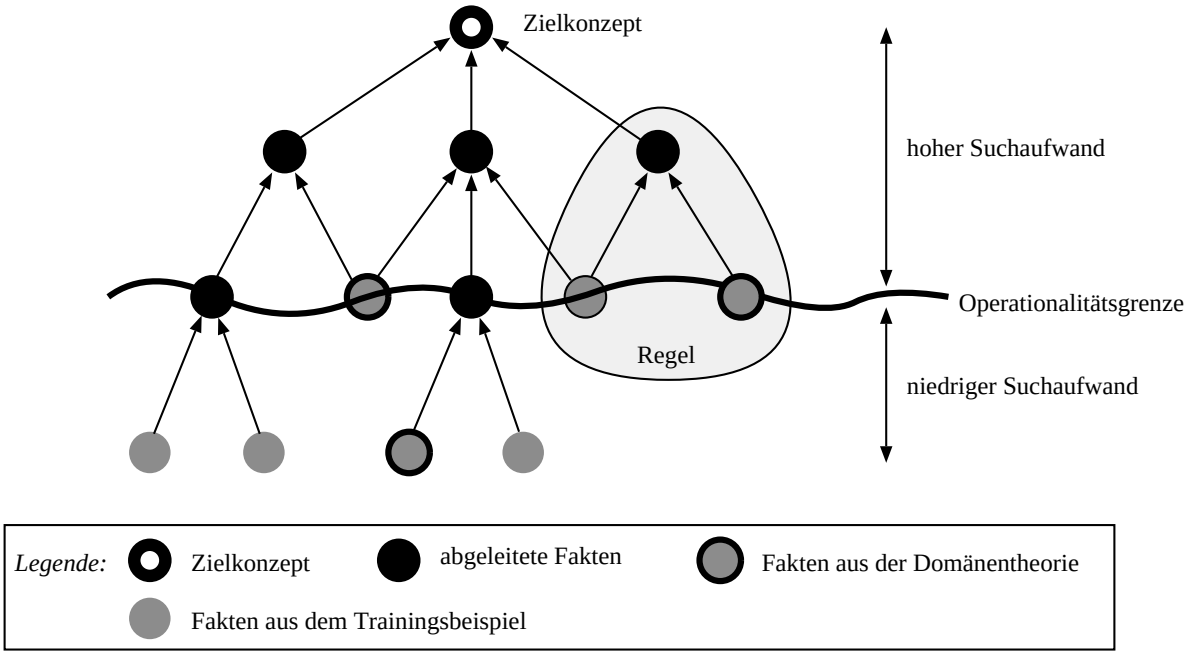

Abbildung 3.4: Beispiel eines Erklärungsgraphen

Ein Beispiel für eine Erklärung in Form eines Graphen ist in Abbildung 3.4 dargestellt. Ausgehend von einer Menge von Fakten der Domänentheorie und der Menge der Fakten, die das Trainingsbeispiel beschreiben, wird hier das Zielkonzept durch die Anwendung einer Menge von Regeln hergeleitet. Hierbei entsteht eine Graphenstruktur, bei der ein Fakt für die Erfüllung der Vorbedingung mehrerer Regeln verwendet werden kann.

Während im ersten Schritt des erklärungsbasierten Lernes eine solche Erklärung konstruiert wird, erfolgt im zweiten Schritt die Umformulierung des Zielkonzeptes in eine operationale Form. Hierzu wird eine Zielregression[5] [Waldinger, 1977] des Zielkonzeptes durch den Beweisbaum durchgeführt, bis zu dem Punkt, bei dem die bestimmte Vorbedingung das Operationalitätskriterium erfüllt. Diese Vorbedingung wird zur Prämisse S der operationalen Konzeptspezialisierung $S \rightarrow K$. Dieses Verfahren wird in Abschnitt 6.4 detailliert dargestellt.

Bemerkung 3.9 Durch diesen Prozeß wird der gesamte Suchraum, der zum Nachweis des Zielkonzeptes durchsucht werden muß, in zwei Bereiche zerlegt (siehe Abbildung 3.4). Die

[5] Bei der Zielregression werden, ausgehend vom Zielkonzept, die Regeln der Domänentheorie so angewandt, wie dies durch die vorliegende Erklärung für das Trainingsbeispiel vorgegeben ist. Bei jeder Regelanwendung wird die schwächste Vorbedingung bestimmt, die zusichert, daß das Zielkonzept, unter Voraussetzung dieser Regeln, erfüllt ist. Dies entspricht genau der schwächsten Vorbedingung im Sinne von Dijkstra [1976]. Eine genauere Beschreibung der Zielregression kann auch KI-Lehrbüchern (z. B. [Nilsson, 1980]) entnommen werden.

Abgrenzung zwischen diesen beiden Bereichen erfolgt durch das Operationalitätskriterium. Um Bedingungen zu lernen, die für das spätere Überprüfen neuer Beispiele auf die Zugehörigkeit zum Zielkonzept nützlich sind, muß der gesamte Suchraum so zerlegt werden, daß der größte Teil des Suchaufwands in dem Bereich des Suchraumes erforderlich ist, der „oberhalb" des Operationalitätsgrenze liegt. Der verbleibende Aufwand zum Nachweis der Bedingung, die dem Operationalitätskriterium genügt, muß vergleichsweise gering sein.

3.2.4 Die Anwendungsphase

Wir betrachten nun den Einsatz der gelernten operationalen Konzeptspezialisierungen beim Nachweis der Konzeptzugehörigkeit neuer Beispiele. Der nachfolgende einfache Algorithmus 3.5 realisiert dies. Hierbei wird der Reihe nach zunächst versucht, die Vorbedingung der vorhandenen operationalen Konzeptspezialisierungen zu beweisen. Gelingt dies für eine Konzeptspezialisierungen, so ist nachgewiesen, daß das Beispiel das Zielkonzept erfüllt. Gelingt dies nicht, so wird die Domänentheorie direkt herangezogen. Das Ergebnis ist in jedem Fall eine korrekte Entscheidung der Konzeptzugehörigkeit.

Eingabe: Ein Beispiel B, ein Zielkonzept K, eine Domänentheorie Σ und eine Menge operationaler Konzeptspezialisierungen $\{S_1 \rightarrow K, \ldots, S_m \rightarrow K\}$.

Ausgabe: Entscheidung ob $\Sigma \cup B \vdash K$ gilt.

for $i := 1$ **to** m **do**
  if $\Sigma \cup B \vdash S_i$ gilt, **then** STOP „Beispiel erfüllt das Zielkonzept"
  end
if $\Sigma \cup B \vdash K$ gilt
  then STOP „Beispiel erfüllt das Zielkonzept"
  else STOP „Beispiel erfüllt nicht das Zielkonzept"

Abbildung 3.5: Basisalgorithmus zur Bestimmung der Zugehörigkeit eines Beispiels zu einem Zielkonzept.

3.2.5 Nützlichkeitsbetrachtungen

Für das erklärungsbasierte Lernen spielt die *Nützlichkeit* der gelernten operationalen Konzeptspezialisierungen eine wichtige Rolle bei der Beurteilung des erzielbaren Effizienzgewinns. Der hierbei verwendete Nützlichkeitsbegriff geht ursprünglich zurück auf v. Neu-

mann und Morgenstern [1944].[6] Wir können die Nützlichkeit auf der Grundlage der *Kosten* in Form von erforderlicher Rechenzeit bestimmen. Die Nützlichkeit entspricht dabei den negativen Kosten, d. h. geringe Kosten bedeuten eine hohe Nützlichkeit.

Globale Nützlichkeitsbetrachtungen

Die folgenden Betrachtungen berücksichtigen die gesamten anfallenden Kosten, sowohl für das Lernen als auch für die Entscheidung der Konzeptzugehörigkeit. Wir wollen zunächst einige Bezeichnungen einführen:

C_K: Kosten zur Entscheidung der Konzeptzugehörigkeit eines Beispiels ohne vorheriges Lernen. Das sind genau die Kosten zur Bestimmung von $\Sigma \cup B \vdash K$. [7]

$C_{K'}$: Kosten zur Bestimmung der Konzeptzugehörigkeit eines Beispiels durch Ausnutzung von gelernten operationalen Konzeptspezialisierungen.

C_L: Kosten, die durch das erklärungsbasierte Lernen selbst verursacht werden.

Sei weiterhin α die Anzahl der Beispiele, für die die Konzeptzugehörigkeit entschieden werden soll, so ergeben sich die Gesamtkosten bei Verwendung des Lernansatzes zu: $C_L + \alpha \cdot C_{K'}$. Dies steht den Kosten $\alpha \cdot C_K$, bei Verzicht auf das Lernen, gegenüber. Der Einsatz des Lernverfahrens führt also genau dann zu einer Effizienzsteigerung, wenn $C_L + \alpha \cdot C_{K'} < \alpha \cdot C_K$ gilt.

In der Regel geht man jedoch davon aus, daß die Anzahl α der Beispiele sehr groß ist oder daß der Lernaufwand auch offline erfolgen kann. Unter dieser Voraussetzung brauchen die Lernkosten C_L nicht berücksichtigt zu werden, so daß sich der Lernansatz lohnt, wenn $C_{K'} < C_K$ ist.

Detaillierte Nützlichkeitsbetrachtungen

Im folgenden werden die Lernkosten vernachlässigt. Wir wollen nun die auftretenden Gesamtkosten $C_{K'}$ für die Entscheidung der Konzeptzugehörigkeit eines Beispiels genauer betrachten, um Aussagen darüber machen zu können, unter welchen Bedingungen gelernte operationale Konzeptspezialisierungen eine Reduktion der Kosten ermöglichen.

Bei der Ermittlung der Gesamtkosten $C_{K'}$, die bei Anwendung von Algorithmus 3.5 entstehen, muß beachtet werden, daß der Ausgang der Tests $\Sigma \cup B \vdash S_i$ nicht vorhergesagt werden kann. Daher betrachten wir den Ausgang dieser Tests als Zufallsereignis, das mit

[6] Nach v. Neumann und Morgenstern [1944] beschreiben Nützlichkeiten (individuelle) Präferenzen zwischen alternativen Ereignissen. Nützlichkeiten werden dabei numerische Werte zugeordnet, deren Ordnung die Präferenz wiederspiegelt. Liegen weiterhin Warscheinlichkeiten für alternative Ereignisse vor, so können die Nützlichkeitswerte wie mathematische Erwartungswerte kombiniert werden.

[7] Wir unterscheiden nicht zwischen Kosten für verschiedene Beispiele.

einer bestimmten Wahrscheinlichkeit behaftet ist. Mit P_1 wird nun die Wahrscheinlichkeit dafür bezeichnet, daß $\Sigma \cup B \vdash S_1$ gilt. P_2 bezeichnet die Wahrscheinlichkeit dafür, daß $\Sigma \cup B \vdash S_1$ nicht gilt und gleichzeitig $\Sigma \cup B \vdash S_2$ gilt. Allgemein ist P_i die Wahrscheinlichkeit dafür, daß $\Sigma \cup B \vdash S_j$ nicht gilt für $j \in \{1, \ldots, i-1\}$ und das $\Sigma \cup B \vdash S_i$ gilt. Liegen m operationale Konzeptspezialisierungen vor, so gilt folglich $\sum_{i=1}^{m} P_i \leq 1$. Des weiteren bezeichnet C_{S_i} die Kosten zur Bestimmung von $\Sigma \cup B \vdash S_i$.

Die Gesamtkosten $C_{K'}$ lassen sich nun wie folgt bestimmen:

$$\begin{aligned} C_{K'} \quad = \quad & C_{S_1} + \\ & C_{S_2} \cdot (1 - P_1) + \\ & C_{S_3} \cdot (1 - P_1 - P_2) + \cdots + \\ & C_{S_m} \cdot (1 - P_1 - \cdots - P_{m-1}) + \\ & C_K \cdot (1 - P_1 - \cdots - P_m) \end{aligned}$$

Die Kosten sind hierbei wie Nützlichkeiten im Sinne der Theorie von v. Neumann und Morgenstern [1944] kombiniert. Sie entsprechen dabei einem mathematischen Erwartungswert. Die Kosten C_{S_1} fallen in jeden Fall (mit Wahrscheinlichkeit $P = 1$) an. Die Kosten C_{S_2} fallen nur dann an, wenn der Test $\Sigma \cup B \vdash S_1$ fehlschlägt. Dies geschieht gerade mit der Wahrscheinlichkeit $1 - P_1$. Die Kosten C_{S_3} fallen nur dann an, wenn sowohl $\Sigma \cup B \vdash S_1$ als auch $\Sigma \cup B \vdash S_2$ fehlschlagen, usw. Schlagen alle Beweisversuche von $\Sigma \cup B \vdash S_i$ fehl – dies geschieht mit der Wahrscheinlichkeit $1 - P_1 - \cdots - P_m$ – so fallen die Kosten für den direkten Test mit der Domänentheorie (C_K) an. Auch bei dieser Betrachtung lohnt sich der Einsatz des erklärungsbasierten Lernens genau dann wenn $C_{K'} < C_K$ gilt.

Wir wollen abschließend noch den einfachen Sonderfall betrachten, daß nur eine operationale Konzeptspezialisierung gelernt wurde. Es gilt folglich: $C_{K'} = C_{S_1} + C_K \cdot (1 - P_1)$. In diesem Fall besagt die Bedingung $C_{K'} < C_K$, daß die einzelne gelernte operationale Konzeptspezialisierung $S_1 \rightarrow K$ nützlich ist, d. h. im Mittel eine Effizienzsteigerung bewirkt. Dies gilt genau dann wenn gilt:

$$\frac{C_{S_1}}{P_1} < C_K.$$

Hieraus erkennt man die wesentlichen Faktoren, von denen die Nützlichkeit einer operationalen Konzeptspezialisierung abhängt: die Wahrscheinlichkeit mit der die Konzeptspezialisierung anwendbar ist und die Kosten, die mit der Anwendung verbunden sind. Die Kosten sollten möglichst gering sein und die Wahrscheinlichkeit für die Anwendbarkeit möglichst hoch.

Im weiteren Verlauf dieser Arbeit werden wir dieses Kostenmodell noch weiter detaillieren und auch zur Auswahl möglichst nützlicher generalisierter abstrakter Fälle anwenden (siehe Abschnitt 7.3).

3.2.6 Einige Aspekte und Probleme des erklärungsbasierten Lernens

Im folgenden sollen einige wichtige Eigenschaften und Probleme, die mit dem Einsatz von erklärungsbasiertem Lernen verbunden sind, erörtert werden.

Das Nützlichkeitsproblem

Im Gebiet des erklärungsbasierten Lernens wurde die Bedeutung der Nützlichkeit, so wie sie zuvor eingeführt wurde, erst relativ spät erkannt. In einigen Arbeiten wird darüber berichtet, daß die Nützlichkeit der Gesamtmenge der operationalen Konzeptspezialisierungen stark abnehmen kann, insbesondere wenn diese Menge sehr groß wird [Tambe & Newell, 1988; Minton, 1988; Minton, 1990b]. Diese Problem wird als Nützlichkeitsproblem (engl. utility problem) des erklärungsbasierten Lernens bezeichnet. Im Rahmen dieser Arbeit wird das Nützlichkeitsproblem des erklärungsbasieren Lernens durch explizite Nützlichkeitsberechnungen auf der Grundlage eines Kostenmodells behandelt (siehe Abschnitt 7.3).

Operationalitätskriterium

Für die Nützlichkeit von Konzeptspezialisierungen hat das Operationalitätskriterium eine entscheidende Bedeutung. Das Operationalitätskriterium bestimmt die Lage der Konzeptspezialisierung zwischen dem speziellen Beispiel und der allgemeinen Zielkonzeptdefinition (siehe Abbildung 3.6). Hierbei ist es in der Regel so, daß die Kosten zur Anwendung der Konzeptspezialisierung C_{S_i} geringer sind, je spezieller das generierte Konzept ist, also je „tiefer“ die Operationalitätsgrenze liegt [Segre, 1987]. Von diesem Gesichtspunkt aus gesehen ist man also daran interessiert, eine möglichst spezielle Konzeptspezialisierung zu finden. Auf der anderen Seite ist man jedoch auch daran interessiert, eine Konzeptspezialisierung zu finden, die für möglichst viele neue Beispiele angewendet werden kann, so daß die Wahrscheinlichkeit P_i möglichst groß wird. Aus diesem Grunde sollte man eigentlich versuchen, eine möglichst allgemeine operationale Konzeptspezialisierung zu erlernen. Diese beiden Ziele sind jedoch offensichtlich gegensätzlich. Die Wahl des Operationalitätskriteri-

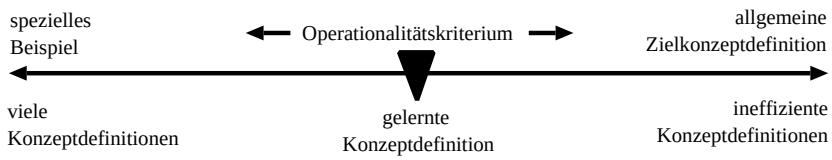

Abbildung 3.6: Bestimmung der Lage der gelernten operationalen Zielkonzeptdefinition

ums muß daher genau bedacht werden. In dieser Arbeit schaffen wir daher die Möglichkeit zur flexiblen Definition von Operationalitätskriterien (siehe Abschnitt 6.4).

Repräsentation der Domänentheorie

Ein weiteres Problem des erklärungsbasierten Lernens besteht darin, daß die erlernte Konzeptspezialisierung sehr stark von der vorliegenden Domänentheorie und deren Repräsentation abhängt. Dies wird offensichtlich, wenn man bedenkt, daß die erlernte Konzeptspezialisierung immer eine Spezialisierung der ursprünglichen Zielkonzeptdefintion und somit der Domänentheorie ist. Eine Domänentheorie zur Defintion eines Konzeptes kann aber auf viele unterschiedliche Weisen gebildet werden. Man kann z. B. eine geringe Anzahl von sehr allgemeinen Regeln mit Variablen verwenden, von denen verschiedene Instantiierungen gebildet werden können. Alternativ ist es jedoch auch möglich, auf die Verwendung von Variablen weitgehend zu verzichten und sehr viele spezielle Regeln anzugeben, die gemeinsam das gewünschte Konzept definieren. Dieser Unterschied ist aber für die durch erklärungsbasiertes Lernen gebildeten Konzeptspezialisierungen wesentlich. Enthält die Domänentheorie überwiegend spezielle Regeln ohne Variable, so ist auch jede gelernte Konzeptspezialisierung speziell und enthält nur wenige Variablen. Ist die Domänentheorie jedoch überwiegend aus allgemeinen Regeln gebildet, so kann diese Allgemeinheit der Regeln auch für die operationale Konzeptspezialisierung erhalten bleiben. Diese Betrachtungen gelten jeweils unabhängig vom festgelegten Operationalitätskriterium. Hieraus ergibt sich, daß man die Formulierung der Domänentheorie sehr genau bedenken muß, um die Bildung nützlicher Konzeptspezialisierungen zu ermöglichen. Dieses Problem wird im Rahmen dieser Arbeit dadurch überwunden, daß erklärungsbasiertes Lernen mit Abstraktionsansätzen kombiniert wird. Hierbei erlaubt die Abstraktion einen Wechsel der Repräsentationssprache. Die neue Repräsentationssprache kann dabei so gewählt werden, daß die zuvor beschriebenen Probleme vermieden werden (siehe Abschnitte 4.2.3 und 8.1).

Vollständigkeit und Korrektheit der Domänentheorie

Eine Voraussetzung für die Anwendung des erklärungsbasierten Lernens ist die Vollständigkeit und Korrektheit der Domänentheorie. Erklärungsbasiertes Lernen ist grundsätzlich nicht in der Lage, aus Beispielen zu lernen, für die die Konzeptzugehörigkeit nicht durch die Domänentheorie nachgewiesen werden kann. Außerdem garantiert die Korrektheit der Domänentheorie die Korrektheit der gelernten Konzepte. Ist die vorliegende Domänentheorie nicht korrekt, so ist es auch die gelernte Konzeptspezialisierung in der Regel nicht. Ansätze zur Behandlung dieser Probleme durch die Kombination von induktiven und erklärungsbasierten Verfahren werden in [Hall, 1988; Ellman, 1989; Bergadano & Giordana, 1990; Lebowitz, 1990; Genest *et al.*, 1990; Wilkins, 1990; Tecuci & Kodratoff, 1990; Duval, 1991] gegeben. Auch das in jüngster Zeit entwickelte *Induktive Logische Programmieren* [Muggleton, 1991], kann als ein Ansatz zur Behandlung dieser Problematik angesehen werden. Bei der Verwendung von erklärungsbasiertem Lernen im Kontext des Problemlösens

ist diese Anforderung an die Domänentheorie jedoch unproblematisch, da eine vollständige und korrekte Domänentheorie in Form der Problemlösedomäne ohnehin für den Problemlöser erforderlich ist.

3.2.7 Hintergrund: Ursprünge des erklärungsbasierten Lernens

Das Forschungsgebiet des erklärungsbasierten Lernens hat sich etwa Mitte der 80'er Jahre als ein Teilgebiet des Maschinellen Lernens entwickelt. Dieses Gebiet ist im wesentlichen aus der Konvergenz einer Reihe von ursprünglich verschiedenen Ansätzen zur Behandlung der folgenden grundsätzlichen Probleme aus der Künstlichen Intelligenz und den Kognitionswissenschaften entstanden [Ellman, 1989].

Gerechtfertigte Generalisierung: Im Maschinellen Lernen wird typischerweise neues Wissen, Konzepte genannt, aus einer Menge von Beispielen induktiv gelernt. Ein solches Konzept ist eine Generalisierung der gesehenen Beispiele. Üblicherweise fordert man, daß diese Generalisierung mit allen bekannten Beispielen konsistent und dabei entweder möglichst allgemein oder möglicht speziell ist. Da es sich bei der Bildung einer solchen Generalisierung um einen induktiven Prozeß handelt, kann die Korrektheit des gelernten neuen Wissens in der Regel nicht garantiert werden. Auch ist es schwierig, bereits bekanntes sicheres Wissen in einem solchen Generalisierungsschritt zu berücksichtigen. Aus dieser Problematik heraus wurden Ansätze entwickelt, um gerechtfertigte Generalisierungen zu bilden, d. h. solche, bei denen eine Hintergrundtheorie von gesichertem Wissen bei der Generalisierung berücksichtigt werden kann. Beispiele für solche Ansätze können in [DeJong, 1981] und [Mitchell, 1983] gefunden werden.

Bildung von „Chunks“: Beim sogenannten *Chunking* werden Sequenzen oder baumartige Strukturen von Regeln oder Operatoren zu einer einzigen Regel oder einem einzigen Operator zusammengefaßt. Eine solche Technik wurde bereits im Jahre 1972 von Fikes und Mitarbeitern [Fikes *et al.*, 1972] zur Bildung von Makrooperatoren in STRIPS beschrieben. Ein Makrooperator ist eine Zusammenfassung einer Folge mehrerer atomarer Operatoren, die bereits in einem früheren Problemlöseprozeß aufgetreten ist. Die Idee hierbei war es, die Suche für die atomaren Operatoren dieses Makrooperators einzusparen und so eine Leistungssteigerung des Problemlösers zu erreichen. In SOAR [Rosenbloom & Newell, 1986; Laird *et al.*, 1987] wird eine Prozedur beschrieben, bei der eine Folge von Anwendungen von OPS-5 Produktionsregeln zu einer einzigen Produktionsregel zusammengefaßt wird, die dieselben Effekte hat, wie die gesamte Sequenz der Regeln. Einen ähnlichen Prozeß beschreibt auch Anderson in seinem ACT-Modell als *Wissenskompilation* (engl. knowledge compilation) [Anderson, 1983; Anderson, 1989]. Diese Arbeiten werden von den Autoren im wesentlichen als kognitive Modelle verstanden, die die Performanzsteigerung von Menschen durch Training mit wiederkehrenden Situationen erklären sollen.

Operationalisierung von Konzepten: Operationalisierung wird als ein Transformationsprozeß verstanden, bei dem eine nicht-operationale Beschreibung eines Konzeptes, z. B. durch eine komplexe und schwer zu handhabende Domänentheorie, in eine operationale Konzeptbeschreibung überführt wird. Hierbei ist eine operationale Konzeptbeschreibung eine solche, bei der die Begriffe, mit denen das Konzept definiert ist, mit geringem Aufwand aufgrund vorliegender Information überprüft werden kann. Operationalisierung bedeutet, eine effiziente Formulierung eines vorgegebenen Konzeptes zu finden und hat insofern auch direkt etwas mit Effizienzsteigerung zu tun. Erste Ansätze zur Operationalisierung werden in [Mostow, 1983] gegeben.

Gerechtfertigte Analogie: Analoges Schließen versucht bestimmte Eigenschaften oder Merkmale eines Objektes auf ein anderes Objekt zu übertragen. Bei klassischer Analogie gibt es keine Garantie, daß die zugeschriebene Übertragung von Merkmalen tatsächlich korrekt ist. Gerechtfertigte Analogie versucht diese Problem dadurch zu lösen, daß Folgen von Inferenzregeln oder Erklärungen übertragen werden. Diese Inferenzregeln können z. B. kausale Beziehungen ausdrücken, wie z. B. in dem Ansatz von Winston [Winston *et al.*, 1983] oder aber einzelne Schritte in einem Problemlöseprozeß, wie im Derivational-Analogy-Ansatz [Carbonell, 1986].

3.2.8 Vergleich von erklärungsbasiertem Lernen und Derivational Analogy

Im folgenden sollen die Gemeinsamkeiten und Unterschiede von erklärungsbasierten Lernen und Derivational Analogy nochmal kurz zusammengefaßt werden.

Beide Ansätze weisen die folgenden Gemeinsamkeiten auf:

- Eine vollständige und korrekte Domänentheorie[8] wird vorausgesetzt.
- Ein Beweis der Korrektheit der Lösung ist erforderlich.
- Mehrere Beweisschritte werden zusammengefaßt.
- Das gelernte bzw. vorhandene Wissen (operationale Konzeptspezialisierungen bzw. Fälle) muß zum effizienten Zugriff geeignet organisiert werden.

Neben diesen Gemeinsamkeiten ergeben sich jedoch erhebliche Unterschiede, was die Art und die Verwendung des gelernten Wissens angeht. Diese Unterschiede werden in Tabelle 3.1 zusammengefaßt.

[8] Bei Derivational Analogy ist dies die Beschreibung der Problemlösedomäne, die der generative Problemlöser voraussetzt.

	erklärungsbasiertes Lernen	Derivational Analogy
gelerntes Wissen	allgemeiner Zusammenhang	konkreter Beweis
Umfang	nicht-operationaler Beweisausschnitt	ganzer Fall
Vorgehenweise	compilierend	interpretierend

Tabelle 3.1: Vergleich zwischen EBL und Derivational Analogy

3.3 Abstraktion

Nachfolgend führen wir in die Grundprinzipen von Abstraktion und deren Verwendung im Problemlösen ein. Wir beschränken uns hierbei auf informelle Beschreibungen. Formalisierungen der für diese Arbeit relevanten Begriffe werden in Kapitel 5 gegeben.

3.3.1 Abstraktion und Konkretisierung

Wenn wir von *Abstraktion* sprechen, unterscheiden wir verschiedene Beschreibungsebenen. Wir können z. B. zwei verschiedene Ebenen betrachten, die wir als *konkrete Ebene* und *abstrakte Ebene* bezeichnen. Hierbei ist zu beachten, daß die Attribute konkret und abstrakt nicht absolut sind sondern relativ, d. h. eine Beziehung zwischen den beiden Ebenen beschreiben (siehe auch [Richter, 1992b], Seite 271 ff.). Typischerweise zeichnet sich die abstrakte Ebene dadurch aus, daß sie im Vergleich zur konkreten Ebene

(a) weniger Information beinhaltet und
(b) einfacher zu handhaben ist (z. B. Inferenzen auf einfachere Weise erlaubt).

Sind die konkrete Ebene $\mathcal{D}_c$ und die abstrakte Ebene $\mathcal{D}_a$ gegeben, so können wir die Begriffe Abstraktion und Konkretisierung als folgende Transformationen beschreiben:

Begriffsbestimmung 3.10 (Abstraktion) Abstraktion ist die Transformation einer Beschreibung von der konkreten auf die abstrakte Ebene. Wir schreiben auch $A : \mathcal{D}_c \rightarrow \mathcal{D}_a$

Begriffsbestimmung 3.11 (Konkretisierung) Konkretisierung ist die Transformation einer Beschreibung von der abstrakten auf die konkrete Ebene. Wir schreiben auch $K : \mathcal{D}_a \rightarrow \mathcal{D}_c$

Bemerkung 3.10 Abstraktion und Konkretisierung sind also Umkehrungen von einander (siehe Abbildung 3.7). Bei der Abstraktion handelt es sich um eine Abbildung von der konkreten Ebene $\mathcal{D}_c$ in die abstrakte Ebene $\mathcal{D}_a$. Jedem Konkretum wird ein eindeutiges

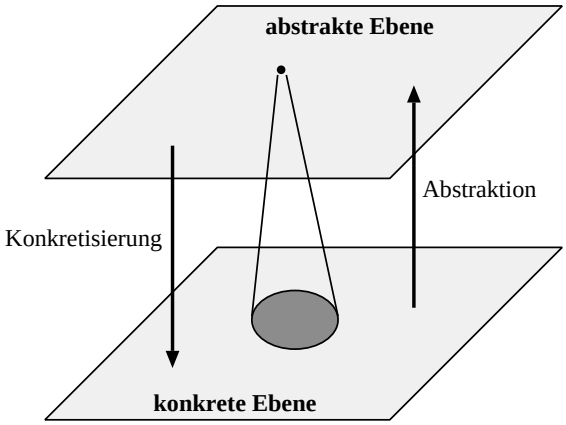

Abbildung 3.7: Abstraktion und Konkretisierung

Abstraktum zugeordnet. Mehreren Konkreta wird üblicherweise dasselbe Abstraktum zugewiesen werden. Zu einer Abstraktion gehören aber immer mehrere Konkretisierungen, da es in der Regel immer mehrere Möglichkeiten gibt, einem Abstraktum ein Konkretum zuzuweisen. Welches Konkretum hierbei auszuwählen ist, hängt vom Einsatz der Abstraktion ab. Wir fordern hier aber, daß die Abstraktion und die Konkretisierung kompatibel sind, d. h. wir fordern $A(K(x)) = x$ für alle $x \in \mathcal{D}_a$ [Richter, 1992b]. Die beiden Begriffe Abstraktion und Generalisierung werden wir später noch weiter präzisieren.

Wie bereits erwähnt, ist Abstraktion ein relativer Begriff, d. h. eine Ebene ist abstrakt in Bezug auf eine bestimmte andere Ebene. Eine solche abstrakte Ebene kann z. B. selbst wieder konkret sein in Bezug auf eine noch abstraktere Ebene. Es ist möglich, mehrere Ebenen in Bezug auf ihren Abstraktionsgrad anzuordnen, wie dies in Abbildung 3.8 gezeigt ist. Wir nennen diese Ebenen auch *Abstraktionsebenen*. Hier ist die Ebene i abstrakter als die Ebenen $i-1, i-2, \ldots, 0$. Zwischen je zwei aufeinanderfolgenden Ebenen sind Abstraktionen A_i und Konkretisierungen K_i angedeutet.

Bemerkung 3.11 Durch Hintereinanderausführung der Abstraktionen $A_i, A_{i+1}, \ldots, A_{i+k}$ ensteht eine neue Abstraktion zur Transformation von Ebene i zur abstrakteren Ebene $j = i + k + 1$. Analoges gilt für Konkretisierungen.

Der Einsatz von Abstraktion wurde ursprünglich durch die menschliche Vorgehensweise beim Problemlösen motiviert [Polya, 1945; Minsky, 1963]. Abstraktionsansätze wurden bereits erfolgreich für eine Reihe verschiedener Aufgabenbereiche eingesetzt, wie etwa im Theorembeweisen [Plaisted, 1981; Plaisted, 1986; Tenenberg, 1987], in der modellbasierten Diagnose [Mozetic, 1990] und in einer ganzen Reihe von Ansätzen im Bereich des Problemlösens [Newell & Simon, 1972; Sacerdoti, 1974; Sacerdoti, 1977; Tenenberg, 1988;

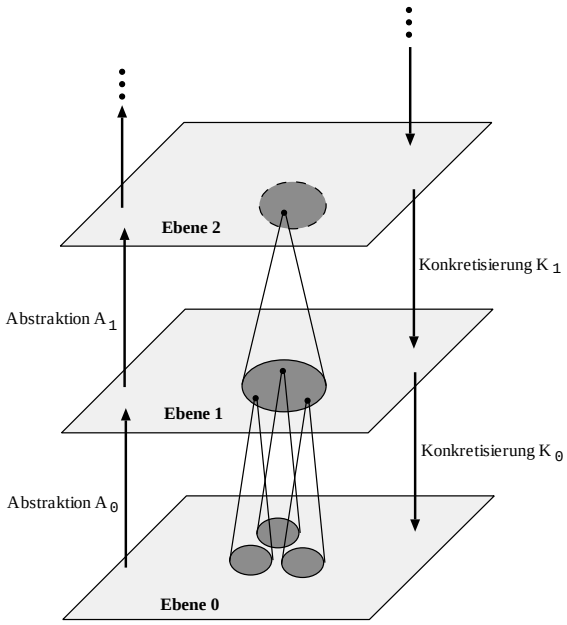

Abbildung 3.8: Mehrere Abstraktionsebenen, die in Bezug auf ihren Abstraktionsgrad geordnet sind

Unruh & Rosenbloom, 1989; Yang & Tenenberg, 1990; Knoblock, 1989; Knoblock, 1994]. Giunchiglia und Walsh [Giunchiglia & Walsh, 1992] geben einen umfassenden und vergleichenden Überblick über die Abstraktionsverfahren in den verschiedenen Bereichen. Wir wollen uns im folgenden aber auf die Verwendung von Abstraktion im Bereich des Problemlösens beschränken.

3.3.2 Hierarchisches Problemlösen

Abstraktionsansätze lassen sich beim Problemlösen einsetzen. Wir sprechen hierbei auch von *hierarchischem Problemlösen.*[9]

Begriffsbestimmung 3.12 (Hierarchisches Problemlösen) Als hierarchisches Problemlösen bezeichnen wir einen Problemlöseprozeß, bei dem verschiedene Abstraktionsebenen, in Form von Problemlösedomänen verwendet werden, um Probleme auf diesen Ebenen mit Hilfe von Abstraktion und Konkretisierung zu lösen.

Beim hierarchischen Problemlösen werden verschiedene Abstraktionsebenen, wie in Abbildung 3.9 dargestellt, vorausgesetzt. Eine Abstraktionsebene besteht dabei aus einer

[9] Mit dem Begriff *hierarchisches Problemlösen* werden gelegentlich auch HTN-Ansätze (z. B. SIPE [Wilkins, 1988]) bezeichnet. Diese wollen wir ausdrücklich nicht mit einbeziehen.

Problemlösedomäne im Sinne von Definition 2.6. Die unterste (detaillierteste) Abstraktionsebene ist die Problemlösedomäne, in der Probleme vorgelegt werden und Lösungen hierzu bestimmt werden müssen. Wir wollen diese Ebene im folgenden *konkrete Ebene* (engl. ground level) nennen (siehe auch Abbildung 3.9).

Begriffsbestimmung 3.13 (Konkrete Ebene) Die konkrete Ebene ist die niedrigste Abstraktionsebene.

Hierarchisches Problemlösen erfordert die Transformation von Problemen und Lösungen zwischen verschiedenen Abstraktionsebenen. Hierzu betrachten wir zunächst die Transformation von Zuständen zwischen verschiedenen Abstraktionsebenen. Die folgenden Begriffe sind spezielle Ausprägungen der Begriffsbestimmungen 3.10 bzw. 3.11.

Begriffsbestimmung 3.14 (Zustandsabstraktion) Eine *Zustandsabstraktion* ist eine Abbildung, die einem Zustand der Abstraktionsebene $\mathcal{D}_i$ einen Zustand auf der nächst höheren Abstraktionsebene $\mathcal{D}_{i+1}$ zuordnet.

Begriffsbestimmung 3.15 (Zustandskonkretisierung) Eine *Zustandskonkretisierung* ist eine Abbildung, die einem Zustand der Abstraktionsebene $\mathcal{D}_i$ einen Zustand auf der nächst niedrigeren Abstraktionsebene $\mathcal{D}_{i-1}$ zuordnet.

Vollständig analog lassen sich die Begriffe *Planabstraktion* und *Plankonkretisierung* ebenfalls als spezielle Ausprägungen der Begriffsbestimmungen 3.10 bzw. 3.11 festlegen.

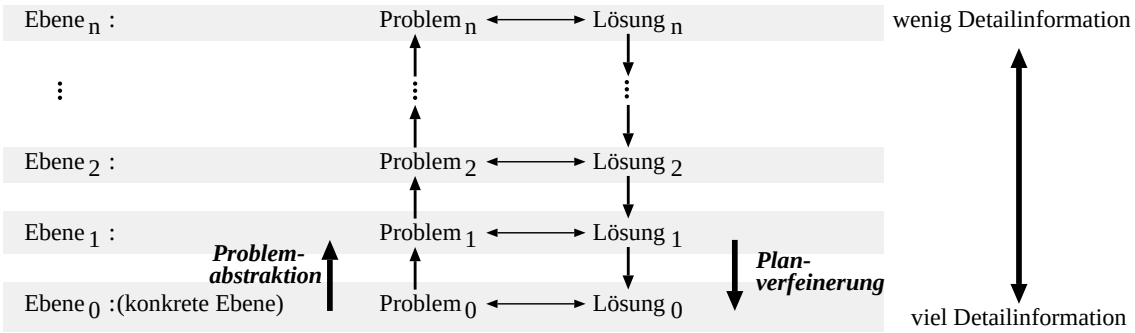

Abbildung 3.9: Problemlösen mit Abstraktion und Verfeinerung

Begriffsbestimmung 3.16 (Planabstraktion) *Planabstraktion* ist eine Abbildung, die einem Plan der Abstraktionsebene $\mathcal{D}_i$ einen Plan auf der nächst höheren Abstraktionsebene $\mathcal{D}_{i+1}$ zuordnet.

Begriffsbestimmung 3.17 (Plankonkretisierung) *Plankonkretisierung* ist eine Abbildung, die einem Plan der Abstraktionsebene $\mathcal{D}_i$ einen Plan auf der nächst niedrigeren Abstraktionsebene $\mathcal{D}_{i-1}$ zuordnet.

3.3.3 Vorgehensweise beim hierarchischen Problemlösen

Im folgenden wird die Vorgehensweise beim hierarchischen Problemlösen allgemein beschrieben. Abbildung 3.10 verdeutlicht die beiden vorkommenden Hauptschritte: Problemabstraktion und Planverfeinerung.

Problemabstraktion

Zunächst wird das vorliegende Problem auf alle betrachteten Abstraktionsebenen transformiert, d. h. das Problem wird abstrahiert. Hierzu werden die Zustandsabstraktionen auf den Initial- und den Zielzustand angewendet.

Begriffsbestimmung 3.18 (Problemabstraktion) *Problemabstraktion* ist eine Abbildung, die einem Problem (Paar, bestehend aus Initial- und Zielzustand) der Abstraktionsebene $\mathcal{D}_i$ eine Problem auf der nächst höheren Abstraktionsebene $\mathcal{D}_{i+1}$ zuordnet.

Planverfeinerung

Nach der Problemabstraktion wird zunächst eine Lösung (Plan) des vorliegenden abstrakten Problems auf der höchsten Abstraktionsebene gebildet. Hierzu wird ein gewöhnlicher generischer Problemlöseprozeß, z. B. Suche im Zustandsraum (siehe Abschnitt 2.5.1), eingesetzt. Die so entstandene abstrakte Lösung wird dann durch schrittweise Plankonkretisierung bis auf die konkrete Ebene transformiert. Hierbei wird auf jeder Abstraktionsebene eine abstrakte Lösung des Problems dadurch gebildet, daß die zuvor auf der nächst höheren Abstraktionsebene konstruierte Lösung geeignet konkretisiert wird. Die Konkretisierung wird hierbei so gebildet, daß die Lösung eine Lösung für das vorgegebene Problem auf der Abstraktionsebene ist. Wir nennen diese Art der Plankonkretisierung auch *Planverfeinerung.*

Begriffsbestimmung 3.19 (Planverfeinerung, Lösungsverfeinerung) Es seien zwei Abstraktionsebenen $\mathcal{D}_i$ und $\mathcal{D}_{i-1}$ gegeben, sowie ein Problem, daß auf jeder Abstraktionsebenen dargestellt ist. Hierbei bezeichne p_i das Problem auf der Abstraktionsebene $\mathcal{D}_i$.

Planverfeinerung (auch *Lösungsverfeinerung* genannt) ist eine Plankonkretisierung, die einem Plan $\bar{o}_i$, der eine korrekte Lösung von p_i auf der Abstraktionsebene $\mathcal{D}_i$ ist, einen Plan $\bar{o}_{i-1}$ der Abstraktionsebene $\mathcal{D}_{i-1}$ zuordnet, der eine korrekte Lösung von p_{i-1} ist.

Die Planverfeinerung wird beim hierarchischen Planen üblicherweise durch das nachfolgend skizzerte Verfahren bestimmt. Eine formale Beschreibung der Planverfeinerung ist in Abschnitt 5.4 zu finden.

Bei der Planverfeinerung wird der abstrakte Plan, der konkretisiert werden soll, zur Steuerung der Suche auf der nächst tieferen Abstraktionsebene eingesetzt (siehe Abbildung 3.10). Hierbei definieren die Zwischenzustände der abstrakten Lösung Zwischenziele auf der nächst tieferen Abstraktionsebene. Ein Zwischenziel ist dabei eine Konkretisierung des entsprechenden abstrakten Zwischenzustandes. Eine abstrakte Lösung liefert dadurch eine Problemzerlegung des konkreten Problems in mehrere kleinere Teilprobleme. Ein solches Teilproblem besteht dabei in der Suche nach a) einem Übergang vom Initialzustand zum ersten Zwischenziel, b) einem Übergang zwischen zwei aufeinanderfolgenden Zwischenzielen oder c) einem Übergang zwischen dem letzten Zwischenziel und dem Zielzustand. Diese Teilprobleme werden dann der Reihe nach gelöst und die Konkatenation der einzelnen Teillösungen ergibt eine gesamte Lösung des Problems auf der entsprechenden Abstraktionsebene. Hierbei wird in der Regel ein Operator der abstrakten Lösung durch eine Folge mehrerer Operatoren auf der nächst tieferen Abstraktionsebene verfeinert. Es gilt daher, daß die Länge der Pläne (Operatoranzahl) mit sinkendem Abstraktionsgrad monoton wächst.

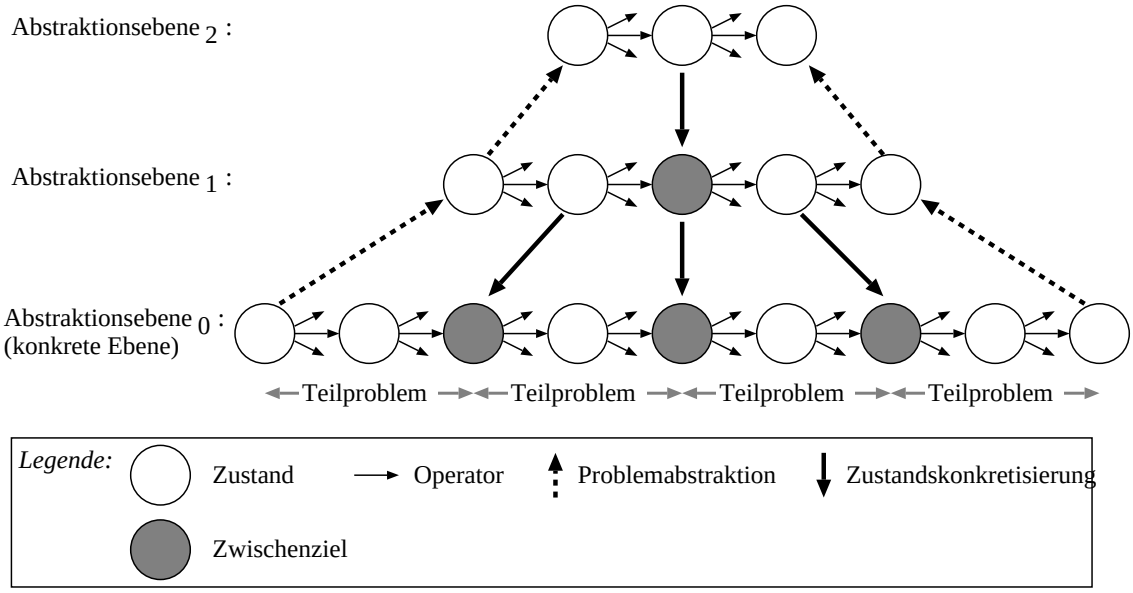

Abbildung 3.10: Hierarchisches Problemlösen

3.3.4 Kosten beim hierarchischen Problemlösen

Beim hierarchischen Problemlösen werden Probleme durch Verfeinerung eines abstrakten Plans gelöst. Wie in Abbildung 3.10 gezeigt, wird hierbei der abstrakte Plan als Problemzerlegung eingesetzt. Die zugrundeliegende Idee beruht auf der Beobachtung, daß eine Reihe kleinerer Teilprobleme insgesamt wesentlich leichter gelöst werden kann, als wenn das vollständige Problem als Ganzes bearbeitet wird. Wir erwarten daher einen Performanzgewinn durch diese Vorgehensweise. Wir werden daher im folgenden diesen Performanzgewinn mit Hilfe eines *Kostenmodells* analysieren. Dabei lehnen wir uns an das von Korf [Korf, 1987] entwickelte und von Knoblock [Knoblock, 1993] weitergeführte Modell des hierarchischen Problemlösens an. Hierbei wird die Suchraumgröße als Maß für die Kosten, die bei der Problemlösung anfallen, verwendet. Wir möchten bereits an dieser Stelle anmerken, daß dieses Modell auf einer Reihe von Annahmen beruht, die am Ende dieses Abschnitt näher untersucht werden.

Im folgenden gehen wir davon aus, daß b der Verzweigungsgrad des Suchraums ist, der zur Problemlösung durchsucht werden muß, d. h. jeder Zustand hat b Nachfolgezustände. Weiter nehmen wir an, daß $b > 1$ gilt. Weiter sei p die Länge der kürzesten Lösung (Anzahl der Operatoren) eines Problems p. Mit $C_S(b, l)$ werden nun die Kosten, die beim Problemlösen mit einem generischen Problemlöser anfallen, bezeichnet. Diese berechnen sich wie folgt (siehe Abbildung 3.11):

$$C_S(b, l) = \sum_{i=1}^{l} b^i = \frac{b^{l+1} - 1}{b - 1} \text{ für } b > 1.$$

Nun soll die Auswirkung von Abstraktion auf diese Kosten ermittelt werden. Wir nehmen daher an, daß eine abstrakte Lösung des Problems p bereits vorliegt und daß dieser die

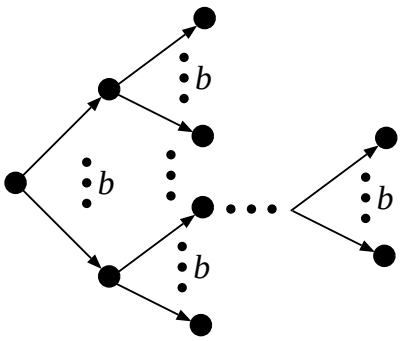

Abbildung 3.11: Suchraum mit einem Verzweigungsgrad b und einer Lösungslänge l.

Länge k besitzt. Der abstrakte Plan zerlegt das gesamte Problem p folglich k kleinere Teilprobleme p_j $(j = 1 \ldots k)$. Sei l_j die Länge der kürzesten Lösungen dieser Teilprobleme, so ergeben sich die Kosten $C'_S(b, l_1, \ldots, l_k)$, die zur Problemlösung anfallen als die Summe der Kosten, die zur Lösung der Teilprobleme anfallen. Es gilt also:

$$C'_S(b, l_1, \ldots, l_k) = \sum_{j=1}^{k} C_S(b, l_j),$$

wobei $l_1 + l_2 + \cdots + l_k = l$ gilt.

Wenn wir weiter vereinfachend davon ausgehen, daß alle Teilprobleme die gleiche Größe besitzen, d. h. alle Teilprobleme Lösungen gleicher Länge erfordern, so ergeben sich mit $l_j = l/k$ die folgenden Kosten.

$$C'_S(b, l, k) = k \cdot C_S(b, \frac{l}{k}) = k \cdot \frac{b^{\frac{l}{k}+1} - 1}{b - 1},$$

Bemerkung 3.12 Es gilt: $C'_S(b, l, 1) = C_S(b, l)$.

Verleicht man nun die Kosten $C'_S(b, l, k)$ mit den Kosten $C_S(b, l)$, so kann man eine deutliche Reduktion feststellen. Insbesondere gilt, daß die Kosten $C'_S(b, l, k)$ mit wachsenden k streng monoton fallen, wenn wir voraussetzen, daß $b > 1$ gilt. Abbildung 3.12 stellt diesen Sachverhalt an einem Beispiel dar.

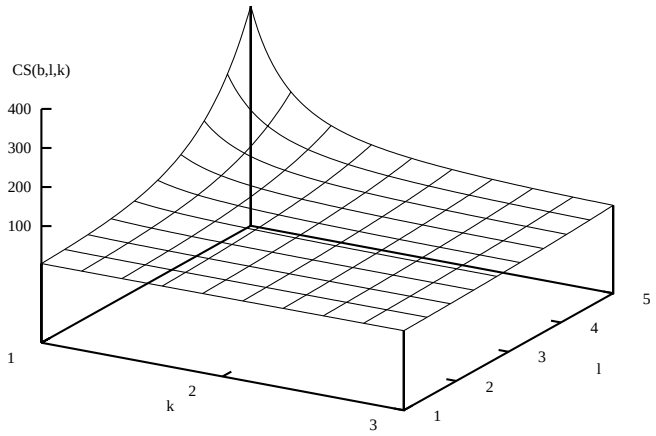

Abbildung 3.12: Beispiel für die Kosten $C_S(b, l, k)$ bei $b = 3$, $k = 1 \ldots 3$ und $l = 1 \ldots 5$.

Problemlösen durch Lösungsverfeinerung ist folglich weniger aufwendig als Problemlösen durch generative Verfahren. Vergleicht man den Aufwand zur Verfeinerung von abstrakten Plänen auf verschiedenenen Abstraktionsebenen, so kann man feststellen, daß der Verfeinerungsaufwand mit dem Abstraktionsgrad des abstrakten Planes wächst. Dies liegt daran, daß die Länge der abstrakten Lösung (k) mit steigendem Abstraktionsgrad sinkt. Diese wichtige Beobachtung werden wir im weiteren Verlauf dieser Arbeit noch ausnutzen.

Grundannahmen dieser Komplexitätsbetrachtung

Die gezeigte Analyse basiert auf einer Reihe von grundlegenden Annahmen, die hier kurz zusammengefaßt werden sollen:

- Um die Vollständigkeit des hierarchischen Problemlösens zu garantieren, müssen wir zunächst voraussetzen, daß jedes Problem, welches eine Lösung auf der konkreten Ebene besitzt, auch eine Lösung auf allen anderen Abstraktionsebenen besitzt. Diese Eigenschaft wird auch als *upward-solution property* [Tenenberg, 1988] bezeichnet.
- Darüber hinaus wird vorausgesetzt, daß sich jede Lösung, die auf einer Abstraktionsebene gefunden wird, auch auf der nächst tieferen Abstraktionsebene verfeinern läßt. Es wird also verlangt, daß die Problemdekomposition, die durch eine abstrakte Lösung bestimmt wird, in jedem Fall auch lösbare Teilprobleme definiert. Diese Eigenschaft wird auch als *downward-refinement property* [Bacchus & Yang, 1991] bezeichnet.
- Die einzelnen Teilprobleme, die durch einen abstrakten Plan definiert sind, müssen unabhängig voneinander lösbar sein.

Sind diese Bedingungen nicht erfüllt, muß zum Erhalt der Vollständigkeit des hierarchischen Problemlösers ein Backtracking zwischen einzelnen Abstraktionsebenen und zwischen der Lösung der einzelnen Teilprobleme auf einer Abstraktionsebene durchgeführt werden. Dies werden wir in Abschnitt 5.4 genauer betrachten. Ein solches Backtracking ist in der zuvor dargestellten Analyse jedoch nicht berücksichtigt und erfordert erhöhte Suchkosten, die den generellen Performanzvorteil des hierarchischen Planes zunichte machen können [Bacchus & Yang, 1994].

3.3.5 Bildung von Abstraktionsebenen

Beim hierarchischen Problemlösen stellt sich allgemein die Frage, wie geeignete Abstraktionsebenen und zugehörige Zustandsabstraktionen definiert werden können. In nahezu allen Ansätzen zum hierarchischen Problemlösen werden die Abstraktionsebenen durch sukzessives Entfernen bestimmter essentieller Sätze aus der Zustandsbeschreibung sowie der Vorbedingung, und der Add- und Deleteliste der Operatoren gebildet [Sacerdoti, 1974; Wilkins, 1984; Tenenberg, 1988; Unruh & Rosenbloom, 1989; Yang & Tenenberg, 1990; Knoblock,

1989; Knoblock, 1994; Bacchus & Yang, 1994].[10] Je höher die Abstraktionsebene ist, desto weniger essentielle Sätze aus der konkreten Repräsentation der Problemlösedomäne werden betrachtet; die Menge der essentiellen Sätze in einer Abstraktionsebene ist immer eine Teilmenge der essentiellen Sätze der nächst tieferen Abstraktionsebene. Die Annahme, die dieser sehr verbreiteten Vorgehensweise zugrunde liegt, ist, daß irrelevante Details eines Problems als isolierte Sätze repräsentiert sind. Daher soll durch das Weglassen einzelner Sätze relevante Information von weniger relevanter Information getrennt werden, so daß sich geeignete Abstraktionsebenen herausbilden. Diese vereinfachte Sicht auf Abstraktion hat einige Vorteile bei der technischen Realisierung von hierarchischen Problemlösern. So gestaltet sich die Zustandsabstraktion sehr einfach durch Ignorieren der entsprechenden essentiellen Sätze. Bei der Verfeinerung kann ein abstrakter Zustand direkt als Ziel auf der nächst tieferen Abstraktionsebene verwendet werden. Außerdem erlaubt diese Art der Abstraktion das automatische Generieren von Abstraktionshierarchien [Knoblock, 1989; Knoblock, 1994; Bacchus & Yang, 1994].

3.3.6 Einige Probleme beim Einsatz von Abstraktion

Im folgenden werden einige Probleme, die beim Einsatz von Abstraktion zum Problemlösen entstehen angesprochen. Diese sind zur Zeit Gegenstand der aktuellen Forschung in diesem Bereich.

Bildung von Abstraktionsebenen durch Elimination essentieller Sätze: In nahezu allen Ansätzen zum hierarchischen Problemlösen wird Abstraktion durch weglassen einzelner essentieller Sätze realisiert. Dieser eingeschränkte Begriff von Abstraktion hat jedoch erhebliche Nachteile. Da Abstraktion auf einem rein syntaktischen Kriterium beruht, sind die Leistungsgewinne, die durch hierarchisches Planen erzielt werden können sehr stark von der gerade vorliegenden Repräsentation der Problemlösedomäne abhängig [Holte *et al.*, 1994; Holte *et al.*, 1995; Bergmann & Wilke, 1995a]. So läßt sich sogar bei sehr einfachen Domänen wie den Türmen von Hanoi [Simon, 1975], eine Repräsentation angegeben, bei der sich die Problemlösezeit durch hierarchisches Problemlösen von einem exponentiellen zu einem linearen Verhalten verbessert, als auch eine Repräsentation, bei der keinerlei Verbesserung durch hierarchisches Problemlösen erzielt werden kann [Knoblock, 1994]. In dieser Arbeit werden wir dieses Problem dadurch umgehen, daß wir es erlauben, Abstraktionsebenen auf beliebige Weise zu bilden und dabei auch den Wechsel der Repräsentationssprache ermöglichen (siehe Abschnitt 5.1).

Nützlichkeit abstrakter Pläne: Wie bereits im vorherigen Abschnitt erörtert wurde, hängt der Grad der möglichen Komplexitätsreduktion wesentlich von der Art der erzeugten abstrakten Pläne ab. Diese sollen möglichst unabhängige, optimale und gleichgroße

[10] Einzige Ausnahmen ist Tenenbergs analogiegeleitete Abstraktion [Tenenberg, 1988].

Teilprobleme definieren. Zur Lösung eines Problems auf einer Abstraktionsebene gibt es jedoch in der Regel viele verschiedene Lösungen. Manche erfüllen die gewünschten Kriterien recht gut und können folglich als nützlich bezeichnet werden, wohingegen andere abstrakten Pläne kontraproduktiv sind [Bacchus & Yang, 1994]. Daher stellt sich die Frage nach Problemlösemethoden, die bevorzugt nützliche abstrakte Pläne generieren und verfeinern. Der in dieser Arbeit entwickelte Ansatz speichert und bewertet bereits vorgekommene abstrakte Pläne und wählt zur Problemlösung die nützlichsten Pläne aus (siehe Kapitel 7).

Teil II

PARIS: Flexible Wiederverwendung von Fällen durch Abstraktion und Generalisierung

Kapitel 4

Überblick

In diesem Kapitel werden wir eine neue Methode zur flexiblen Wiederverwendung von Fällen vorstellen und exemplarisch zeigen, wie hierdurch effizientes Problemlösen realisiert werden kann. Hierbei liegt der Schwerpunkt auf der informellen Darstellung der allgemeinen Prinzipien, der Zerlegung der Methode in klar abgegrenzte Teilaufgaben, sowie einer explarischen Anwendung. Hierdurch wird die Voraussetzung für die in den folgenden Kapiteln entwickelten formalen Darstellungen der einzelnen Teilaufgaben und derer algorithmischer Realisierung gelegt.

4.1 Wiederverwendung von Fällen auf verschiedenen Abstraktionsebenen

Die erste zentrale Idee dieses Ansatzes besteht im Einführen von Abstraktionsmethoden in das fallbasierte Schließen. Hierbei kombinieren wir die beiden Ansätze, um die Wiederverwendung von Fällen auf verschiedenen Abstraktionsebenen zu realisieren.

4.1.1 Fallbasiertes Schließen und Abstraktion

Aus dem fallbasierten Schließen entnehmen wir die Idee, Fälle als Grundlage für die Problemlösung heranzuziehen. Hierbei gehen wir von Fällen im Sinne von Definition 3.1 aus. Wie beim fallbasierten Schließen üblich, möchten wir zur Problemlösung einen geeigneten Fall aus einer Fallbasis auswählen und die in ihm enthaltene Lösung wiederverwenden. Aus dem hierarchischen Problemlösen entnehmen wir die Idee, Probleme und Lösungen auf verschiedenen Abstraktionsebenen zu betrachten. Jedoch streben wir nicht ein hierarchisches Problemlösen im konventionellen Sinne an. Vielmehr möchten wir Lösungen von Problemen auf verschiedenen Abstraktionsebenen wiederverwenden.

4.1.2 Fälle auf verschiedenen Abstraktionsebenen

Üblicherweise werden beim fallbasierten Schließen die Fälle auf einer einzigen Ebene betrachtet. In dieser Arbeit möchten wir Fälle jedoch auf verschiedenen Abstraktionsebenen betrachten. Gemäß Begriffsbestimmung 3.12 zum hierarchischen Problemlösen sind Abstraktionsebenen Problemlösedomänen im Sinne von Definition 2.6. Ein Fall liegt auf einer Abstraktionsebene, wenn die im Fall enthaltene Problemstellung und Lösung in der durch die Abstraktionsebene bestimmten Repräsentation vorliegen. Ein Fall, der auf einer niedrigen Abstraktionsebene liegt, enthält eine sehr detaillierte Beschreibung von Problem und Lösung. Ein Fall auf einer höheren Abstraktionsebene enthält hingegen nur eine sehr „grobe“ Beschreibung. Unabhängig davon auf welcher Abstraktionsebene ein Fall angesiedelt ist, muß die im Fall enthaltene Lösung immer eine korrekte Lösung des im Fall beschriebenen Problems sein. In Abhänigkeit von der Abstraktionsebene, auf der die Fälle liegen, soll im folgenden zwischen konkreten und abstrakten Fällen unterschieden werden (vgl. auch Begriffsbestimmungen 3.10 und 3.11).

Begriffsbestimmung 4.1 (Konkreter Fall) Ein konkreter Fall ist ein Fall, dessen Problem und Lösung auf der konkreten Ebene liegen.

Begriffsbestimmung 4.2 (Abstrakter Fall) Ein abstrakter Fall ist ein Fall, dessen Problem und Lösung auf einer Abstraktionsebene liegen, die höher als die konkrete Ebene ist.

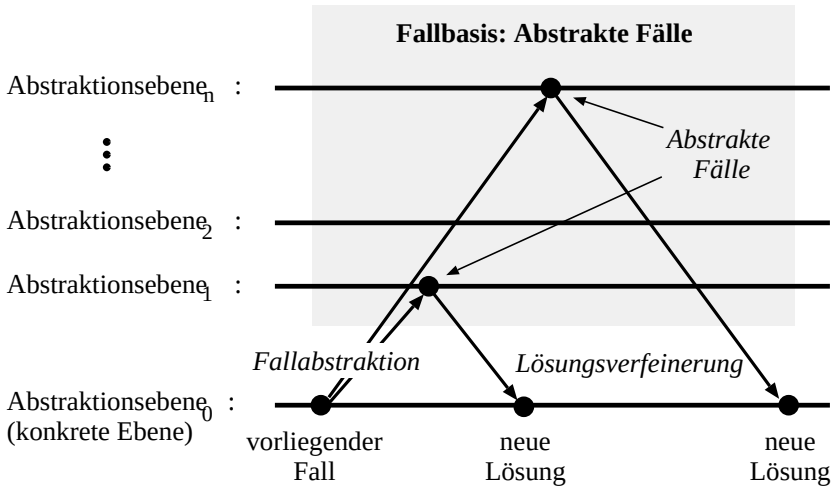

Abbildung 4.1: Wiederverwendung abstrakter Fälle

4.1.3 Die Vorgehensweise

Abbildung 4.1 zeigt schematisch die Wiederverwendung von Fällen auf verschiedenen Abstraktionsebenen. Hierbei kann der Wiederverwendungsprozeß in die beiden Teilprozesse *Fallabstraktion* und *Lösungsverfeinerung* zerlegt werden. Die Lösungsverfeinerung entspricht der des hierarchischen Problemlösens und wurde bereits in Abschnitt 3.3.3 erörtert. Fallabstraktion kommt jedoch im hierarischen Problemlösen nicht vor. Daher werden wir diese nachfolgend kurz erörtern.

Fallabstraktion

Die unterste Abstraktionsebene (konkrete Ebene) stellt die Repräsentationsebene dar, auf der Probleme und Lösungen formuliert und ausgetauscht werden müssen. Bisherige Problemlöseerfahrungen liegen also immer in Form von konkreten Fällen vor. Damit Fälle auf verschiedenen Abstraktionsebenen wiederverwendet werden können, müssen die vorliegenden konkreten Fälle in abstrakte Fälle transformiert werden. Diesen Prozeß nennen wir *Fallabstraktion* (vgl. auch Begriffsbestimmung 3.10). In den Abschnitten 5.1 und 5.2 werden wir die Fallabstraktion detailliert beschreiben.

Begriffsbestimmung 4.3 (Fallabstraktion) Fallabstraktion ist der Prozeß der Transformation eines konkreten Falls in einen abstrakten Fall.

Hierbei muß das Problem des abstrakten Falls eine Abstraktion des Problems des konkreten Falls sein (siehe Begriffsbestimmung 3.18). Ebenso muß der im abstrakten Fall enthaltene Plan eine Abstraktion des Plans des konkreten Falls sein (siehe Begriffsbestimmung 3.17).

Die Bildung von abstrakten Fällen aus konkreten Fällen braucht nicht erst zu geschehen wenn ein neues Problem gelöst werden soll. Abstrakte Fälle können bereits in einem Lernschritt „off-line“ aus den gegebenen konkreten Fällen erzeugt werden. Die so gelernten abstrakten Fälle können dann in einer Fallbasis gespeichert und entsprechend organisiert werden. Dadurch ist es dann nicht mehr erforderlich, den Fallabstraktionsprozeß bei jeder Wiederverwendung des Falls durchzuführen.

Wiederverwendbarkeit abstrakter Fälle

Zur Lösung eines neuen Problems kann nun ein vorliegender abstrakter Fall verfeinert werden. Hierbei stellt sich die Frage, wann ein abstrakter Fall zur Lösung eines konkreten Problems verwendet werden kann.

Begriffsbestimmung 4.4 (Wiederverwendbarkeit eines abstrakten Falls) Ein abstrakter Fall kann zur Lösung eines vorgelegten Problems auf der konkreten Ebene wiederverwendet werden, falls das im abstrakten Fall enthaltene Problem eine Abstraktion des vorgelegten Problems ist.

Diese im abstrakten Fall enthaltene abstrakte Lösung ist dann direkt eine Lösung des Problems und braucht nicht mehr angepaßt werden. Jedoch ist die Lösung nicht hinreichend detailliert und muß verfeinert werden (vgl. Begriffsbestimmung 3.19).

Bemerkung 4.1 Abstrakte und konkrete Fälle unterscheiden sich bezüglich ihrer Anwendbarkeit und der Kosten, die durch die Lösungsverfeinerung entstehen. Je niedriger die Abstraktionsebene ist, auf der ein Fall angesiedelt ist, um so mehr Detailinformation der Lösung steht zur Verfügung und kann wiederverwendet werden. Die vorliegende Lösung ist jedoch sehr spezifisch und kann nur in einem sehr eng begrenzten Umfeld eingesetzt werden. Lösungen aus konkreten Fälle sind sogar meist spezifisch für ein einzelnes Problem und können ohne Modifikation überhaupt nicht für abweichende Probleme eingesetzt werden. Fälle, die auf höheren Abstraktionsebenen liegen, sind für alle möglichen Konkretisierungen wiederverwendbar. Hierdurch ist das Spektrum der Situationen, in denen ein abstrakter Fall wiederverwendet werden kann, in Abhängigkeit vom Abstraktionsgrad, wesentlich erweitert. Dem gegenüber steht die Tatsache, daß die Verfeinerung einer Lösung eines abstrakteren Falls mit höheren Kosten verbunden ist als die Verfeinerung eines weniger abstrakten Falls (siehe Abschnitt 3.3.4).

4.1.4 Ein Beispiel für die Fallabstraktion

Im folgenden wollen wir die Wiederverwendung von abstrakten Fällen an einem einfachen, noch informell gehaltenen Beispiel aus dem Bereich der Arbeitsplanung (siehe auch Abschnitt 2.1.2) erörtern.

Abstraktionsebenen bei der Arbeitsplanung

Die Aufgabe in dieser Domäne besteht darin, eine Folge von Fertigungsoperationen zur Herstellung eines rotationssymmetrischen Werkstückes zu finden. Diese Aufgabe können wir auf verschiedenen Abstraktionsebenen betrachten.

Auf der konkreten Ebene berücksichtigen wir alle notwendigen Details der Problemstellung und Lösung, die relevant sind, um ein Werkstück auf einer Drehbank fertigen zu können. Hierzu gehört eine exakte Beschreibung des gewünschten Werkstücks, des Rohlings, sowie eine exakte Beschreibung der einzelnen Fertigungsschritte. Folgende Aspekte müssen hierbei betrachtet werden (siehe z. B. [Eversheim, 1989]):

- Beschreibung von Werkstück und Rohling:
  - Beschreibung des Werkstoffs, aus dem das Werkstück gefertigt werden soll.
  - Beschreibung der Kontur von Werkstück und Rohling mit genauen Abmessungen und ggf. Toleranzen.
  - Beschreibung von Anforderungen an die Oberflächenbeschaffenheit bestimmter Regionen des Werkstückes.

- Beschreibung des Fertigungsplans
  - Angabe der Aufspannung: Spannmittel, Spannkraft, Rotationsgeschwindigkeit.
  - Angabe der verwendeten Schneidwerkzeuge mit Form und Ausrichtung der Schneidplatten.
  - Beschreibung der durchgeführten Schnitte mit Schnittweg, Schnittgeschwindigkeit und Schnittiefe.

Auf einer abstrakten Ebene können wir einige dieser Details ignorieren. Folgende Aspekte können auf einer bestimmten Abstraktionsebene betrachtet werden:

- Beschreibung von Werkstück und Rohling:
  - Qualitative Angabe der Härte des Werkstoffes.
  - Grobe Beschreibung der Kontur ohne Angabe von Abmessungen und Toleranzen; Zusammenfassung von größeren Konturbereichen zu funktionalen Features (z. B. Nuten, Gewinden).
- Beschreibung des Fertigungsplans
  - Angabe der Seite und Art der Aufspannung.
  - Angabe der Kategorie der Schneidwerkzeuge.
  - Grobe Beschreibung der durchgeführten Featurebearbeitungen.

Konkrete und abstrakte Fälle in der Arbeitsplanung

Abbildung 4.2 zeigt ein Beispiel für ein einfaches Werkstück und einen Rohling. Die Angaben in dieser Abbildung beschreiben das vorliegende Fertigungsproblem auf der konkreten Ebene. Eine Lösung für dieses Problem ist in Abbildung 4.3 dargestellt. Auch diese Lösung ist auf der konkreten Ebene der Problemlösung dargestellt. Der Plan besteht insgesamt aus 8 Operatoren: Zunächst wird das Werkstück in einem Backenfutter eingespannt (Schritt

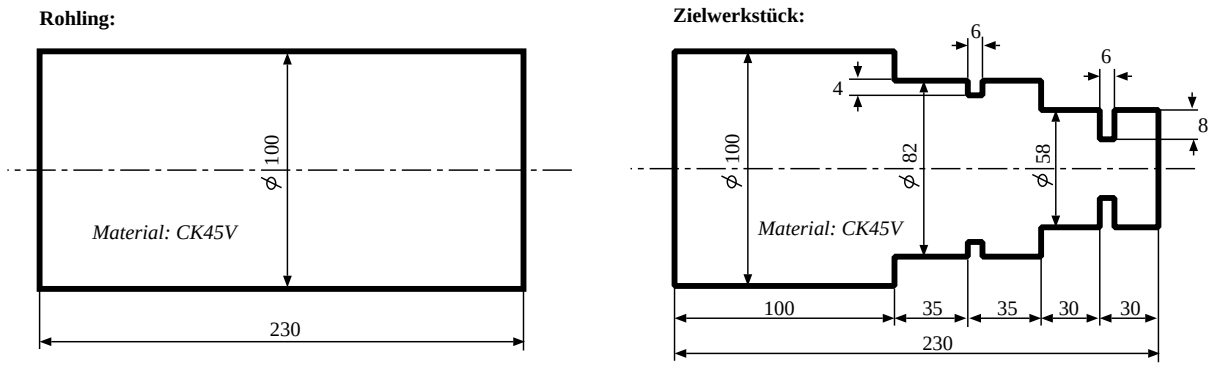

Abbildung 4.2: Beispiel für eine Beschreibung eines Werkstücks und eines Rohlings auf der konkreten Ebene

1). Danach wird ein Werkzeug ausgewählt (Schritt 2), und es werden zwei Bearbeitungsbereich des Zielwerkstückes in zwei Schnittoperationen gefertigt (Schritt 3 und 4). Dann wird ein Werkzeug ausgewählt (Schritt 5), welches verwendet werden kann um die Nuten zu fertigen (Schritt 6 und 7). Schließlich wird das Werkstück aus der Einspannung gelöst (Schritt 8). Die beiden Abbildungen 4.2 und 4.3 zusammen stellen also einen *konkreten Fall* dar.

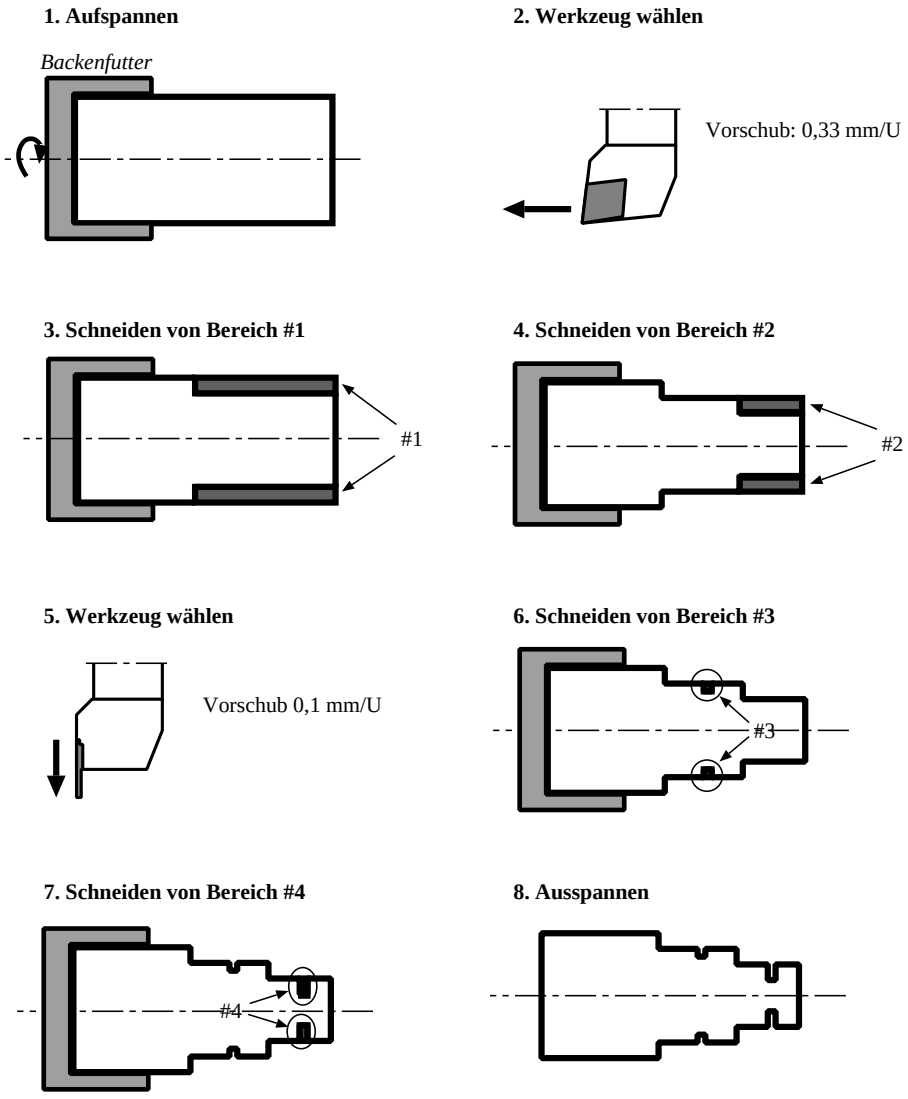

Abbildung 4.3: Beschreibung einer Lösung für die Fertigungsaufgabe aus Abbildung 4.2 auf der konkreten Ebene

- Rohling
  - Zylindrische Form.
  - Mittlere Größe.
  - Geringe Härte.
- Werkstück
  - Großflächige Bearbeitung auf der rechten Seite.
  - Kleinere Formelemente auf der rechten Seite.
  - Keine Bearbeitung auf der linken Seite.
  - Keine Bearbeitung an den Stirnseiten.

Abbildung 4.4: Beispiel für die Beschreibung eines Werkstückes und eines Rohlings auf der abstrakten Ebene

Die Aufgabe der Fallabstraktion besteht nun darin, diesen konkreten Fall in einen *abstrakten Fall* zu transformieren. Setzen wir die zuvor skizzierte abstrakte Problemlösedomäne voraus, so kann sich ein entsprechender abstrakter Fall wie z. B. in den Abbildungen 4.4 und 4.5 dargestellt, ergeben.
Abbildung 4.4 zeigt eine Abstraktion des Problems aus Abbildung 4.2. Man erkennt, daß die genauen Spezifikationen der Geometrie des Werkstückes hier ignoriert werden. Wir verwenden lediglich bestimmte Lageinformationen (z. B. rechte und linke Seite), die für die Lösung des Problems wesentlich sind. Auch die in Abbildung 4.5 dargestellte abstrakte Lösung ist eine Abstraktion der konkreten Lösung aus Abbildung 4.3. Diese abstrakte Lösung besteht hier lediglich aus vier Schritten. Von der konkreten Wahl des Werkzeuges und der genauen Angabe der Bearbeitungsbereiche wird abstrahiert.

- Lösung:
  1. Werkstück auf der linken Seite einspannen.
  2. Großflächige Bearbeitungen auf der rechten Seite des Werkstücks durchführen.
  3. Kleinere Formelemente auf der rechten Seite des Werkstücks bearbeiten.
  4. Werkstück ausspannen.

Abbildung 4.5: Lösung für die Fertigungsaufgabe aus Abbildung 4.4 auf der abstrakten Ebene

Wiederverwendung des abstrakten Falls

Wir wollen nun davon ausgehen, daß der zuvor gezeigte abstrakte Fall zur Wiederverwendung zur Verfügung steht. Als Beispiel sollen nun die beiden in Abbildung 4.6 gezeigten neuen konkreten Probleme p_2 und p_3 betrachten werden. Man kann erkennen, daß sich das Problem p_2 ebenfalls so abstrahieren läßt, daß die in Abbildung 4.4 gezeigte abstrakte Problembeschreibung entsteht. Auch kommen auf der rechten Seite des Werkstcks großflächige Bearbeitungen sowie kleinere Formelemente vor. Obwohl das Problem p_2 vom dem in Abbildung 4.2 dargestellten Problem auf der konkreten Ebene sehr verschieden ist – sie weisen z. B. vollkommen verschiedene Abmessungen auf – sind sie auf dieser abstrakten Ebene identisch. Dies bedeuted, daß die in Abbildung 4.5 gezeigte abstrakte Lösung auch zur Lösung des neuen Problems p_2 verwendet werden kann. Die Aufgabe der Lösungsverfeinerung besteht darin, die entsprechenden Details, also z. B. die Wahl der Werkzeuge und die genau Parametrierung der Schnittoperationen, zu generieren.
Betrachtet man nun im Gegensatz dazu das Problem p_3 aus Abbildung 4.6, so kann man feststellen, daß der zuvor vorausgesetzte abstrakte Fall zur Problemlösung nicht wieder-

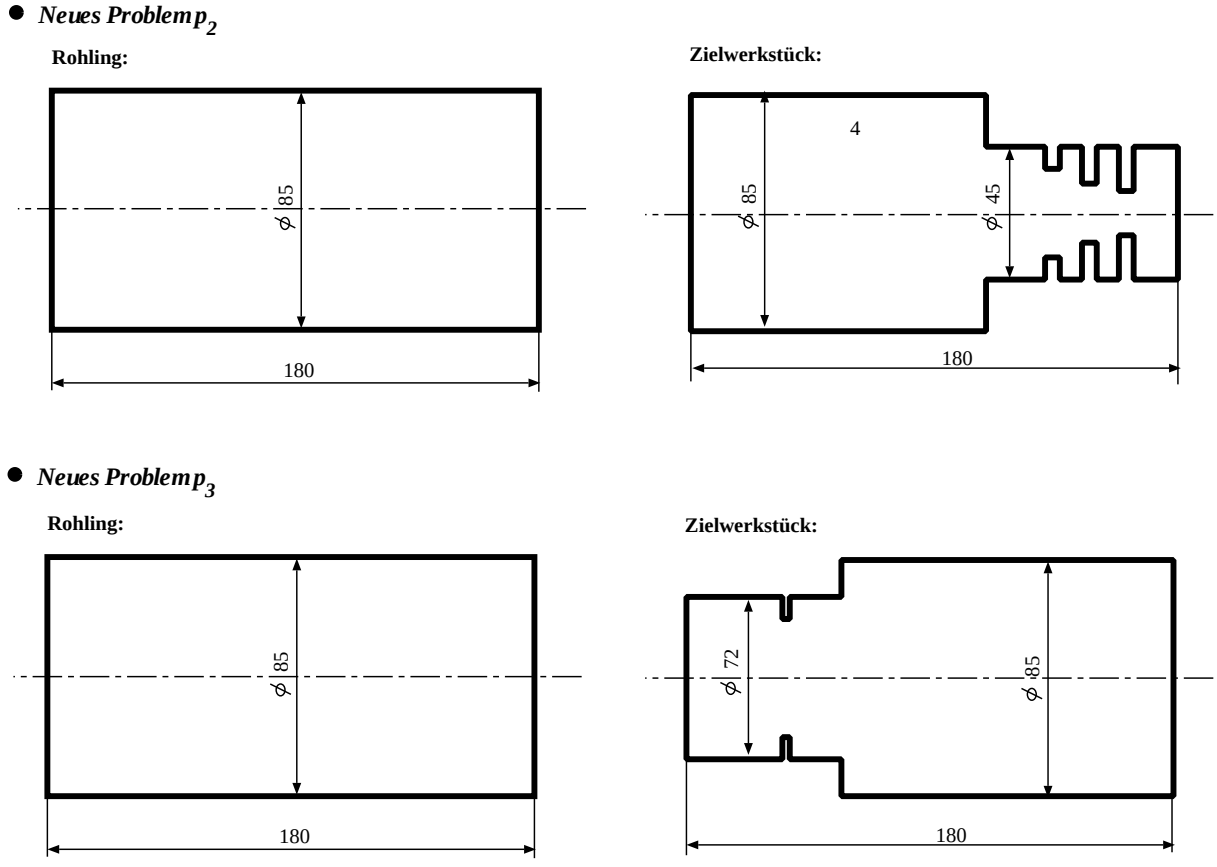

Abbildung 4.6: Zwei neue Probleme auf der konkreten Ebene

verwendet werden kann, weil das Problem p_3 eine Bearbeitung auf der linken Seite des Werkstückes erfordert. Der abstrakte Fall setzt jedoch die Bearbeitung auf der rechten Seite voraus und verwendet die linke Seite zur Aufspannung. Folglich kann die abstrakte Lösung aus Abbildung 4.5 nicht zu einer Lösung von p_3 verfeinert werden.

4.2 Erklärungsbasiertes Generalisieren abstrakter Fälle

Im folgenden soll der zuvor eingeführte Ansatz zur Wiederverwendung von Fällen auf verschiedenen Abstraktionsebenen um einen erklärungsbasierten Ansatz erweitert werden. Während wir bislang Fälle auf einer Abstraktionsebene direkt übernommen haben, ohne die im abstrakten Fall vorliegende Lösung zu modifizieren, wollen wir nun abstrakte Fälle zusätzlich mit erklärungsbasiertem Lernen generalisieren.

4.2.1 Generalisierte Fälle

Während ein Fall ein einzelnes Problem zusammen mit einer einzelnen Lösung darstellt, wollen wir mit einem *generalisierten Fall* eine Menge von Problemen mit zugehörigen Lösungen beschreiben. Hierzu werden die folgenden Begriffe eingeführt:

Begriffsbestimmung 4.5 (Problemklasse) Eine Problemklasse ist eine *Menge von Problemen*, die durch eine verallgemeinerte Beschreibung von Initial- und Zielzustand sowie durch eine Menge von Bedingungen (Constraints) an den Initial- und den Zielzustand beschrieben wird.

Verallgemeinerte Zustandsbeschreibungen werden durch essentielle Satzformen (Bezeichnung 2.1) gebildet, wodurch Variablen in die Problembeschreibung eingeführt werden. Die Constraints bestehen aus Atomformeln der Problemlösedomäne und beschreiben Einschränkungen an die Belegung dieser Variablen. Hierdurch können auch Variablen miteinander in Beziehung gesetzt werden. Eine Problemklasse umfaßt genau die Probleme, deren Initial- und Zielzustand die verallgemeinerte Zustandsbeschreibungen so instantiieren, daß die Constraints erfüllt sind (formale Definition siehe 6.2 und 6.3).

Begriffsbestimmung 4.6 (Lösungsklasse) Eine Lösungsklasse ist eine *Menge von Lösungen (Plänen)*, die durch eine Folge von nur teilweise instantiierten parametrisierten Operatoren beschrieben ist.

Da die Operatoren nur teilweise instantiiert sind, können auch Variablen vorkommen. Die Lösungsklasse beschreibt genau die Menge von Lösungen, die sich durch Instantiierung der Variablen bilden lassen.

Begriffsbestimmung 4.7 (Generalisierter Fall) Ein generalisierter Fall ist eine *Problemklasse* zusammen mit einer *Lösungsklasse*.

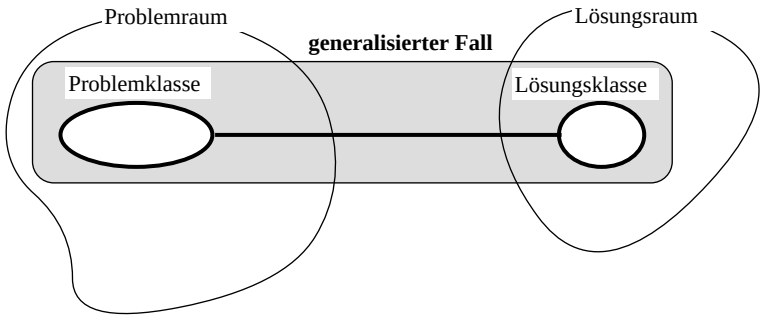

Abbildung 4.7: Schematische Darstellung eines generalisierten Falls

Generalisierte Fälle stellen somit einen Zusammenhang zwischen einer Problemklasse und einer Lösungsklasse her. Darüber hinaus können sie aber auch einem Problem der Problemklasse ein Lösung der Lösungsklasse direkt zuordnen (siehe Abbildung 4.7). Dies ist dann der Fall, wenn die Variablen in der Lösungsklasse vollständig in der Menge der Variablen, die in der Problemklasse vorkommen, enthalten sind. In diesem Fall ergibt sich durch eine konkrete Instantiierung einer Problemklasse durch ein Problem eine vollständige Belegung der Variablen der Lösungsklasse und somit die Bestimmung einer eindeutigen Lösung.

Bemerkung 4.2 Ein generalisierter Fall ist für ein größeres Spektrum von Problemen direkt wiederverwendbar als ein spezieller Fall. Hierbei ist das Spektrum dieser Probleme ebenso wie die Art der zugehörigen Lösungen explizit im Fall repräsentiert. Der generalisierte Fall stellt somit ein Lösungsschema dar.

Abbildung 4.8 zeigt schematisch die Vorgehensweise bei der Wiederverwendung generalisierter Fälle. Der Wiederverwendungsprozeß kann hierbei in die beiden Teilprozesse *Fallgeneralisierung* und *Fallspezialisierung* zerlegt werden.

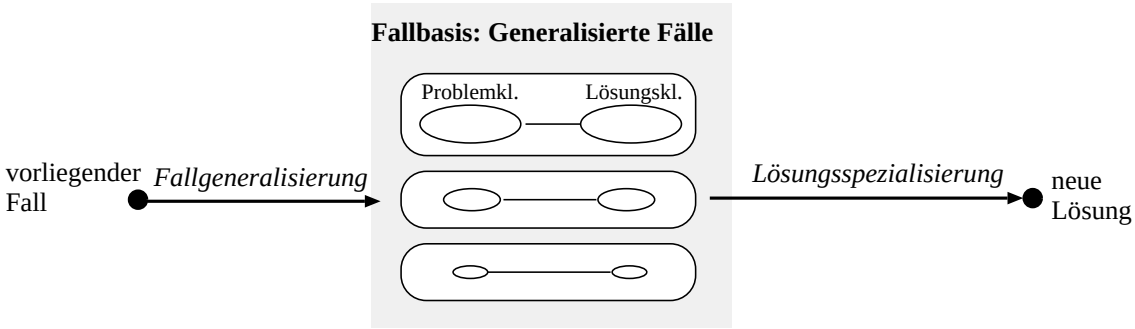

Abbildung 4.8: Wiederverwendung generalisierter Fälle

4.2.2 Die Vorgehensweise

Fallgeneralisierung

Begriffsbestimmung 4.8 (Fallgeneralisierung) Fallgeneralisierung ist der Prozeß der Transformation eines Falls in einen generalisierten Fall unter Berücksichtigung der folgenden zwei Bedingungen:

(i) Problem und Lösung des speziellen Falls müssen innerhalb der Problem- und Lösungsklassen des generalisierten Falls liegen.
(ii) Der generalisierte Fall muß korrekt sein in dem Sinne, daß die Lösung, die der generalisierte Fall einem Problem aus der Problemklasse zuordnet, eine korrekte Lösung ist.[1]

Erklärungsbasiertes Lernen kann zur Fallgeneralisierung eingesetzt werden. Im Detail werden wir das in den Abschnitten 6.2 bis 6.5 erörtern. Die grundsätzliche Idee ist dabei die folgende: Aus dem Zielzustand des Problems im Fall wird ein Zielkonzept dadurch definiert, daß alle essentiellen Sätze durch die entsprechenden essentiellen Satzformen ersetzt werden. Hierdurch ergibt sich eine verallgemeinerte Zustandsbeschreibung des Zielzustands. Aus dem Initialzustand des Problems wird ein Trainingsbeispiel konstruiert. Ist dies gegeben, so kann durch Anwendung der Operatoren und der universellen Implikationen der Problemlösedomäne nachgewiesen werden, daß es einen Plan gibt, der ausgehend vom Initialzustand, zu einem Zustand führt, den das Zielkonzept (verallgemeinerte Zustandsbeschreibung) umfaßt. Dieser Nachweis ist ein spezieller Inferenzprozeß, wie er durch das erklärungsbasierte Lernen betrachtet wird. Als Domänentheorie fungiert die Problemlösedomäne. Als Ergebnis liefert das erklärungsbasierte Lernen hierbei eine operationale Konzeptspezialisierung, die zu einem generalisierter Fall umformuliert werden kann. Die operationale Konzeptspezialisierung beinhaltet dabei eine Menge von Bedingungen, die zusichern, daß das Zielkonzept erfüllt ist. Diese Bedingungen werden zu den Constraints der Problemklasse.

Bemerkung 4.3 Mit einem Operationalitätskriterium läßt sich außerdem der Grad der Allgemeinheit der gelernten Konzeptspezialisierung und somit des generalisierten Falls steuern. Hierdurch besteht die Möglichkeit, verschieden große Problem- und Lösungsklassen zu erzeugen (siehe auch Abbildung 4.8).

Ebenso wie die Fallabstraktion, kann die Fallgeneralisierung in einem Lernschritt vor der eigentlichen Problemlösung durchgeführt werden. Die so gelernten generalisierten Fälle stehen dann für eine spätere Problemlösung bereit und können in einer Fallbasis gespeichert werden.

[1] Wenn eine Lösung nicht eindeutig durch den generalisierten Fall bestimmt ist, d. h. wenn die Variablen, die in der Lösungsklasse vorkommen, die nicht durch die Problemklasse gebunden werden, so muß jede beliebige Instantiierung dieser Variablen eine korrekte Problemlösung darstellen.

Lösungsspezialisierung

Um ein neues Problem zu lösen, muß ein generalisierter Fall aus der Fallbasis gefunden werden, der zur Lösung des neuen Problemes wiederverwendet werden kann.

Begriffsbestimmung 4.9 (Wiederverwendbarkeit eines generalisierten Falls) Ein generalisierter Fall ist genau dann zur Lösung eines Problems wiederverwendbar, wenn das Problem innerhalb der im generalisierten Fall enthaltenen Problemklasse liegt.

Ist ein generalisierter Fall zur Lösung eines Problems wiederverwendbar, so ist sichergestellt, daß die durch den generalisierten Fall zugeordnete Lösung eine korrekte Lösung ist. Zum Nachweis der Wiederverwendbarkeit muß hierbei jedoch überprüft werden, ob die Constraints, die in der Problemklasse vorkommen, erfüllt sind. Ein solcher Test muß eine Belegung der vorkommenden Variablen bestimmen, so daß alle Constraints erfüllt sind (siehe Abschnitt 6.7). Hierdurch ist die Lösung des Problems direkt determiniert. Das Überprüfen der Wiederverwendbarkeit eines generalisierten Falls und das Bestimmen einer speziellen Lösung sind also direkt voneinander abhängig. Dieses fassen wir daher unter dem Begriff der Lösungsspezialisierung zusammen.

Begriffsbestimmung 4.10 (Lösungsspezialisierung) Lösungsspezialisierung ist der Prozeß, der für ein gegebenes Problem und einen gegebenen generalisierten Fall nachweist, ob der generalisierte Fall zur Lösung des Problems wiederverwendbar ist gegebenenfalls eine Lösung aus der Lösungsklasse bestimmt.

4.2.3 Integration von Abstraktion und Generalisierung

Um eine möglichst flexible Wiederverwendung von Fällen zu ermöglichen, wollen wir nun die beiden Ansätze zur Wiederverwendung abstrakter und generalisierter Fälle kombinieren.
Hierzu werden wir *generalisierte Fälle auf verschiedenen Abstraktionsebenen* bilden. Die Abstraktionsebene gibt die Ebene vor, auf der generalisierte Fälle gebildet werden können. Wie bei (nicht-generalisierten) Fällen bestimmt die Abstraktionsebene, welche Merkmale der Domäne überhaupt betrachtet werden und folglich in der generalisierten Problemstellung und Lösung vorkommen können. Wir sprechen in diesem Fall von *generalisierten abstrakten Fällen.*

Begriffsbestimmung 4.11 (Generalisierter abstrakter Fall) Ein generalisierter abstrakter Fall ist ein generalisierter Fall, der auf einer Abstraktionsebene angesiedelt ist, die oberhalb der konkreten Ebene liegt.

In Ergänzung von Abbildung 4.1 und 4.8 zeigt Abbildung 4.9 die Vorgegensweise bei der Wiederverwendung generalisierter abstrakter Fälle. Hierbei wird zunächst jeder konkrete Fall in abstrakte Fälle auf verschiedenen Abstraktionsebenen transformiert. Danach wird jeder abstrakte Fall generalisiert, woraus sich dann generalisierte abstrakte Fälle ergeben. Die generalisierten abstrakten Fällen können dann in der Fallbasis zur späteren Wiederverwendung gespeichert werden. Um ein neues Problem zu lösen, muß aus dieser Fallbasis ein

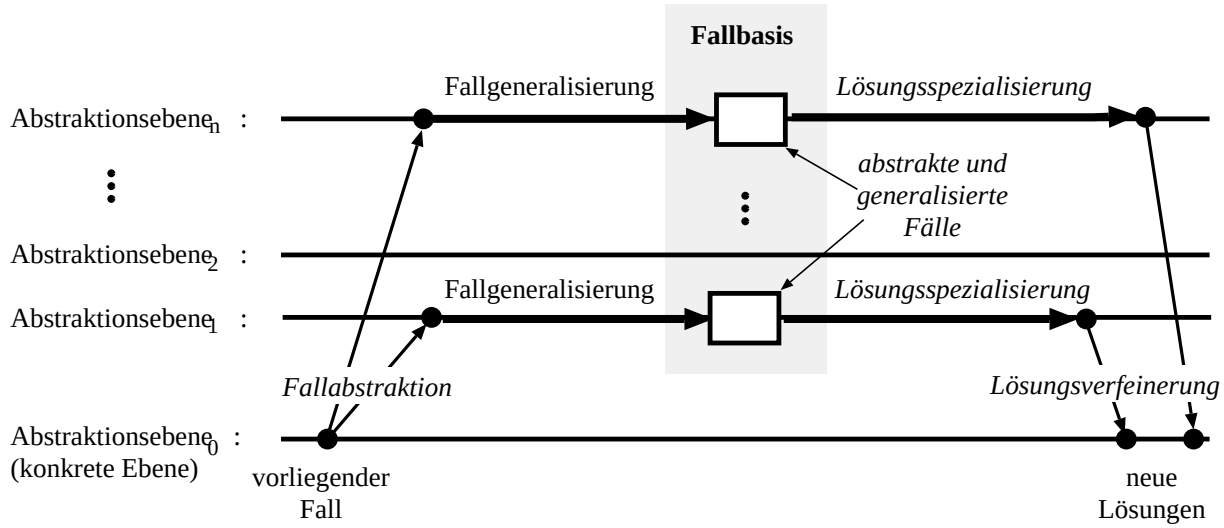

Abbildung 4.9: Wiederverwendung generalisierter abstrakter Fälle

geeigneter generalisierter abstrakter Fall ausgewählt werden. Die in diesem Fall enthaltene abstrakte Lösungsklasse muß zunächst zu einer spezifischen abstrakten Lösung spezialisiert werden. Da die vorliegenden generalisierten Fälle auf einer höheren Abtraktionsebene liegen, entstehen durch die Lösungsspezialisierung zwar spezielle Fälle, jedoch liegen diese auf einer Abstraktionsebene oberhalb der konkreten Ebene. Durch die Lösungsspezialisierung entstehen somit wieder abstrakte Fälle im Sinne der Begriffsbestimmung 4.2. Zur Bildung einer konkreten Lösung ist folglich noch eine Lösungsverfeinerung erforderlich.

4.2.4 Ein Beispiel für die Fallgeneralisierung

An dieser Stelle wollen wir das Beispiel aus Abschnitt 4.1.4 fortführen. Der in den Abbildungen 4.4 und 4.5 dargestellte abstrakte Fall kann in einem Fallgeneralisierungschritt in einen generalisierten abstrakten Fall transformiert werden. Ein solcher generalisierter abstrakter Fall ist in Abbildung 4.10 dargestellt. In dieser Abbildung sind X_1,X_2 und X_3 Variablen für unterschiedliche Seiten (z. B. linke Seite, mittlere Seite, rechte Seite) des Werkstücks. Durch geeignete Instantiierung dieser Variablen entstehen aus dem generalisierten Fall verschiedene spezifische Probleme mit zugehörigen Lösungen auf der abstrakten Ebene. Diese umfassen beispielsweise auch die Lösung des Problems p_3 aus Abbildung 4.6. Der in Abschnitt 8.3.1 betrachtete abstrakte Fall konnte nicht zur Lösung von p_3 herangezogen werden. Hingegen ist der generalisierte abstrakte Fall zur Lösung von p_3 wiederverwendbar, da die vorkommenden Variablen so instantiiert werden können, daß die Aufspannung auf der rechten Seite und die Bearbeitung auf der linken Seite erfolgt.

- Problemklasse
  - Rohling hat zylindrische Form und geringe Härte.
  - Das Werkstück erfordert großflächige Bearbeitungen auf der Seite X_1.
  - Das Werkstück besitzt kleinere Formelemente auf der Seite X_2.
  - Das Werkstück erfordert keine Bearbeitung auf der Seite X_3.
  - Das Werkstück erfordert keine Bearbeitung an der Stirnseiten.
- Lösungsklasse
  1. Werkstück auf der Seite X_3 einspannen.
  2. Großflächige Bearbeitungen auf der Seite X_1 des Werkstücks durchführen.
  3. Kleinere Formelemente auf der Seite X_2 des Werkstücks bearbeiten.
  4. Werkstück ausspannen.
- Abhängigkeiten
  - Die Seite X_3 ist verschieden von der Seite X_1.
  - Die Seite X_3 ist verschieden von der Seite X_2.
  - Das Werkstück kann an der Seite X_3 eingespannt werden.

Abbildung 4.10: Beispiel für einen generalisierten Fall

4.3 Fallauswahl

Wie bei jedem fallbasierten Ansatz stellt sich auch bei der Wiederverwendung generalisierter abstrakter Fälle die Frage nach der Fallauswahl (engl. Retrieval). Um das Ziel einer effizienten Problemlösung zu erreichen, müssen hierbei zwei wichtige Faktoren berücksichtigt werden:

- Der gefundene Fall muß mit möglichst geringem Aufwand zu einer Lösung des aktuellen Problems führen.
- Der Aufwand zum Auffinden eines wiederverwendbaren Falls muß möglichst gering sein.

4.3.1 Aufwand zur Lösungsverfeinerung

Bei der Wiederverwendung von Fällen auf verschiedenen Abstraktionsebenen spielt die Abstraktionsebene, auf der die Wiederverwendung stattfindet, eine entscheidende Rolle. Sie hat insbesondere einen Einfluß darauf, wieviel Aufwand zur Wiederverwendung erforderlich ist. Die Verfeinerung einer Lösung auf einer hohen Abstraktionsebene erfordert mehr Aufwand als die Verfeinerung einer Lösung auf einer niedrigeren Abstraktionsebene (siehe Abschnitt 3.3.4). Aufgrund dieser Betrachtung wird offensichtlich, daß man Fälle

immer auf einer möglichst niedrigen Abstraktionsebene wiederverwenden sollte, um die Lösungsverfeinerung möglichst effizient zu gestalten.

4.3.2 Aufwand durch die Lösungsspezialisierung

Der Aufwand bei der Lösungsspezialisierung besteht im wesentlichen im Überprüfen der Constraints der Problemklasse des generalisierten Falls. Wie wir bereits in Abschnitt 3.2.6 erörtert haben, hängen die Kosten zum Nachweis der Gültigkeit einer Konzeptspezialisierung mit dessen Generalität zusammen. Dies kann beim erklärungsbasierten Lernen durch das Operationalitätskriterium gesteuert werden. Je genereller eine Konzeptspezialisierung ist, desto größer ist der Aufwand, der beim Nachweis seiner Gültigkeit geleistet werden muß. Diese Überlegung betrifft in gleicher Weise den Aufwand für die Lösungsspezialisierung eines generalisierten Falls. Der Aufwand ist hier umso geringer, je spezieller der generalisierte Fall ist.
Das Operationalitätskriterium ist aber nur ein Faktor, der den Aufwand der Lösungsspezialisierung bestimmt. Der zweite Faktor ist auch hier die Höhe der Abstraktionsebene, auf der die Lösungsspezialisierung für einen generalisierten abstrakten Fall durchgeführt wird. Hierbei ist zu bedenken, daß auf höheren Abstraktionsebenen weniger Details von Problemstellung und Lösung betrachtet werden, als auf niedrigeren Abstraktionsebenen. Daher kommt ein generalisierter abstrakter Fall auf einer hohen Abstraktionsebene in der Regel mit weniger Constraints aus als ein generalisierter abstrakter Fall, der auf einer niedrigeren Abstraktionsebene angesiedelt ist. Folglich nimmt der Aufwand zur Lösungsspezialisierung mit steigendem Abstraktionsgrad ab, da eine geringere Anzahl von Constraints überprüft werden muß. Aufgrund dieser Betrachtung wird jedoch offensichtlich, daß Fälle immer auf einer möglichst hohen Abstraktionsebene wiederverwendt werden sollten, um die Lösungsspezialisierung möglichst effizient zu gestalten.

4.3.3 Gesamter Wiederverwendungsaufwand

Der gesamte Wiederverwendungsaufwand ergibt sich aus der Summe der Aufwände zur Lösungsverfeinerung und Lösungsspezialisierung. Wie wir gesehen haben, verhalten sich, in Abhängigkeit vom Abstraktionsgrad, die Aufwände zur Lösungsverfeinerung und Lösungsspezialisierung gegensätzlich. Dies ist in Abbildung 4.11 noch einmal beispielhaft dargestellt. Wie man sieht, kann für die der Wiederverwendung generalisierter abstrakter Fälle keine eindeutige Präferenz für die niedrigste oder höchste Abstraktionsebene gegeben werden. Vielmehr muß eine mittlere Abstraktionsebene gefunden werden, bei der der Gesamtaufwand minimal wird.
Den Ansatz, den wir hier verfolgen, basiert auf einer Abschätzung des Wiederverwendungsaufwandes für jeden generalisierten abstrakten Fall aufgrund eines Kostenmodells. Dies werden wir ausführlich in Abschnitt 7.3 behandeln.

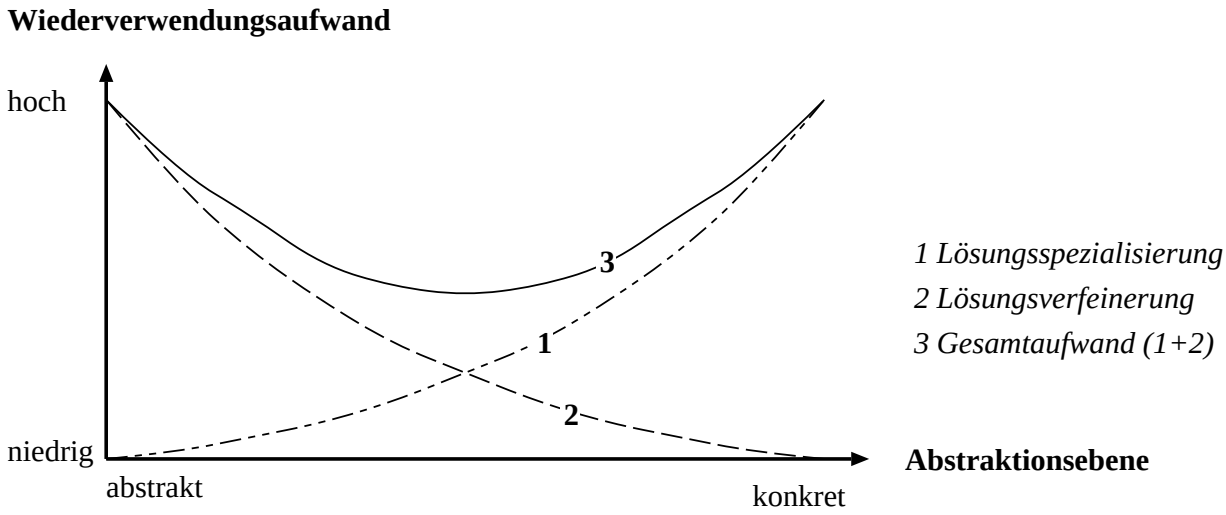

Abbildung 4.11: Aufwand bei der Wiederverwendung generalisierter abstrakter Fälle in Abhängigkeit der Abstraktionsebene der Wiederverwendung

4.3.4 Retrievalaufwand und Wiederverwendungsaufwand

Neben dem Auffinden eines möglichst guten generalisierten abstrakten Falls ist es ebenso wichtig, diesen in möglichst kurzer Zeit zu finden. Dies erfordert u. U. einen Kompromiß zwischen dem zu leistenden Retrievalaufwand und dem Wiederverwendungsaufwand für den gefundenen Fall. Hierbei stellt sich die Frage, inwieweit es sich lohnt, zusätzliche Retrievalzeit zu investieren, um einen Fall zu finden, der effizienter wiederverwendet werden kann als der Fall, den das Retrieval in kürzerer Zeit findet. Dieser Zusammenhang wurde bereits von Manuela Veloso im System PRODIGY/ANALOGY beobachtet [Veloso, 1992b; Veloso, 1994b] und ist in Abbildung 4.12 beispielhaft dargestellt.

4.3.5 Indexierung der Fallbasis

Um die Retrievalzeit während des Problemlösens möglichst kurz zu halten, ist es erforderlich, die Gesamtmenge der gelernten generalisierten abstrakten Fälle in einer geeigneten Art und Weise zu organisieren. Ein in der Informatik oft gebrauchter Ansatz zur Organisation von Datenbeständen besteht in der Verwendung von Bäumen.[2] Wir wollen hier einen Ansatz vorschlagen, bei dem generalisierte abstrakte Fälle in einer sogenannten *Wiederverwendbarkeitshierarchie* angeordnet werden. Dies ist ein n-ärer Baum, bei dem jedem Knoten (innere und äußere) ein generalisierter abstrakter Fall zugeordnet ist. Ein generalisierter abstrakter Fall an einem Knoten dieser Hierarchie ist hierbei für eine größere (oder

[2] Bäume werden auch im fallbasierten Schließen häufig zur Organisation von Fallbasen eingesetzt. kd-Bäume können z. B. zum effizienten Fallretrieval in analytischen Domänen verwendet werden [Wess *et al.*, 1994b].

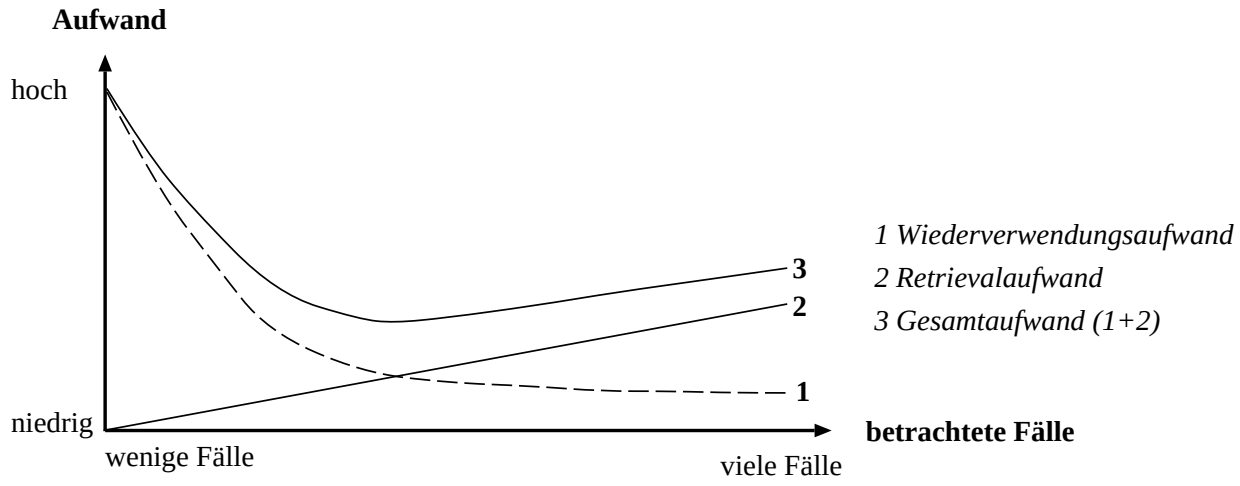

Abbildung 4.12: Abhängigkeit von Retrievalzeit und Wiederverwendungszeit nach Veloso (1992)

gleich große) Menge von konkreten Problemen wiederverwendbar als jeder generalisierte abstrakte Fall an den Nachfolgerknoten (siehe Abbildung 4.13). Die Partialordnung, die dieser Hierarchie zugrundeliegt, kann wie folgt definiert werden.

Begriffsbestimmung 4.12 (Wiederverwendbarkeitsrelation) Ein generalisierter abstrakter Fall GAF_1 ist *flexibler wiederverwendbar* als ein generalisierter abstrakter Fall GAF_2 genau dann wenn für alle Probleme p gilt: falls GAF_1 wiederverwendbar ist für p dann ist auch GAF_2 wiederverwendbar für p. Wir schreiben $GAF_1 \sqsupseteq GAF_2$ und nennen $\sqsupseteq$ Wiederverwendbarkeitsrelation.

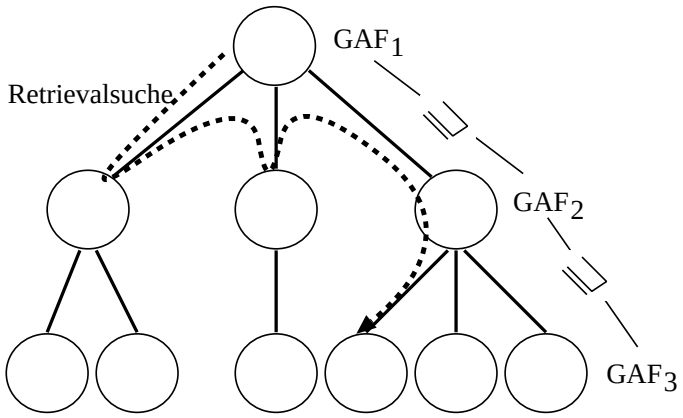

Abbildung 4.13: Ausschnitt aus einer Wiederverwendbarkeitshierarchie

Bemerkung 4.4 Die so definierte $\sqsupseteq$-Relation entspricht der üblichen Definition der Relation *allgemeiner als*, wie sie im Maschinellen Lernen verwendet wird. In unserem Ansatz ist GAF_1 flexibler wiederverwendbar als GAF_2, wenn entweder GAF_1 eine größere Problemklasse aufgrund der Fallgeneralisierung umfaßt als GAF_2, oder wenn GAF_1 abstrakter ist als GAF_2. Daher wollen wir bewußt den Begriff *allgemeiner als* vermeiden, um dem Eindruck entgegenzuwirken, daß die Wiederverwendbarkeit allein durch die Generalisierung bewirkt wird.

Ist ein generalisierter abstrakter Fall *nicht* zur Lösung eines Problems wiederverwendbar, so sind auch alle generalisierten abstrakten Fälle, die unterhalb in der Hierarchie angeordnet sind, nicht zur Lösung des Problems wiederverwendbar. Falls man beim Durchlaufen dieses Baumes auf einen nicht anwendbaren Fall stößt, so kann man alle Fälle im darunterliegenden Teilbaum ignorieren, da sie keinesfalls wiederverwendbar für das aktuelle Problem sein können. Hierdurch kann der Suchaufwand im Vergleich zu einer vollständigen erheblich reduziert werden.
In Abschnitt 7.2 werden wir detailliert beschreiben, wie eine solche Wiederverwendbarkeitshierarchie aufgebaut werden kann.

4.3.6 Ein Beispiel für eine Klassifikationshierarchie

In Fortführung des Beispiels aus Abschnitt 4.2.4 zeigt Abbildung 4.14 eine Wiederverwendbarkeitshierarchie wie sie für die Domäne der Arbeitsplanung entstehen kann. In dieser

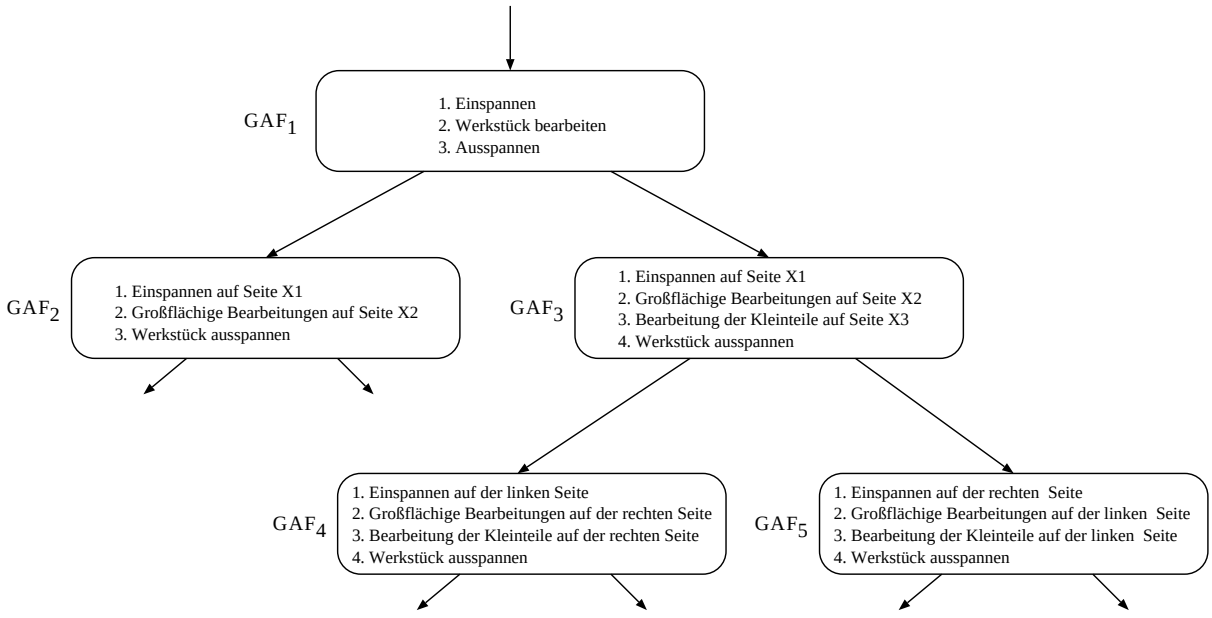

Abbildung 4.14: Ein Beispiel für eine Wiederverwendbarkeitshierarchie

Abbildung sind generalisierte abstrakte Fälle durch die Angabe des jeweiligen abstrakten Plans beschrieben. Ein Fall der höher in dieser Hierarchie angeordnet ist, ist hierbei flexibler wiederverwendbar als ein Fall, der tiefer liegt. So ist z. B. der Fall GAF_3 flexibler wiederverwendbar als der Fall GAF_5, da GAF_5 für die Bearbeitung auf der rechten Seite spezialisiert ist, wo hingegen GAF_3 eine Lösung für beliebige Bearbeitungsseiten liefert. Wie bereits in Abschnitt 4.2.4 erörtert wurde, ist GAF_3 zur Lösung der Probleme p_2 und p_3 aus Abbildung 4.6 geeignet, GAF_5 jedoch nur zur Lösung von p_2.

4.4 Die PARIS-Architektur

Auf der Grundlage der bislang skizzierten Ansätze zur flexiblen Wiederverwendung von Fällen wollen wir nun die Gesamtarchitektur eines fallbasierten Problemlösers aus den einzelnen Komponenten aufbauen. Abbildung 4.15 zeigt diese Architektur und das Zusammenwirken der einzelnen Komponenten. In dieser Architektur unterscheiden wir eine *Lernphase* und eine *Problemlösephase*.

4.4.1 Die Lernphase

Die Lernphase führt die Vorverarbeitung der konkreten Fälle durch. Aus jedem Fall werden durch einen Fallabstraktionsprozeß eine Menge abstrakter Fälle gebildet, die auf verschiedenen Abstraktionsebenen liegen können. Die hierzu erforderlichen Abstraktionsebenen werden dabei durch ein Domänenmodell vorgegeben. Daran schließt sich der Prozeß der Fallgeneralisierung an. Aus jedem zuvor gebildeten abstrakten Fall wird nun ein gene-

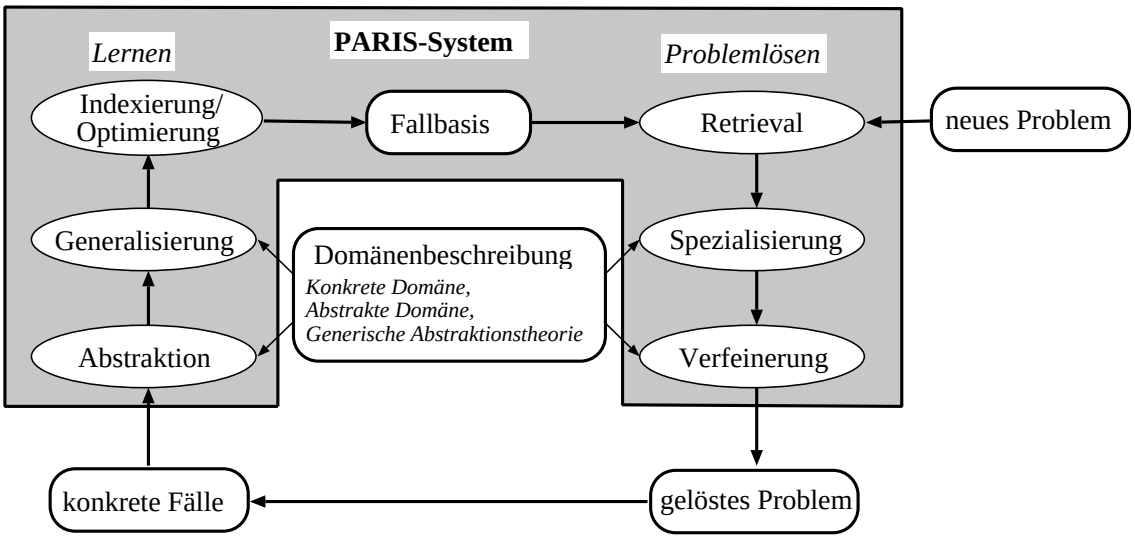

Abbildung 4.15: Die Architektur des PARIS Systems

ralisierter abstrakter Fall erzeugt. In der Evaluations und Indexierungskomponente wird sodann der Wiederverwendungsaufwand jedes generalisierten abstrakten Falls bestimmt. Die Fälle werden in die Fallbasis hinzugenommen und die Wiederverwendbarkeitshierarchie wird als Indexstruktur entsprechend angepaßt. Wir sprechen hierbei von einer Lernphase, da das in den Fällen vorhandene Wissen „kompiliert“ wird. Hierbei werden im Sinne des maschinellen Lernens abstrakte generalisierte Fälle gelernt. Außerdem wird die Wiederverwendbarkeitshierarchie zur Indexierung der Fälle in der Fallbasis gelernt.

4.4.2 Die Problemlösephase

Die Problemlösephase wird aktiviert, wenn ein neues Problem zur Lösung ansteht. Zunächst wird ein generalisierter abstrakter Fall aus der Fallbasis ausgewählt, dessen Wiederverwendungsaufwand als gering eingeschätzt wird. Ist ein wiederverwendbarer generalisierter abstrakter Fall gefunden, so wird die darin enthaltene Lösung zunächst spezialisiert, so daß sich eine instantiierte abstrakte Lösung ergibt. Diese abstrakte Lösung wird sodann in einem Verfeinerungsschritt in eine konkrete Lösung des Problems überführt. Das Problem zusammen mit der so gefundenen Lösung steht nun als neuer Fall zur Verfügung und kann durch die Anwendung der Lernphase als neue Erfahrung der Fallbasis zugefügt werden. Hierdurch ergibt sich insgesamt ein geschlossener Zyklus.

Kapitel 5

Fallabstraktion und Verfeinerung

In diesem Kapitel wird der in dieser Arbeit entwickelte Ansatz zur flexiblen Wiederverwendung von Fällen auf verschiedenen Abstraktionsebenen im Detail dargestellt. Wir beschreiben somit die in Abbildung 5.1 grau hinterlegten Komponenten der PARIS Architektur. Hierzu wird in Abschnitt 5.1 zunächst ein Modell der Fallabstraktion zum Problemlösen eingeführt. In Abschnitt 5.2 wird dann ein Algorithmus zur Realisierung des Fallabstraktionsprozesses entwickelt und als vollständig und korrekt in Bezug auf das zuvor eingeführte Modell nachgewiesen. Ein Algorithmus zur Lösungsverfeinerung wird in Abschnitt 5.4 präsentiert. Zur Verdeutlichung der dargestellten Begriffe und Algorithmen verwenden wir in diesem Kapitel nur sehr einfache Beispiele. Für komplexere Beispiele aus dem Bereich der Arbeitsplanung wird daher auf Kapitel 8 verwiesen.

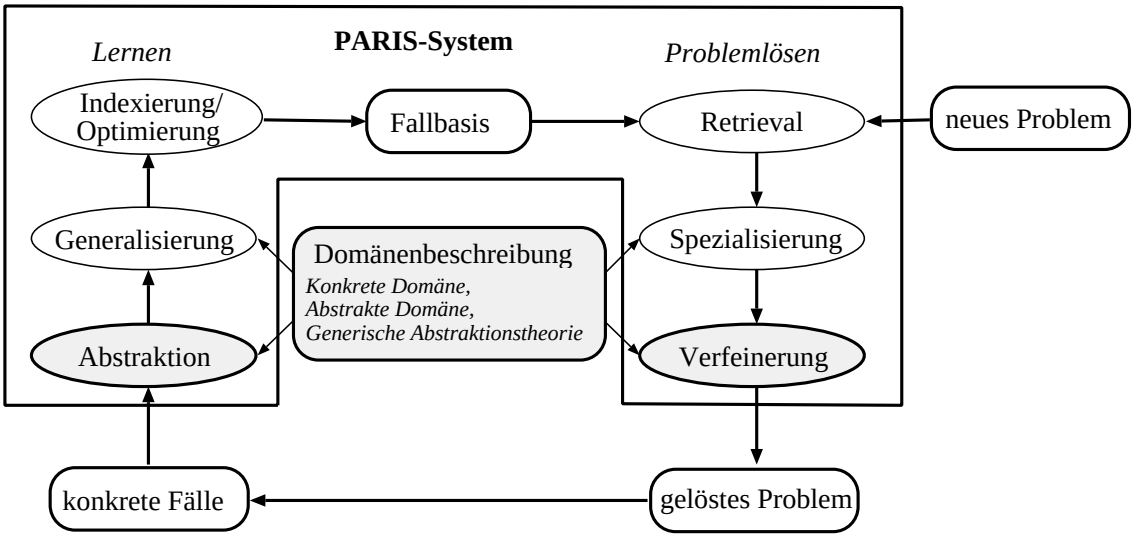

Abbildung 5.1: Die Architektur des PARIS Systems

5.1 Modell der Fallabstraktion

5.1.1 Abstraktion durch Wechsel der Repräsentationssprache

Die Aufgabe der Abstraktion ist die Reduktion des Detailliertheitsgrades einer Beschreibung [Michalski & Kodratoff, 1990; Giunchiglia & Walsh, 1992; Michalski, 1994]. Die Wahl der Abstraktionsebenen ist dabei von vielen Faktoren abhängig, die wir in Kapitel 8 Abschnitt 8.1 genauer untersuchen werden. Oft wird ihre Wahl jedoch dadurch bestimmt, daß Wissen nur auf ganz bestimmten Abstraktionsebenen vorliegt.

Wie bereits in Abschnitt 3.3.5 erörtert wurde, werden die Abstraktionsebenen im Problemlösen in der Regel durch Elimination bestimmter essentieller Sätze aus der Zustandsbeschreibung realisiert. Diese Sichtweise von Abstraktion ist jedoch sehr problematisch. In [Bergmann & Wilke, 1995a] wurden die Beschränkungen von Abstraktion durch Elimination essentieller Sätze aufgezeigt und anhand einer Reihe von Beispielen analysiert. Im folgenden soll ein einfaches Beispiel ein typisches Problem dieses klassischen Abstraktionsansatzes aufzeigen.

Beispiel 5.1 In vielen Domänen ist es erforderlich, bestimmte Ereignisse zu zählen. Hierzu stelle man sich einen *Zähloperator* vor, der wie folgt beschrieben ist.

Operator: `erhöhe_wert`
Pre: { `wert(X)` }
Del: { `wert(X)` }
Add: { `wert(x+1)` }

In diesem Beispiel enthält ein Zustand genau einen essentiellen Satz `wert`, in dessen ganzzahligem Argument Ereignisse gezählt werden können. Bei jeder Anwendung dieses Operators wird dieses Argument inkrementiert. Betrachtet man nun z. B. das Problem, genau 8 Ereignisse auszulösen, d. h. `wert(0)` nach `wert(8)` zu transformieren, so ist eine Lösung dieses Problems ein Plan, bestehend aus 8 `erhöhte_wert` Operatoren. Dieses Problem läßt sich jedoch nicht durch Elimination essentieller Sätze vereinfachen, bzw. in Teilprobleme zerlegen. Da alle Informationen (der Zahlenwert X) in einem einzigen essentiellen Satz repräsentiert sind, geht durch das Entfernen dieses Satzes gleich die gesamte Information verloren.

Eine wesentlich allgemeinere Sichtweise auf Abstraktion besteht darin, Abstraktion als einen Wechsel der Repräsentationssprache aufzufassen, wie dies auch unlängst in Literatur vorgeschlagen wird [Michalski & Kodratoff, 1990; Giunchiglia & Walsh, 1992; Michalski, 1994; Holte *et al.*, 1994; Holte *et al.*, 1995]. Hierbei kann eine konkrete Repräsentationsebene z. B. eine quantitative Beschreibung eines Zustandes darstellen und eine abstrakte Ebene eine rein qualtitative Beschreibung. Beide Beschreibungssprachen können hierbei völlig disjunkt sein.

Beispiel 5.2 Wendet man diesen Ansatz auf das zuvor beschriebene Beispiel an, so wird offensichtlich, daß man lediglich eine geeignete qualitative Beschreibung des Zahlenwertes einführen muß, um nützliche Abstraktionen zu ermöglichen. Im einfachsten Fall benötigt man drei verschiedene qualitative Werte `niedrig`, `mittel` und `hoch`, die z. B. als Argument eines qualitativen Wertprädikats `qual_wert` auftreten können. Hierbei kann die Zusammenfassung verschiedener konkreter Ausprägungen wie folgt definiert werden:

$$\begin{aligned}
&\texttt{qual_wert(niedrig)} \Leftrightarrow \texttt{wert(X)} \wedge X \in \{0, \ldots, 3\},\\
&\texttt{qual_wert(mittel)} \Leftrightarrow \texttt{wert(X)} \wedge X \in \{4, \ldots, 7\},\\
&\texttt{qual_wert(hoch)} \Leftrightarrow \texttt{wert(X)} \wedge X \in \{8, \ldots, 11\}.
\end{aligned}$$

Benutzt man eine solche qualitative Repräsentation, so kann das Problem, 8 Ereignisse auszulösen, in die zwei Teilprobleme zerlegt werden, welche daraus bestehen, den Wert zunächst von `niedrig` nach `mittel` zu erhöhen und in einem zweiten Schritt den Wert von `mittel` nach `hoch` zu erhöhen.

5.1.2 Voraussetzung zum Wechsel der Repräsentationsebenen

Um einen Abstraktionsansatz zu realisieren, der es erlaubt die Repräsentationssprache zwischen den Abstraktionsebenen zu wechseln, müssen wir im folgenden die Voraussetzung treffen, daß die Abstraktionsebenen selbst, so wie Regeln zu deren Wechsel, vorgegeben sind. Während ein Wissensingenieur, der die Problemlösedomäne für einen klassischen hierarchischen Problemlöser wie ABSTRIPS [Sacerdoti, 1974] oder ABTWEAK [Yang & Tenenberg, 1990] erstellt, lediglich jedem essentiellen Satz eine Abstraktionsebene zuordnen kann, soll dieser Ansatz die explizite Modellierung geeigneter Abstraktionsebenen durch die Angabe eigenständiger Problemlösedomänen ermöglichen.

Daher wollen wir im folgenden davon ausgehen, daß zwei vollständige Problemlösedomänen gemäß Definition 2.6 vorgegeben sind: eine *konkrete Problemlösedomäne* und eine *abstrakte Problemlösedomäne*.

Definition 5.1 (Konkrete Problemlösedomäne) Eine konkrete Problemlösedomäne ist eine Problemlösedomäne gemäß Definition 2.6, die die konkrete (niedrigste) Abstraktionsebene beim Problemlösen beschreibt.

Die konkrete Problemlösedomäne wird mit $\mathcal{D}_c = (\mathcal{L}_c, \mathcal{E}_c, \mathcal{O}_c, \mathcal{R}_c)$ bezeichnet. Zustände aus der konkreten Problemlösedomäne werden mit s^c bezeichnet und Operatoren mit o^c.

Definition 5.2 (Abstrakte Problemlösedomäne) Eine abstrakte Problemlösedomäne ist eine Problemlösedomäne gemäß Definition 2.6, die die abstrakteren (höheren) Abstraktionsebenen beim Problemlösen beschreibt.

Die abstrakte Problemlösedomäne wird mit $\mathcal{D}_a = (\mathcal{L}_a, \mathcal{E}_a, \mathcal{O}_a, \mathcal{R}_a)$ bezeichnet. Zustände aus der konkreten Problemlösedomäne werden mit s^a bezeichnet und Operatoren mit o^a.

In Beispiel 5.2 entspricht die konkrete Problemlösedomäne der Modellierung mit Hilfe des essentiellen Satzes `wert` und dem zugehörigen `erhöhe_wert` Operator. Die abstrakte Problemlösedomäne enthält die qualitative Beschreibung mit den Werten `tief`, `mittel` und `hoch` sowie einen entsprechenden abstrakten Zähloperator. In Abschnitt 8.2 werden wir eine abstrakte und eine konkrete Problemlösedomäne für den Bereich der Arbeitsplanung rotationssymmetricher Werkstücke detailliert vorstellen.

Um den Zusammenhang zwischen der konkreten und der abstrakten Problemlösedomäne herzustellen, ist es zusätzlich erforderlich, Beziehungen zwischen den konkreten und den abstrakten essentiellen Sätzen anzugeben. Hierzu führen wir den Begriff der generischen Abstraktionstheorie ein.

Definition 5.3 (Generische Abstraktionstheorie) Eine generische Abstraktionstheorie ist eine Menge von Hornklauseln der Form $e_a \leftarrow a_1, \ldots, a_k$. Hierbei ist e_a ein essentieller Satz der abstrakten Problemlösedomäne, d. h. $e_a = E_a\sigma$ für $E_a \in \mathcal{E}_a$ und eine Substitution σ. Die Vorbedingung $a_1, \ldots, a_k$ der Hornklausel besteht aus Atomen der konkreten oder der abstrakten Sprache, d. h. $a_i \in \mathcal{L}_c \cup \mathcal{L}_a$. Die generische Abstraktionstheorie wird im folgenden mit $\mathcal{A}$ bezeichnet.

Beispiel 5.3 In Beispiel 5.2 entspricht die generische Abstraktionstheorie der Definition der qualitativen Werte durch quantitative Werte, wie z. B.
`qual_wert(niedrig)` $\leftarrow$ `wert(X)` $\wedge X \in \{0, \ldots, 3\}$.

5.1.3 Fallabstraktion

Im folgenden wollen wir präzisieren, wann ein abstrakter Fall als Abstraktion eines konkreten Falles bezeichnet werden soll. Hierzu müssen die Zustände und Operatoren des konkreten Falles zu denen des abstrakten Falles in Beziehung gesetzt werden. Die Reduzierung des Detailliertheitsgrades eines Falles bedingt jedoch, daß der abstrakte Fall aus weniger (oder höchstens gleich vielen) Zuständen und Operatoren besteht wie der konkrete Fall. Daher ist es erforderlich, die Zustände und Operatoren des abstrakten Falles zu ausgewählten Zuständen und Operatoren des konkreten Falles in Beziehung zu setzen. Abbildung 5.2 zeigt diese Beziehung auf. Hierbei werden bestimmte Zustände des konkreten Falles zu Zuständen des abstrakten Falles abstrahiert (z. B. wird s_0^c zu s_0^a abstrahiert). Andere konkrete Zustände werden „übersprungen“ (z. B. s_1^c und s_2^c) und kommen auf der abstrakten Ebene nicht mehr vor. Durch dieses „Überspringen“ von Zuständen wird immer auch eine Folge konkreter Operatoren zu einem abstrakten Operator zusammengefaßt (z. B. wird o_1^c, o_2^c, o_3^c zu o_1^a zusammengefaßt).

Bemerkung 5.1 Generell läßt sich jeder konkrete Zustand abstrahieren, also auch die Zustände, die übersprungen werden. Daß bestimmte Zustände auf der abstrakten Ebene (z.B. in Abbildung 5.2) nicht dargestellt sind, bedeutet lediglich, daß sie keine Bedeutung für die Fallabstraktion haben.

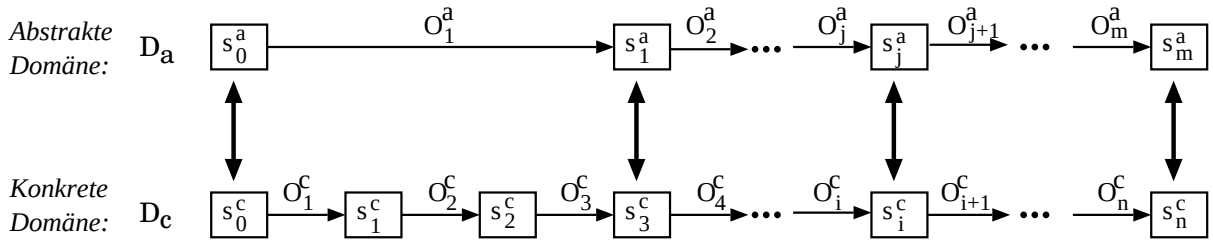

Abbildung 5.2: Idee der Fallabstraktion

Diese Beziehung von Zuständen und Operatoren wird im folgenden in zwei unabhängige Abstraktionsabbildungen zerlegt, die mit *Zustandsabstraktionsabbildung* und die *Sequenzabstraktionsabbildung* bezeichnet werden sollen (vgl. [Bergmann, 1992b]). Die Zustandsabstraktionsabbildung transformiert eine Auswahl von konkreten Zustandsbeschreibungen, die im konkreten Fall vorkommen, in abstrakte Zustandsbeschreibungen, die im abstrakten Fall vorkommen. Die Sequenzabstraktionsabbildung hingegen spezifiziert, welche konkreten Zustände in abstrakte Zustände transformiert werden.

Zustandsabstraktionsabbildung

Definition 5.4 (Zustandsabstraktionsabbildung) Eine Zustandsabstraktionsabbildung $\alpha : \mathcal{S}_c \rightarrow \mathcal{S}_a$ ist eine Abbildung von $\mathcal{S}_c$, der Menge der Zustände der konkreten Domäne, in $\mathcal{S}_a$, der Menge der Zustände der abstrakten Domäne. Ein Zustandsabstraktionsabbildung ist eine totale Funktion.

Bemerkung 5.2 Eine Zustandsabstraktionsabbildung bildet üblicherweise eine Menge unterschiedlicher konkreter Zustände auf einen abstrakten Zustand ab (vgl. auch Beispiel 5.5). Zustände, die dieselbe Abstraktion besitzen, werden hierdurch auf der abstrakten Ebene nicht mehr unterscheidbar (vgl. auch Abbildung 3.7). Somit wird die Menge der konkreten Zustände partitioniert.

Diese Definition der Zustandsabstraktion ist jedoch noch zu allgemein, um den Fallabstraktionsbegriff darauf aufbauend definieren zu können. Daher müssen weitere Einschränkungen für Zustandsabstraktionen gefordert werden, die es erlauben, vorliegendes Domänenwissen über „zulässige“ Abstraktionen zu berücksichtigen. Hierzu führen wir den Begriff der *gerechtfertigten Zustandsabstraktionsabbildung* ein.

Definition 5.5 (Gerechtfertigte Zustandsabstraktionsabbildung) Eine Zustandsabstraktionsabbildung $\alpha : \mathcal{S}_c \rightarrow \mathcal{S}_a$ ist gerechtfertigt durch eine generische Abstraktionstheorie $\mathcal{A}$ wenn die folgenden Bedingungen erfüllt sind:

(i) Für alle $s^c \in \mathcal{S}_c$ gilt: wenn $\phi \in \alpha(s^c)$ dann gilt auch $s^c \cup \mathcal{R}_c \cup \mathcal{A} \vdash \phi$ und

(ii) Für alle $s^c, \tilde{s}^c \in \mathcal{S}_c$ gilt: wenn $\phi \in \alpha(s^c)$ und $\tilde{s}^c \cup \mathcal{R}_c \cup \mathcal{A} \vdash \phi$, dann gilt auch $\phi \in \alpha(\tilde{s}^c)$.

In dieser Definition garantiert die erste Bedingung die „Korrektheit“ der Abstraktion in Bezug auf die generische Abstraktionstheorie. Dies bedeutet, daß sich jeder abstrakte Satz, der bei Anwendung der Zustandsabstraktionsabbildung auf eine konkrete Zustandsbeschreibung im Funktionswert vorkommt, mit Hilfe der generische Abstraktionstheorie aus diesem konkreten Zustand ableiten läßt. Die durch die Abbildung bestimmten Sätze sind also korrekt bezüglich der generischen Abstraktionstheorie. Die zweite Bedingung garantiert zusätzlich eine Art „Vollständigkeit“. Dies bedeutet, daß ein abstrakter Satz, der im Wertebereich einer Zustandsabstraktion vorkommt, als Bild jedes konkreten Zustands vorkommen muß, aus dem sich dieser Satz mit Hilfe der generischen Abstraktionstheorie ableiten läßt.

Beispiel 5.4 Für die generische Abstraktionstheorie, die sich aus Beispiel 5.2 ergibt, würde eine Zustandsabstraktionsabbildung α_1, mit $\alpha_1(\texttt{wert(0)}) = \{\texttt{qual_wert(hoch)}\}$, nicht der Korrektheitsbedingung entsprechen. Ebenso würde eine Zustandsabstraktionsabbildung α_2, mit $\alpha_2(\texttt{wert(0)}) = \{\texttt{qual_wert(niedrig)}\}$ und $\alpha_2(\texttt{wert(1)}) = \{\}$ nicht der Vollständigkeitsbedingung entsprechen. Jedoch ist α_3, mit

$$\alpha_3(\texttt{wert(X)}) = \begin{cases} \{\texttt{qual_wert(niedrig)}\} & \text{falls} \quad X \in \{0, \ldots, 3\} \\ \{\} & \text{sonst} \end{cases}$$

eine gerechtfertigte Zustandsabstraktionsabbildung.

Wie man sieht, ist es nicht unbedingt erforderlich, daß alle abstrakten Sätze, die durch die generische Abstraktionstheorie abgeleitet werden können, auch im abstrakten Zustand vorkommen müssen. Dies eröffnet die Möglichkeit, verschiedene Abstraktionen oder verschiedene Abstraktionsebenen in einer einzigen generischen Abstraktionstheorie zu kodieren. Erst die Zustandsabstraktionsabbildung wählt aus dieser Gesamtheit der Möglichkeiten, die durch die generische Abstraktionstheorie vorgegeben werden, eine bestimmte Zustandsabstraktion aus.

Im folgenden gehen wir immer von gerechtfertigten Zustandsabstraktionsabbildungen aus.

Bemerkung 5.3 Bei einer gegebenen generischen Abstraktionstheorie wird jede gerechtfertigte Zustandsabstraktionsabbildung α durch eine Teilmenge der essentiellen Sätze $\alpha^* \subseteq \mathcal{E}_a$ wie folgt induziert: $\alpha(s^c) := \{\phi \in \alpha^* | s^c \cup \mathcal{R}_c \cup \mathcal{A} \vdash \phi\}$. Dabei legt α^* genau die Menge der essentiellen Sätze fest, die für eine Zustandsabstraktionsabbildung betrachtet werden sollen.

Sequenzabstraktionsabbildung

Während die Zustandsabstraktionsabbildung die Beschreibung der einzelnen Zustände abstrahiert, bestimmt die Sequenzabstraktionsabbildung welche abstrakten Zustände zu welchen konkreten Zuständen in Beziehung gesetzt werden.

Definition 5.6 (Sequenzabstraktionsabbildung) Eine Sequenzabstraktionsabbildung β ist eine Abbildung $\beta : \mathbb{N} \to \mathbb{N}$ die die folgenden Bedingungen erfüllt:

(i) $\beta(0) = 0$ und $\beta(m) = n$.
(ii) $\beta(u) < \beta(v)$ genau dann wenn $u < v$.

Eine Sequenzabstraktionsabbildung setzt eine abstrakte Zustandssequenz $\langle s_0^a, \ldots, s_m^a \rangle$ zu einer konkreten Zustandssequenz $\langle s_0^c, \ldots, s_n^c \rangle$ durch die Abbildung der Indizes $j \in \{1, \ldots, m\}$ der abstrakten Zustände s_j^a in die Indizes $i \in \{1, \ldots, n\}$ der konkreten Zustände s_i^c in Beziehung. Die erste Bedingung in dieser Definition besagt, daß der Initialzustand des abstrakten Falles mit dem Initialzustand des konkreten Falles und der Zielzustand des abstrakten Falles mit dem Zielzustand des konkreten Falles korrespondieren muß. Hierdurch wird sichergestellt, daß der abstrakte Fall eine Abstraktion des gesamten konkreten Falles ist, und nicht nur einen Ausschnitt daraus darstellt. Die zweite Bedingung besagt, daß die Ordnung der Zustände, so wie sie durch den konkreten Fall vorgegeben ist, für den abstrakten Fall erhalten werden muß.

Definition der Fallabstraktion

Auf der Grundlage der eingeführten Abstraktionsabbildungen kann nun Fallabstraktion wie folge definiert werden:

Definition 5.7 (Fallabstraktion) Ein Fall $F_a = ((s_0^a, s_m^a), \langle o_1^a, \ldots, o_m^a \rangle)$ ist eine Abstraktion eines Falles $F_c = ((s_0^c, s_n^c), \langle o_1^c, \ldots, o_n^c \rangle)$ in Bezug auf die Problemlösedomänen $(\mathcal{D}_c, \mathcal{D}_a)$ wenn:

(i) für alle $i \in \{1, \ldots, n\}$ gilt $s_{i-1}^c \xrightarrow{o_i^c} s_i^c$ und
(ii) für alle $j \in \{1, \ldots, m\}$ gilt $s_{j-1}^a \xrightarrow{o_j^a} s_j^a$ für alle $j \in \{1, \ldots, m\}$ und
(iii) wenn eine Zustandsabstraktionsabbildung α und eine Sequenzabstraktionsabbildung β existieren, so daß für alle $j \in \{0, \ldots, m\}$ gilt $s_j^a = \alpha(s_{\beta(j)}^c)$.

In dieser Definition wird also gefordert, daß die Fallabstraktion durch eine einzige Zustandsabstraktionsabbildung, die alle betroffenen Zustände abstrahiert, realisiert wird.

Abbildung 5.3 zeigt noch einmal die Verwendung der Zustands- und Sequenzabstraktionsabbildungen bei der Fallabstraktion. Während die Zustandsabstraktionsabbildung die Zustandsbeschreibungen abstrahiert, wählt die Sequenzabstraktionsabbildung die konkreten Zustände aus, die abstrahiert werden. Gemäß Definition 5.4 ist die Zustandsabstraktion eine totale Abbildung. Folglich besitzen alle konkreten Zustände eine Abstraktion. Jedoch kommen, wie Abbildung 5.3 zeigt, nur die abstrakten Zustände im abstrakten Fall vor, die durch die Sequenzabstraktion ausgewählt werden. Außerdem ist zu beachten, daß die abstrakten Operatoren jeweils die abstrakten Zustände, die sich als Bild der Zustandsabstraktionsabbildung ergeben, ineinander überführen müssen.

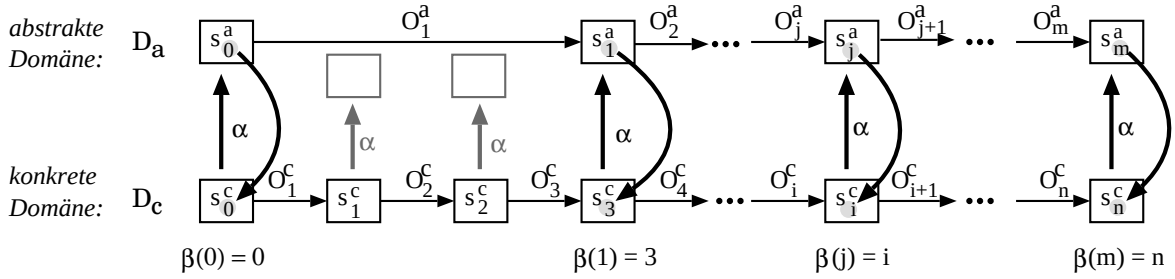

Abbildung 5.3: Fallabstraktion mit Zustands- und Sequenzabstraktionsabbildung

Beispiel 5.5 Durch die Fortführung der vorherigen Beispiele wollen wir die eingeführte Formalisierung der Fallabstraktion verdeutlichen. Abbildung 5.4 zeigt die Abstraktion des in Beispiel 5.1 beschriebenen Falls (Erhöhung eines Zahlenwertes von 0 auf 8). Hierbei wird eine gerechtfertigte Zustandsabstraktionsabbildung α verwendet, wie sie in Beispiel 5.4 angedeutet ist. Der abstrakte Fall

$$(\{\texttt{qual_wert(niedrig)}\}, \{\texttt{qual_wert(hoch)}\}), \langle \texttt{abs_erhöhe_wert}, \texttt{abs_erhöhe_wert} \rangle)$$

ist hierbei eine gültige Fallabstraktion gemäß Definition 5.7. In Abbildung 5.4 ist die Sequenzabstraktionsabbildung β mit $\beta(0) = 0, \beta(1) = 4, \beta(2) = 8$ dargestellt. Man beachte aber, daß dies nicht die einzige Sequenzabstraktionsabbildung ist, die für die Bildung dieses abstrakten Falles verwendet werden kann. Sequenzabstraktionsabbildungen mit $\beta(1) \in \{5, \ldots, 7\}$ führen ebenfalls zu einer Fallabstraktion mit demselben abstrakten Fall als Resultat.

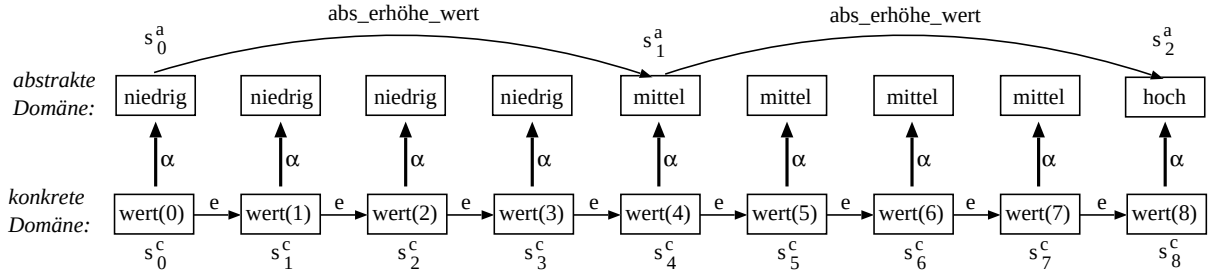

Abbildung 5.4: Einfaches Beispiel für eine Fallabstraktion. Der Operator `e` steht für `erhöhe_wert`; `niedrig`, `mittel` und `hoch` stehen für `qual_wert(niedrig)`, `qual_wert(mittel)` bzw. `qual_wert(hoch)`.

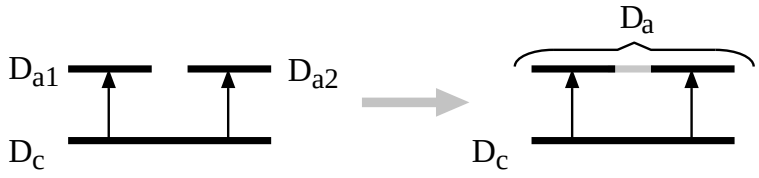

Abbildung 5.5: Vereinigung verschiedener abstrakter Sichten

Eigenschaften des Modells der Fallabstraktion

Im folgenden diskutieren wir kurz die Allgemeinheit des vorgestellten Modells der Fallabstraktion und zeigen, daß sowohl verschiedene Abstraktionsebenen, wie im hierarchischen Problemlösen üblich, als auch vollständig verschiedene abstrakte Sichten in diesem Modell realisiert werden können.

In der Regel ist es möglich, daß ein Objekt in der Welt verschiedene mögliche Abstraktionen besitzt. So kann man z. B. ganze Zahlen entsprechend ihrer Größe zu qualitativen Werten (z. B. hoch, mittel, und niedrig) abstrahieren oder aber man kann dazwischen differenzieren, ob die Zahl eine Primzahl ist oder nicht. Dies sind zwei vollständig verschiedene abstrakte Sichten, die zunächst nichts miteinander zu tun haben. Die eingeführte Abstraktionsmethodik ist in der Lage, verschiedene abstrakte Sichten, die durch den Benutzer vorgegeben werden, zu behandeln. Wir nehmen an, daß eine konkrete Domäne $\mathcal{D}_c$ gegeben ist sowie zwei abstrakte Domänen $\mathcal{D}_{a1}$ und $\mathcal{D}_{a2}$, die jeweils verschiedene Abstraktionen darstellen. Darüber hinaus wollen wir annehmen, daß die beiden abstrakten Domänen disjunkt sind, d. h. verschiedene Symbole verwenden. Nun können wir immer eine einzige abstrakte Domäne $\mathcal{D}_a$ durch Vereinigung der beiden einzelnen Domänen $\mathcal{D}_{a1}$ und $\mathcal{D}_{a2}$ definieren, die beide Abstraktionen gleichzeitig ermöglicht (siehe Abbildung 5.5).

Satz 5.1 (Vereinigung verschiedener Abstraktionen) Wenn eine konkrete Domäne $\mathcal{D}_c$ und zwei disjunkte abstrakte Domänen $\mathcal{D}_{a1} = (\mathcal{L}_{a1}, \mathcal{E}_{a1}, \mathcal{O}_{a1}, \mathcal{R}_{a1})$ und $\mathcal{D}_{a2} = (\mathcal{L}_{a2}, \mathcal{E}_{a2}, \mathcal{O}_{a2}, \mathcal{R}_{a2})$ gegeben sind, dann kann eine vereinigte abstrakte Domäne $\mathcal{D}_a$ wie folgt definiert werden: $\mathcal{D}_a = \mathcal{D}_{a1} \cup \mathcal{D}_{a2} = (\mathcal{L}_{a1} \cup \mathcal{L}_{a2}\,, \mathcal{E}_{a1} \cup \mathcal{E}_{a2}\,, \mathcal{O}_{a1} \cup \mathcal{O}_{a2}\,, \mathcal{R}_{a1} \cup \mathcal{R}_{a2})$. Es gilt die folgende Eigenschaft für $\mathcal{D}_a$: wenn F_a eine Abstraktion von F_c bezüglich ($\mathcal{D}_c$, $\mathcal{D}_{a1}$) oder bezüglich ($\mathcal{D}_c$, $\mathcal{D}_{a2}$) ist, dann ist F_a ebenfalls eine Abstraktion von F_c bezüglich ($\mathcal{D}_c, \mathcal{D}_a$).

Auf einen Beweis dieses Satzes wollen wir hier verzichten; er kann in [Bergmann & Wilke, 1995a] nachgelesen werden.

Im hierarchischen Problemlösen ist es üblich, daß verschiedene Abstraktionsebenen zum Problemlösen verwendet werden. Obwohl das vorgestellte Modell der Fallabstraktion nur zwei ausgezeichnete Abstraktionsebenen besitzt, können verschiedene Abstraktionsebenen auf dieses Zwei-Ebenen-Modell abgebildet werden (siehe Abbildung 5.6). Nehmen wir an,

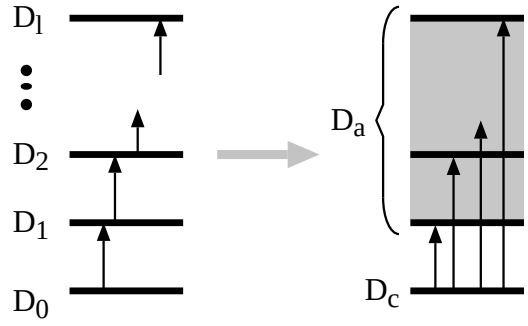

Abbildung 5.6: Vereinigung verschiedener Abstraktionsebenen

daß eine Hierarchie von Abstraktionsebenen $(\mathcal{D}_0, \ldots, \mathcal{D}_l)$ gegeben ist, wobei $\mathcal{D}_{\nu+1}$ einen höheren Abstraktionsgrad besitzt als $\mathcal{D}_\nu$. In einer solchen Abstraktionshierarchie ist ein Fall F_ν auf der Abstraktionsebene $\mathcal{D}_\nu$ eine Abstraktion des Falles F_0 der Abstraktionsebene $\mathcal{D}_0$, wenn eine Folge von Fällen $F_1, \ldots, F_{\nu-1}$ existiert, so daß F_i auf der Abstraktionsebene $\mathcal{D}_i$ liegt und F_{i+1} eine Abstraktion von F_i ist. Eine solche Hierarchie von Abstraktionsebenen kann immer zu einer Zwei-Ebenen-Hierarchie wie folgt zusammengefaßt werden.

Satz 5.2 (Vereinigung verschiedener Abstraktionsebenen) Sei $(\mathcal{D}_0, \ldots, \mathcal{D}_l)$ eine Hierarchie von disjunkten Abstraktionsebenen. Es läßt sich eine Zwei-Ebenen-Hierarchie $(\mathcal{D}_c, \mathcal{D}_a)$ wie folgt definieren: $\mathcal{D}_a = \bigcup_{\nu=1}^{l} \mathcal{D}_\nu$ und $\mathcal{D}_c = \mathcal{D}_0$. Es gilt die Eigenschaft: falls F_a eine Abstraktion von F_c bezüglich $(\mathcal{D}_0, \ldots, \mathcal{D}_l)$ ist, dann ist auch F_a eine Abstraktion von F_c bezüglich $(\mathcal{D}_c, \mathcal{D}_a)$.

Ein Beweis dieses Satzes ist ebenfalls in [Bergmann & Wilke, 1995a] zu finden.

Aufgrund dieser beiden Eigenschaften ist sichergestellt, daß das eingeführte Modell der Fallabstraktion mit seinen zwei Ebenen hinreichend allgemein ist und bisherige Ansätze subsummiert. Daher können wir uns im folgenden auf dieses Zwei-Ebenen-Modell zurückziehen.

5.2 Ein Algorithmus zur Fallabstraktion

Auf der Basis der zuvor eingeführten Spezifikation der Fallabstraktion wollen wir uns nun mit dem Prozeß der Fallabstraktion gemäß Begriffsbestimmung 4.3 beschäftigen. Hierzu wurde der PABS-Algorithmus [Bergmann, 1992a] entwickelt, der bei gegebener abstrakter Problemlösedomäne $\mathcal{D}_a$, konkreter Problemlösedomäne $\mathcal{D}_c$ und generischer Abstraktionstheorie $\mathcal{A}$, eine Menge abstrakter Fälle automatisch aus einem konkreten Fall konstruiert (vgl. [Bergmann, 1992b]).[1] Bevor wir diesen Algorithmus im Detail erörtern, geben wir

[1] Details der Implementierung finden sich in [Wilke, 1993].

zunächst einen Überblick über seine wesentlichen Phasen und demonstrieren diese an dem zuvor eingeführten einfachen Beispiel. Für ein komplexeres Beispiel aus dem Bereich der Arbeitsplanung sei auf Kapitel 8, Abschnitt 8.3.1 verwiesen.

5.2.1 Überblick über den Algorithmus

Der Prozeß der Fallabstraktion ist als ein Verfahren realisiert, das im wesentlichen den Raum der abstrakten Fälle systematisch durchsucht und dabei die abstrakten Fälle auswählt, die eine Abstraktion gemäß Definition 5.7 sind. Die Suche im Raum aller abstrakten Fälle ist dabei in vier Phasen aufgeteilt, die in den nachfolgenden Abschnitten detailliert dargestellt werden. Zum besseren Verständnis dieser Phasen wird hier zunächst eine grobe Übersicht gegeben:

Phase-I (Bestimmung der konkreten Zustandsequenz): Zunächst wird für den konkreten Fall F_c aus dem Initialzustand und der Operatorfolge die Folge der Zwischenzustände s_i^c bestimmt.

Phase-II (Bestimmung aller ableitbaren abstrakten Sätze): Für jeden so bestimmten konkreten Zustand wird mit Hilfe der generischen Abstraktionstheorie A die Menge aller ableitbaren abstrakten essentiellen Sätze bestimmt. Diese bilden eine Obermenge der essentiellen Sätze, die in den Zuständen des abstrakten Falls vorkommen können.

Phase-III (Suche nach anwendbaren abstrakten Operatoren): Für jedes Zustandspaar s_i^c, s_j^c (mit $i < j$) wird nach abstrakten Operatoren $o^a \in \mathcal{O}_a$ gesucht, so daß eine gerechtfertigte Zustandsabstraktionsabbildung α existiert mit $\alpha(s_i^c) \xrightarrow{o^a} \alpha(s_j^c)$. Alle Operatoren werden dabei in einem gerichteten Graph gespeichert.

Phase-IV (Suche nach gerichtetem Pfad im Graph): In dem so konstruierten Graph wird nach Pfaden vom Anfangszustand zum Zielzustand gesucht, so daß bei jedem Operator dieselbe Zustandsabstraktionsabbildung α angewendet wird. Für jeden Pfad gilt nun: $\alpha(s_0^c) \xrightarrow{o_1^a} \alpha(s_i^c) \xrightarrow{o_2^a} \cdots \xrightarrow{o_m^a} \alpha(s_n^c)$. Folglich entspricht jeder Pfad einem abstrakten Fall, der nun als Ergebnis zurückgegeben wird.

Beispiel 5.6 Abbildung 5.7 demonstriert diese vier Phasen für die in Beispiel 5.5 beschriebene Fallabstraktion. Wir setzen hierzu voraus, daß die abstrakte Domäne essentielle Sätze `qual_wert(X)` zur qualitativen Beschreibung der Zahlenwerte enthält, sowie einen essentiellen Satz `prim`, der angibt, ob der Zahlenwert eine Primzahl ist. Weiterhin setzen wir den zuvor eingeführten Operator `abs_erhöhe_wert` sowie einen neuen Operator `abs_erhöhe_sprunghaft` voraus. Letzterer erhöht einen qualitativen Wert von `niedrig` auf `hoch`. Die generische Abstraktionstheorie enthalte für dieses Beispiel die Klauseln zur Beschreibung der qualitativen Zahlenwerte, sowie zur Beschreibung von `prim`. Phase-I bestimmt, ausgehend vom Initialzustand s_0^c und dem Plan (8 `erhöhe_wert` Operatoren) die einzelnen Zustände $s_1^c, \ldots, s_8^c$. In Phase-II werden für jeden konkreten Zustand die durch die Abstraktionstheorie ableitbaren abstrakten essentiellen Sätze bestimmt. So läßt

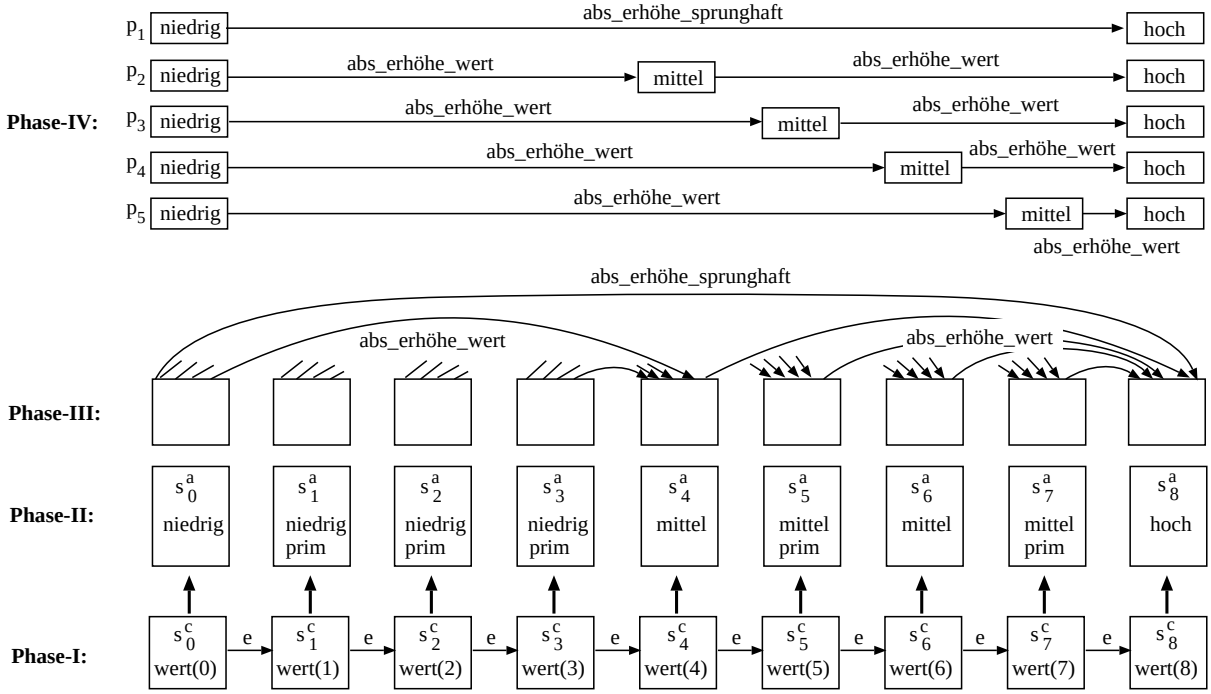

Abbildung 5.7: Die vier Phasen der Fallabstraktion am Beispiel

sich z. B. aus dem Zustand s_5^c die Gültigkeit der essentiellen Sätze `qual_wert(mittel)` und `prim` ableiten. In Phase-III wird ein gerichteter Graph konstruiert, der für jeden Zustand $s_0^a, \ldots, s_8^a$ ein Zustandknoten und gerichtete Kanten, die mit abstrakten Operatoren (`abs_erhöhe_wert` bzw. `abs_erhöhe_sprunghaft`) markiert sind, enthält. Immer wenn in einem abstrakten Zustand s_i^a der essentielle Satz `qual_wert(niedrig)` und in einem nachfolgenden abstrakten Zustand s_j^a $(j > i)$ der essentielle Satz `qual_wert(mittel)` vorkommt, wird eine gerichtete Kante vom Zustandsknoten für s_i^a zum Zustandsknoten für s_j^a eingeführt, die mit `abs_erhöhe_wert` markiert ist. Dies gilt ebenfalls, wenn s_i^a den Satz `qual_wert(mittel)` und s_j^a den Satz `qual_wert(hoch)` enthält. Außerdem enthält der Graph eine Kante, die mit dem `abs_erhöhe_sprunghaft`-Operator markiert ist und vom Initial- zum Zielzustand führt. In Phase-IV werden gerichtete Pfade vom Initial- zum Zielzustand bestimmt. In dem gezeigten Graph kommen fünf solche Pfade vor. Jeder dieser Pfade wird in einen abstrakten Fall, bestehend aus der Folge der vorkommenden Operatoren, überführt. Hierbei ergibt sich, daß die vier Pfade $p_2, \ldots, p_5$ zu demselben Fall führen (siehe auch Beispiel 5.5). Als Ergebnis werden daher die folgenden beiden abstrakten Fälle zurückgegeben:

$$(\{\{\texttt{qual_wert(niedrig)}\}, \{\texttt{qual_wert(hoch)}\}), \langle \texttt{abs_erhöhe_sprunghaft} \rangle)$$
$$(\{\{\texttt{qual_wert(niedrig)}\}, \{\texttt{qual_wert(hoch)}\}), \langle \texttt{abs_erhöhe_wert}, \texttt{abs_erhöhe_wert} \rangle).$$

5.2.2 Vorbemerkung: Beweise in der Hornlogik

Bevor wir diese vier Phasen im Detail darstellen, wollen wir zunächst eine Konvention erörtern. Da die verwendete Repräsentation des Domänenwissens auf der Hornlogik basiert, sind in den nachfolgend erörterten Algorithmen entsprechende Beweisprozeduren erforderlich. Hierzu verwenden wir die SLD-Resolution [Lloyd, 1984], wie sie auch in der Programmiersprache PROLOG als Inferenzmechanismus Verwendung findt. Diese erhält als Eingabe eine Menge von Hornklauseln C (das Programm) und eine Zielklausel G (konjunktive Formel). Sie bestimmt eine Menge von Antwortsubstitutionen Ω, so daß $C \vdash G\sigma$ für alle $\sigma \in \Omega$ gilt. Wir schreiben in diesem Fall auch: $\Omega = \mathrm{SLD}(C, G)$. Wenn der Beweisversuch von G erfolglos geblieben ist, dann liefert die SLD-Prozedur keine Antwortsubstitution, d. h. $\Omega = \emptyset$. Dies darf jedoch nicht mit dem Fall verwechselt werden, in dem der Beweis erfolgreich war und keine Bindung von Variablen aus G erforderte. In diesem Fall liefert die SLD-Prozedur eine Menge mit der leeren Substitution zurück, also $\Omega = \{\emptyset\}$. In einigen Fällen benötigen wir außerdem zu jeder Antwortsubstitution den zugehörigen Beweisbaum als Rechtfertigung. In diesem Fall schreiben wir $\Pi = SLD(C, G)$, wobei Π eine Menge von Paaren der Gestalt (σ, τ) ist und σ eine Antwortsubstitution und τ der zugehörige Beweisbaum ist. Um weiterhin die Terminierung des SLD-Beweises zuzusichern, müssen wir zusätzlich verlangen, daß die konkrete und abstrakte Problemlösedomäne sowie die generische Abstraktionstheorie keine unendlichen Beweisketten verursachen und daß die Menge der möglichen Antwortsubstitutionen endlich ist.

5.2.3 Phase-I: Bestimmung der konkreten Zustandssequenz

Als Eingabe in Phase-I erwarten wir einen konkreten Fall $F_c = ((s_I^c, s_G^c), \langle o_1^c, \ldots, o_n^c \rangle)$. In dieser Phase wird die Zustandssequenz bestimmt, die sich aus der Planausführung ergibt. Abbildung 5.8 zeigt den entsprechenden Algorithmus. Durch diesen Algorithmus werden

---

Eingabe: Konkreter Fall $F_c = ((s_I^c, s_G^c), \langle o_1^c, \ldots, o_n^c \rangle$
Ausgabe: Konkrete Zustandssequenz $s_0^c, \ldots, s_n^c$

$s_0^c := s_I^c$
for $i := 1$ **to** n **do**
  if $\mathrm{SLD}(s_{i-1}^c \cup \mathcal{R}_c,\ Pre_{o_i^c}) = \emptyset$ **then** STOP "Fehler: Operator nicht anwendbar"
  $s_i^c := (s_{i-1}^c \setminus Del_{o_i^c}) \cup Add_{o_i^c}$
  end
if $s_G^c \not\subseteq s_n^c$ **then** STOP "Fehler: Zielzustand nicht erreicht"

Abbildung 5.8: Algorithmus zur Bestimmung der konkreten Zustandssequenz

die Zustände s_i^c berechnet, so daß $s_{i-1}^c \xrightarrow{o_i^c} s_i^c$ für alle $i \in \{1, \ldots, n\}$ gilt. Der Algorithmus terminiert mit einer Fehlermeldung genau dann, wenn die im vorliegenden Fall enthaltene Lösung keine korrekte Lösung des vorliegenden Problems ist.

5.2.4 Phase-II: Bestimmung aller ableitbaren abstrakten Sätze

In Phase-II (siehe Abbildung 5.9) werden für jeden konkreten Zustand die Menge der durch die generische Abstraktionstheorie ableitbaren abstrakten Sätze bestimmt. Durch diesen Schritt wird für jeden konkreten Zustand eine Obermenge der abstrakten Sätze bestimmt, die als Bild einer gerechtfertigten Zustandsabstraktionsabbildung entstehen können. Daher gilt für jede gerechtfertigte Zustandsabstraktionsabbildung α die Bedingung: $\alpha(s_i^c) \subseteq s_i^a$.

Eingabe: Konkrete Zustandssequenz $s_0^c, \ldots, s_n^c$
Ausgabe: Abstrakte Zustandssequenz $s_0^a, \ldots, s_n^a$

for $i := 0$ **to** n **do**
  $s_i^a := \emptyset$
  for each $E \in \mathcal{E}_a$ **do**
    $\Omega := \mathrm{SLD}(s_i^c \cup \mathcal{R}_c \cup \mathcal{A}, E)$
    for each $\sigma \in \Omega$ **do**
      $s_i^a := s_i^a \cup \{E\,\sigma\}$
      end
    end
  end

Abbildung 5.9: Algorithmus zur Bestimmung der ableitbaren abstrakten Sätze

5.2.5 Phase-III: Suche nach anwendbaren abstrakten Operatoren

In dieser Phase suchen wir nach instantiierten abstrakten Operatoren, die einen abstrakten Zustand $\tilde{s}_i^a \subseteq s_i^a$ in einen abstrakten Folgezustand $\tilde{s}_j^a \subseteq s_j^a$ $(i < j)$ überführen. Hierzu muß in jedem Fall die Vorbedingung des abstrakten Operators in $\tilde{s}_i^a$ und folglich auch in s_i^a erfüllt sein. Außerdem müssen alle Effekte der Addliste des Operators in $\tilde{s}_j^a$ gelten und folglich auch in s_j^a.

Die Menge aller möglichen Übergänge mit abstrakten Operatoren werden in einem gerichteten Graph representiert, in dem die Knoten abstrakte Zustände darstellen und die Kanten die abstrakten Operatoren. In dem in Abbildung 5.10 dargestellten Algorithmus

zur Realisierung von Phase-III wird die Menge G der Kanten dieses Graphen konstruiert. Für jedes Paar von Zuständen (s_i^a,s_j^a) mit $i < j$ wird überprüft, ob ein abstrakter Operator $o(x_1, \ldots, x_u)\sigma$ existiert, der in s_i^a anwendbar ist. Hierzu bestimmt die SLD-Beweisprozedur

---

Eingabe: Abstrakte Zustandssequenz $s_0^a, \ldots, s_n^a$
Ausgabe: Gerichteter Graph G mit anwendbaren abstrakten Operatoren

$G := \emptyset$
for $i := 0$ **to** $n - 1$ **do**
  for $j := i + 1$ **to** n **do**
    for each $o(x_1, \ldots, x_u) \in \mathcal{O}_a$ **do**
      Sei (Pre_o, Del_o, Add_o) die Beschreibung von $o(x_1, \ldots, x_u)$
      $\Pi :=$ SLD$(s_i^a \cup \mathcal{R}_a, Pre_o)$
      for each $(\sigma, \tau) \in \Pi$ **do**
        let $Add'_o = \{a\sigma | a \in Add_o\}$
        (Bestimme alle möglichen Instantiierungen *)*
        (von Sätzen der Addliste, die in s_j^a gelten. *)*
        $M := \{\emptyset\}$
        (M ist die Menge der möglichen Substitutionen, *)*
        (am Anfang die leere Substitution. *)*
        for each $a \in Add'_o$ **do**
          $M' := \emptyset$
          for each $\theta \in M$ **do**
            for each $e \in s_j^a$ **do**
              if es gibt eine Substitution ρ mit $a\theta\rho = e$ **then** $M' := M' \cup \{\theta\rho\}$
              end
            end
          $M := M'$
          end
          (M enthält nun die Menge aller möglichen Substitutionen, *)*
          (so daß alle hinzugefügten Sätze aus s_j^a sind. *)*
          for each $\theta \in M$ **do**
            $G := G \cup \{(i, j, o(x_1, \ldots, x_u)\sigma\theta, \tau)\}$
            end
          end
        end
      end
    end

Abbildung 5.10: Algorithmus zur Suche nach anwendbaren abstrakten Operatoren

die Menge aller Antwortsubstitutionen, σ so daß die Vorbedingung des Operators in s_i^a erfüllt ist. Außerdem wird mit jedem Zustandsübergang auch der Beweisbaum τ, der die Gültigkeit der Operatorvorbedingung nachweist, gespeichert. Dieser Beweisbaum ist ein „Und-Baum“, in dem die inneren Knoten die Resolution eines Zielliterals mit dem Kopf einer Klausel repräsentieren, und die Blattknoten repräsentieren die Resolution mit einem Fakt. Um die Vorbedingung eines Operators nachzuweisen, beziehen sich die inneren Knoten immer auf eine Hornklausel aus der Regelmenge $\mathcal{R}_a$, während die Blattknoten Fakten aus $\mathcal{R}_a$ oder essentielle Sätze aus s_i^a darstellen (siehe Abbildung 5.11). Jede Anwortsubstitution σ wird sodann auf die Addliste des Operators angewendet und führt zu einer partiell instantiierten Addliste Add'_o. Da in Add'_o noch Variablen enthalten sein können, wird die Menge M aller möglichen Substitutionen θ inkrementell derart konstruiert, daß $a\theta \in s_{aj}$ für alle $a \in Add'_o$ gilt. Der vollständig instantiierte Operator, der hierdurch bestimmt ist, wird schließlich als gerichtetete Kante dem Graphen G hinzugefügt.

Durch diesen Algorithmus ist zugesichert, daß falls $\alpha(s^c_{\beta(i-1)}) \xrightarrow{o_i^a} \alpha(s^c_{\beta(i)})$ für eine beliebige Zustandsabstraktionsabbildung α und eine beliebige Sequenzabstraktionsabbildung β gilt, auch $(\beta(i-1), \beta(i), o_i^a, \tau)$ in G enthalten ist.

5.2.6 Phase-IV: Suche nach gerichtetem Pfad im Graph

In Phase-IV (siehe Abbildung 5.12) wird nun nach Pfaden vom Initialzustand zum Zielzustand im Graph G gesucht, so daß jeder Pfad eine gültige Abstraktion gemäß Definition 5.7 darstellt. Hierzu werden entsprechende Pfade konstruiert und dabei die Menge relevanter abstrakter Sätze α^* (siehe Bemerkung 5.3) und die Sequenzabstraktion bestimmt.

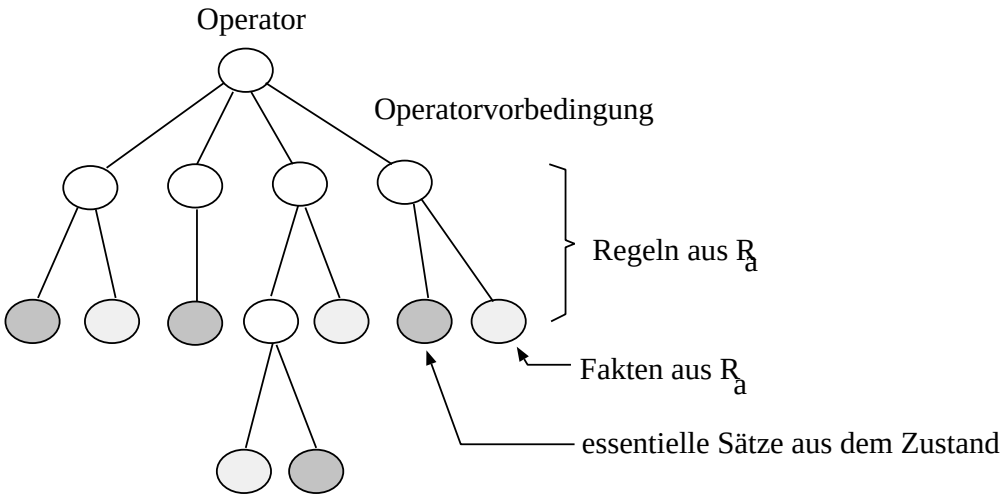

Abbildung 5.11: Beispiel für einen Beweisbaum für eine Operatorvorbedingung

Der Algorithmus beginnt mit einem leeren Pfad. In jeder Iteration wird der gerade vorliegende Pfad durch einen Operator aus G erweitert, bis der Pfad den Zielzustand erreicht. Neue essentielle Sätze, die im Beweisbaum der Operatorvorbedingung vorkommen oder neue essentielle Sätze aus der Addliste werden dabei zur Menge der relevanten Sätze α^* hinzugenommen und es wird geprüft, ob der Anfang des Pfades noch immer mit der neuen Menge α^* konsistent ist. Als Ergebnis liefert die Phase-IV alle Fälle, die eine Abstraktion des konkreten Eingabefalles sind, zurück. Wir verwenden die folgende funktionale Notation, um auszudrücken, daß ein abstrakter Fall F_a in der Menge der vom PABS-Algorithmus zurückgelieferten Fälle liegt: $F_a \in PABS((\mathcal{D}_c, \mathcal{D}_a, \mathcal{A}), F_c)$.

Für die in Beispiel 5.6 gezeigten Pfade $p_2, \ldots, p_5$ (siehe Abbildung 5.7) wurde in diesem Algorithmus die Menge $\alpha^* = \{$`qual_wert(niedrig)`,`qual_wert(mittel)`,`qual_wert(hoch)` $\}$ und für den Pfad p_1 die Menge $\alpha^* = \{$ `qual_wert(niedrig)`, `qual_wert(hoch)` $\}$ konstruiert. Der essentielle Satz `prim` kommt hier nicht vor. Der Algorithmus bestimmt also in Phase-IV eine minimale Menge von essentiellen Sätzen der abstrakten Problemlösedomäne, so daß der durch den Pfad repräsentierte abstrakte Plan korrekt ist.

Eingabe: Gerichteter Graph G mit anwendbaren abstrakten Operatoren
Ausgabe: Menge von abstrakten Fällen $Cases_{ABS}$

$Paths := \{(\langle\rangle, \emptyset, (\beta(0) = 0))\}$
while es existiert $(\langle o_1^a, \ldots, o_k^a\rangle, \alpha^*, \beta) \in Paths$ mit $\beta(k) < n$ **do**
  $Paths := Paths \setminus (\langle o_1^a, \ldots, o_k^a\rangle, \alpha^*, \beta)$
  for each $(i, j, o^a, \tau) \in G$ mit $i = \beta(k)$ **do**
    Sei $\tau_{\mathcal{E}}$ die Menge der essentiellen Sätze aus den Blättern von τ
    Sei $\alpha' = \tau_{\mathcal{E}} \cup Add_{o^a} \cup \alpha^*$
    if for all $\nu \in \{1, \ldots, k\}$ gilt:
      $(s^a_{\beta(\nu-1)} \cap \alpha') \xrightarrow{o^a_\nu} (s^a_{\beta(\nu)} \cap \alpha')$ **and**
      $(s^a_{\beta(k)} \cap \alpha') \xrightarrow{o^a} (s^a_j \cap \alpha')$ **then**
        $Paths := Paths \cup \{(\langle o_1^a, \ldots, o_k^a, o^a\rangle, \alpha', \beta \cup \{\beta(k+1) = j\})\}$
    end
  end
$Cases_{Abs} := \emptyset$
for each $(\langle o_1^a, \ldots, o_k^a\rangle, \alpha^*, \beta) \in Paths$ mit $\beta(k) = n$ **do**
  $Cases_{Abs} := Cases_{Abs} \cup \{((s_0^a \cap \alpha^*, s_n^a \cap \alpha^*), \langle o_1^a, \ldots, o_k^a\rangle)\}$
  end
return $Cases_{Abs}$

Abbildung 5.12: Algorithmus zur Suche nach gerichteten Pfaden im Graph

5.2.7 Korrektheit und Vollständigkeit des PABS-Algorithmus

Wie in [Bergmann & Wilke, 1995a] gezeigt wurde, ist der PABS-Algorithmus vollständig und korrekt in Bezug auf die Definition 5.7 der Fallabstraktion.

Satz 5.3 (Vollständigkeit und Korrektheit des PABS-Algorithmus) Wenn im PABS-Algorithmus eine vollständige SLD-Beiweisprozedur verwendet wird, dann ist ein Fall F_a eine Abstraktion eines Falls F_c bezüglich $(\mathcal{D}_c, \mathcal{D}_a)$ und unter Verwendung einer Zustandsabstraktionsabbildung, die durch die generische Abstraktionstheorie $\mathcal{A}$ gerechtfertigt ist, *genau dann wenn* der PABS-Algorithmus F_a als Ausgabe liefert, d. h. wenn $F_a \in PABS((\mathcal{D}_c, \mathcal{D}_a, \mathcal{A}), F_c)$ gilt.

5.2.8 Komplexität des PABS-Algorithmus

Die Komplexität des Algorithmus wird durch die Phasen III und IV bestimmt, da die ersten beiden Phasen eine geringere Komplexität aufweisen. Die Worst-Case-Komplexität von Phase-III ist $O(n^2 \cdot C_1 \cdot C_2)$, wobei n die Länge des Plans im konkreten Eingabefall ist. Die Werte C_1 und C_2 hängen wie folgt von der abstrakten Problemlösedomäne ab: $C_1 = |\mathcal{O}_a| \cdot |\Omega|$ und $C_2 = |Add_{\mathcal{O}_a}| \cdot (|\mathcal{E}_a| \cdot |\Omega|)^{|Add_{\mathcal{O}_a}|}$. Hierbei entspricht $|\mathcal{O}_a|$ der Anzahl der abstrakten Operatoren, $|\Omega|$ ist die maximale Anzahl von Substitutionen, die die SLD-Prozedur findet, $|Add_{\mathcal{O}_a}|$ ist die maximale Größe der Addlist aller abstrakten Operatoren, und $|\mathcal{E}_a|$ ist die Anzahl der abstrakten essentiellen Sätze, die in der Problemlösedomäne vorkommen. Die Komplexität von Phase-IV läßt sich zu $O(n \cdot 2^{(n-1)} \cdot C_1)$ bestimmen. Der exponentielle Anteil ergibt sich aus der exponentiellen Anzahl von Pfaden in dem azyklischen gerichteten Graphen G. Wenn man die Domänentheorien als konstant annimmt, so ergibt sich eine Gesamtkomplexität des Verfahrens von $O(n \cdot 2^{(n-1)})$.

Satz 5.4 (Komplexität des PABS-Algorithmus) Der PABS-Algorithmus hat bei konstanten Domänentheorien eine Komplexität von $O(n \cdot 2^{(n-1)})$, wobei n die Länge des Plans im konkreten Falls ist.

5.3 Bestimmung der Wiederverwendbarkeit abstrakter Fälle

Um einen abstrakten Fall zur Lösung eines neuen Problemes einsetzen zu können, ist eine Präzisierung des Wiederverwendbarkeitsbegriffes gemäß Begriffsbestimmung 4.4 erforderlich. Hierbei sollte ein abstrakter Fall wiederverwendbar zur Lösung eines neuen konkreten Problems sein, wenn sich die Lösung im abstrakten Fall zu einer konkreten Lösung des Problems verfeinern läßt. Aufgrund dieser Überlegung läßt sich die Wiederverwendbarkeit eines abstrakten Falles wie folgt definieren.

Definition 5.8 (Wiederverwendbarkeit eines abstrakten Falles) Ein abstrakter Fall F_a kann zur Lösung eines konkreten Problems p wiederverwendet werden, wenn es eine Lösung $\bar{o}_c$ von p gibt, so daß F_a eine Abstraktion von $F_c = (p, \bar{o}_c)$ gemäß Definition 5.7 ist.

Die Wiederverwendbarkeit ist jedoch als praktisches Auswahlkriterium für abstrakte Fälle ungeeignet, da sie unentscheidbar ist.[2] Aus diesem Grunde ist es erforderlich, die Wiederverwendbarkeitsbedingung so weit abzuschwächen, daß sie entscheidbar wird. Die folgende Definition gibt eine solche abgeschwächte Wiederverwendbarkeitsbedingung.

Definition 5.9 (Schwache Wiederverwendbarkeit eines abstrakten Falles) Ein abstrakter Fall $F_a = ((s_0^a, s_m^a), \bar{o}_a)$ ist zur Lösung eines konkreten Problems $p = (s_0^c, s_n^c)$ schwach wiederverwendbar, falls es eine gerechtfertigte Zustandsabstraktionsabbildung α gibt, so daß $\alpha(s_0^c) = s_0^a$ und $\alpha(s_n^c) = s_m^a$ gilt.

Wie leicht zu sehen ist, ist die schwache Wiederverwendbarkeit eine notwendige Voraussetzung für die Wiederverwendbarkeit eines Falles. Jedoch ist sie natürlich keine hinreichende Bedingung dafür, daß die im Fall enthaltene Lösung tatsächlich zu einer Lösung des Problems p verfeinert werden kann. Jedoch hat die schwache Wiederverwendbarkeit den Vorteil, daß sie a-priori entscheidbar ist, und das sogar mit geringem Aufwand.

Beispiel 5.7 Betrachten wir die in Beispiel 5.6 dargestellten abstrakten Fälle. Beide Fälle sind wiederverwendbar (und somit auch schwach wiederverwendbar) zur Lösung des Problems ({`wert(2)`}, {`wert9)`}) (Voraussetzung: generische Abstraktionstheorie gemäß Beispiel 5.2).

Der in Abbildung 5.13 dargestellte Algorithmus prüft die schwache Wiederverwendbarkeit eines abstrakten Falles $F_a = ((s_0^a, s_m^a), \langle o_1^a, \ldots, o_m^a \rangle)$ zur Lösung des konkreten Problems $p = (s_0^c, s_n^c)$. In diesem Algorithmus wird zunächst die Menge α^* bestimmt, die eine Zustandsabstraktionsabbildung α induziert (siehe Bemerkung 5.3). Danach wird diese durch Ableitung der in α^* vorkommenden Sätze angewendet, so daß $s_I^a = \alpha(s_0^c)$ und $s_G^a = \alpha(s_n^c)$ gilt. Letztlich wird die Gültigkeit der schwachen Wiederverwendbarkeitsbedingung überprüft. Als Ergebnis bestimmt dieser Algorithmus, neben der Entscheidung über die schwache Wiederverwendbarkeit, zusätzlich auch die Zustandsabstraktionsabbildung α, aufgrund welcher die schwache Wiederverwendbarkeit beruht.

Der Aufwand zur Bestimmung der schwachen Wiederverwendbarkeit ist sehr gering und wird im wesentlichen durch die Kosten zur Ableitung der abstrakten essentiellen Sätze bestimmt.

Satz 5.5 (Komplexität des Tests auf schwache Wiederverwendbarkeit) Sei m die Länge des abstrakten Plans und sei s die maximale Anzahl der essentiellen Sätze in

[2] Wäre diese Bedingung entscheidbar, so wäre auch die Frage entscheidbar, ob ein Problem eine Lösung besitzt. Dies ist jedoch bekanntermaßen ein unentscheidbares Problem [Chapman, 1987].

Eingabe: Ein abstrakter Falle $F_a = ((s_0^a, s_m^a), \langle o_1^a, \ldots, o_m^a \rangle)$ und ein konkretes Problem $p = (s_0^c, s_n^c)$

Ausgabe: Entscheidung der schwachen Wiederverwendbarkeit

$\alpha^* := s_0^a \cup \bigcup_{i=1}^{m} Add_{o_i^a}$
$s_I^a := s_G^a := \emptyset$
for each $E \in \alpha^*$ **do**
  if $\mathrm{SLD}(s_0^c \cup \mathcal{R}_c \cup \mathcal{A}, E) \neq \emptyset$ **then** $s_I^a := s_I^a \cup E$
for each $E \in \alpha^*$ **do**
  if $\mathrm{SLD}(s_n^c \cup \mathcal{R}_c \cup \mathcal{A}, E) \neq \emptyset$ **then** $s_G^a := s_G^a \cup E$
if $s_I^a = s_0^a$ und $s_G^a = s_m^a$
  then F_a ist schwach wiederverwendbar für p
  else F_a ist nicht schwach wiederverwendbar für p

Abbildung 5.13: Algorithmus zur Entscheidung der schwachen Wiederverwendbarkeitsbedingung

einem abstrakten Zustand, d. h. $|s_i^a| \leq s$. Seien weiterhin C_{Abs} die Kosten zur Ableitung eines abstrakten Satzes durch die SLD-Resolution. Die Komplexität zur Entscheidung der schwachen Wiederverwendbarkeitsbedingung ergibt sich zu $O(m \cdot s \cdot (m + s + C_{Abs}))$.

Beweis Zur Berechnung von α^* werden m Vereinigungsoperationen durchgeführt, wobei α^* schrittweise anwächst. Schätz man $|Add_{o_i^a}|$ nach oben durch s ab, so führt dies zu einer Komplexität von $O(m^2 \cdot s)$ Die Kardinalität von α^* läßt sich durch $m \cdot s$ begrenzen. Jede der beiden Schleifen erfordert bei jedem Durchlauf eine Komplexität von $O(C_{Abs} + s)$. Insgesamt ergibt sich also $O(m \cdot s \cdot (m + s + C_{Abs}))$.

5.4 Verfahren zur Lösungsverfeinerung

Liegt ein schwach wiederverwendbarer abstrakter Fall für ein Problem vor, so stellt sich das Problem der Lösungsverfeinerung gemäß Begriffsbestimmung 3.19. Lösungsverfeinerung kann nun wie folgt formalisiert werden.

Definition 5.10 (Lösungsverfeinerung) Eine Lösung $\bar{o}$ ist eine Verfeinerung des Falls F_a genau dann wenn es ein Problem p auf der konkreten Problemlösedomäne gibt, so daß F_a eine Abstraktion des Falls $(p, \bar{o})$ gemäß Definition 5.7 ist.

5.4.1 Grundlegende Vorgehensweise

Um eine solche Verfeinerung zu finden, kann die abstrakte Lösung als eine Dekomposition des gesamten Problems in Teilprobleme angesehen werden. Diese Teilprobleme werden durch die abstrakten Zwischenzustände definiert, die sich aufgrund des vorliegenden abstrakten Plans ergeben. Jedes Teilproblem besteht dabei im Finden einer Sequenz von konkreten Operatoren, die einen konkreten Zustand erreichen, der sich auf den angestrebten abstrakten Zustand mit Hilfe einer Zustandsabstraktionsabbildung abbilden läßt (siehe Abbildung 5.14).

Der Algorithmus in Abbildung 5.15 realisiert auf einfache Weise eine Lösungsverfeinerung eines abstrakten Falles $F_a = ((s_0^a, s_m^a), \langle o_1^a, \ldots, o_m^a \rangle)$ derart, daß eine Lösung eines vorgelegten konkreten Problems p bestimmt wird. Hierbei wird das Vorhandensein einer Zustandsabstraktionsabbildung α, wie sie bei der Bestimmung der schwachen Wiederverwendbarkeit des abstrakten Falles entsteht, vorausgesetzt.

Dieser Algorithmus bestimmt die zu lösenden Teilprobleme und löst sie ausgehend vom Initialzustand. Dadurch führt dieser Basisalgorithmus insgesamt eine vorwärtsgerichtete Lösungssuche durch. Als Ergebnis liefert er in $\bar{o}$ eine Lösung des Problems p. Die gefundene Lösung ist eine korrekte Lösung des vorliegenden Problems und ist eine Verfeinerung der Lösung im abstrakten Fall.

Beispiel 5.8 Betrachten wir den in Beispiel 5.6 dargestellten abstrakten Fall, der einen abstrakten Plan, bestehend aus 2 `abs_erhöhe_wert`-Operatoren, enthält. Zur Lösung des Problems ({`wert(2)`}, {`wert9)`}) löst der Verfeinerungsalgorithmus zwei Teilprobleme:

1. das Problem, vom konkreten Zustand {`wert(2)`} zu einem Zustand zu gelangen, der sich zu `qual_wert(mittel)` abstrahieren läßt (z. B. {`wert(4)`}), und
2. das Problem, vom zuvor erreichten konkreten Zustand zum Zustand {`wert(9)`} zu gelangen.

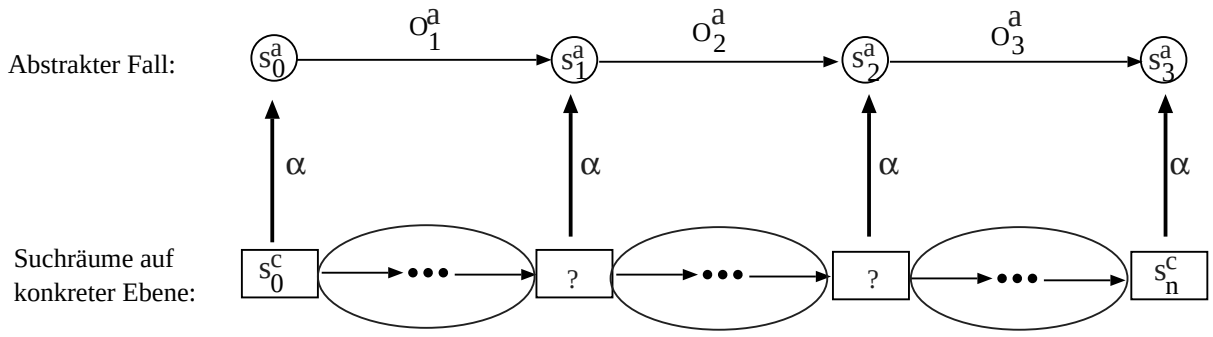

Abbildung 5.14: Suchräume bei der Lösungsverfeinerung

Eingabe: Ein abstrakter Fall $F_a = ((s_0^a, s_m^a), \langle o_1^a, \ldots, o_m^a \rangle)$, ein konkretes Problem $p = (s_0^c, s_n^c)$ und eine Zustandsabstraktionsabbildung α mit $\alpha(s_0^c) = s_0^a$ und $\alpha(s_n^c) = s_m^a$

Ausgabe: Eine Lösung $\bar{o}$ für p

$s^c := s_0^c$
$\bar{o} := \langle\rangle$
for $i := 1$ **to** $m - 1$ **do**
  $s_i^a := (s_{i-1}^a \setminus Del_{o_i^a}) \cup Add_{o_i^a}$
  Löse das Problem, einen Pfad von Zustand s^c zu einem Zustand $\tilde{s}_c$ mit $\alpha(\tilde{s}^c) = s_i^a$ zu finden.
  Die Lösung des Problems sei $\bar{o}_i$.
  $\bar{o} := \bar{o} \circ \bar{o}_i$
  $s^c := \tilde{s}^c$
  end
Löse das Problem, einen Pfad von Zustand s^c zum Zustand s_n^c zu finden.
Die Lösung des Problems sei $\bar{o}_m$.
$\bar{o} := \bar{o} \circ \bar{o}_m$

Abbildung 5.15: Basisalgorithmus zur Lösungsverfeinerung

Für weitere, komplexere Beispiele zur Lösungsverfeinerung sei auf Kapitel 8, Abschnitt 8.3.6 verwiesen.

Satz 5.6 (Korrektheit der Lösungsverfeinerung) Wenn der Algorithmus 5.15 terminiert, liefert er eine korrekte Lösung $\bar{o}$ des Problems p. Die Lösung $\bar{o}$ ist eine Verfeinerung von F_a

Beweisskizze Die Korrektheit der Lösung ist offensichtlich, da die Teillösungen o_i korrekt sind, die Konkatenation der Teillösungen eine neue Lösung ergibt und diese vom Initial- zum Zielzustand führt. Außerdem ist die Lösung eine Verfeinerung des Falls F_a, da F_a eine Abstraktion des Falls $(p, \bar{o})$ ist.

Bemerkung 5.4 Eine alternative Strategie besteht darin, ein rückwärtsgerichtete Suche durchzuführen und dabei zunächst das letzte Teilproblem zu lösen, dann das vorletzte, usw. Der Algorithmus hierzu ist „symmetrisch“ und unterscheidet sich im wesentlichen dadurch, daß die Schleife, die die Teilprobleme der Reihe nach löst, in umgekehrter Reihenfolge durchlaufen werden muß. Daher wird auf eine detaillierte Darstellung dieser Variante hier verzichtet.

Dieser Algorithmus läßt jedoch zwei wichtige Fragen offen:

- Im vorliegenden Basisalgorithmus müssen die entstehenden Teilprobleme einzeln gelöst werden. Es stellt sich die Frage, welche Problemlösemethoden hierzu eingesetzt werden können.
- Aus verschiedenen Gründen kann es vorkommen, daß einige der Teilprobleme, nicht lösbar sind. Daher stellt sich die Frage, was in diesem Fall zu tun ist.

5.4.2 Suchverfahren zur Lösung der Teilprobleme

Im Basisalgorithmus zur Lösungsverfeinerung durch vorwärtsgerichtete Suche wird ein Teilproblem durch einen konkreten Anfangszustand s^c und einen abstrakten Zielzustand s_i^a definiert. Der abstrakte Zielzustand korrespondiert dabei mit einer Menge konkreter Zielzustände $\tilde{s}_c$ mit $\alpha(\tilde{s}^c) = s_i^a$. Obwohl der Basisalgorithmus vorwärtsgerichtet vorgeht, lassen sich die hierbei definierten Teilprobleme prinzipiell sowohl durch einen Problemlöseprozeß, der eine vorwärtsgerichtete Suche durchführt, als auch durch einen Problemlöseprozeß, der eine rückwärtsgerichtete Suche durchführt, lösen. Da es jedoch keinen eindeutigen konkreten Zielzustand gibt, ergeben sich bei der Realisierung erhebliche Unterschiede zwischen vorwärtsgerichteter und rückwärtsgerichteter Suche.

Vorwärtsgerichtete Suche zur Lösung der Teilprobleme

Bei der vorwärtsgerichteten Suche wendet man, ausgehend vom konkreten Anfangszustand, solange Operatoren an, bis ein Zustand erreicht ist, der dem Zielzustand entspricht. Hierbei kann die Steuerung der Suche auf verschiedene Arten, wie etwa durch Breitensuche, Tiefensuche oder Bestensuche erfolgen. Die wesentliche Eigenschaft der vorwärtsgerichteten Suche ist, daß die gerade betrachteten Zustände explizit berechnet werden. Dies ermöglicht es, zu testen, ob ein Zustand erreicht wurde, der sich geeignet abstrahieren läßt. Die Bedingung für das Erreichen eines Zielzustand $\tilde{s}_c$ mit $\alpha(\tilde{s}^c) = s_i^a$ läßt sich daher sehr einfach als Erfolgskriterium einer vorwärtsgerichteten Suche realisieren.

Rückwärtsgerichtete Suche zur Lösung der Teilprobleme

Die rückwärtsgerichtete Suche beginnt mit einem konkreten Zielzustand und wendet die verfügbaren Operatoren rückwärts an, bis der vorliegende Anfangszustand erreicht ist. Hierbei ist es jedoch ein Problem, daß kein konkreter Zielzustand vorliegt, sondern lediglich ein abstrakter. Daher muß zunächst ein konkreter Zielzustand $\tilde{s}_c$ mit $\alpha(\tilde{s}^c) = s_i^a$ aus der Menge aller möglichen konkreten Zustände ausgewählt werden. Erst danach kann das so entstandene konkrete Problem mit rückwärtsgerichteter Suche gelöst werden. Schlägt die Suche fehl, so müssen alternative Zustandskonkretisierungen betrachtet werden. Durch die Auswahl eines konkreten Zielzustandes wird also ein zusätzlicher Entscheidungspunkt in die Suche eingefügt, wodurch sich der Suchaufwand insgesamt vergrößert, insbesondere wenn ein abstrakter Zustand sehr viele mögliche Konkretisierungen besitzt.

Wie bereits erwähnt, setzen die vorherigen Betrachtungen zur Suchrichtung bei der Lösung der Teilprobleme ein vorwärtsgerichtetes Vorgehen im Basisalgorithmus voraus. Bei einem rückwärtsgerichten Vorgehen im Basisalgorithmus kehrt sich die Situation um, und eine rückwärtsgerichtete Suche zur Lösung der Teilprobleme bringt Vorteile gegenüber der vorwärtsgerichteten Lösung. Im folgenden halten wir an der Variante, eine vorwärtsgerichtete Vorgehensweise im Basisalgorithmus und zur Lösung der Teilprobleme zu verfolgen, fest.

5.4.3 Backtracking zwischen der Lösung von Teilproblemen

Der eingeführte Basisalgorithmus löst die entstehenden Teilprobleme unabhängig voneinander. Ist für ein Teilproblem eine Lösung bestimmt, so wird diese nicht mehr verändert. Diese Vorgehensweise führt jedoch zu Problemen, wenn die Teilprobleme nicht unabhänigig voneinander sind, also die Lösung eines Teilproblemes einen Einfluß auf die Lösung eines anderen Teilproblemes hat.

Ob diese Unabhängigkeit gegeben ist, hängt von der Definition der abstrakten und konkreten Problemlösedomänen ab. Die folgende Bedingung definiert die unabhängige Verfeinerbarkeit abstrakter Operatoren.

Definition 5.11 (Unabhängige Verfeinerbarkeit abstrakter Operatoren) Ein abstrakter Operator o^a ist unabhängig verfeinerbar, wenn für alle Zustände s^c, $\tilde{s}^c \in \mathcal{S}_c$ und jede Zustandsabstraktionsabbildung α gilt: falls $\alpha(s^c) \xrightarrow{o^a} \alpha(\tilde{s}^c)$ gilt, dann existiert eine Folge von konkreten Operatoren $\langle o_1^c, \ldots, o_k^c \rangle$, so daß $s^c \xrightarrow{o_1^c} \ldots \xrightarrow{o_k^c} \tilde{s}^c$ gilt.

Beispiel 5.9 Die in den vorherigen Beispielen vorkommenden abstrakten Operatoren `abs_erhöhe_wert` und `abs_erhöhe_sprunghaft` sind unabhängig verfeinerbar gemäß dieser Definition.

Sind alle abstrakten Operatoren in einem abstrakten Fall unabhängig verfeinerbar, so können alle Teilprobleme bei der Lösungsverfeinerung, so wie im Algorithmus 5.15 vorausgesetzt, unabhängig voneinander gelöst werden.

Jedoch ist diese Bedingung so restriktiv, daß es in der Praxis sehr schwierig ist, eine Problemlösedomäne zu erstellen, in der alle abstrakten Operatoren dieser Bedingung genügen. Daher ist es erforderlich, den Fall zu berücksichtigen, daß eine abstrakte Lösung verfeinert werden muß, in der nicht alle abstrakten Operatoren unabhängig verfeinerbar sind. In diesem Fall muß ein Backtracking zwischen der Lösung der einzelnen Teilprobleme eingeführt werden. Ist ein Teilproblem nicht lösbar, so muß versucht werden, eine alternative Lösung des vorherigen Teilproblems zu finden. Diese alternative Lösung resultiert dann (in der Regel) in einem anderen konkreten Zielzustand und folglich in einem anderen konkreten Anfangszustand für das zuvor fehlgeschlagene Teilproblem.

5.4.4 Lösungsverfeinerung mit Backtracking und Tiefensuche mit schrittweiser Tiefenerhöhung

Die Überlegungen der vorherigen Abschnitte sollen nun dazu genutzt werden, einen detaillierten Algorithmus zur Lösungsverfeinerung zu entwickeln, der eine

- vorwärtsgerichtete Suche realisiert,
- Backtracking zwischen der Lösung der Teilprobleme durchführt und
- Tiefensuche mit schrittweiser Tiefenvergrößerung (engl. depth-first iterative-deepening) [Korf, 1985a; Korf, 1993] verwendet.

Bemerkung 5.5 Dies stellt eine ganz spezielle Möglichkeit dar, die Lösungsverfeinerung durchzuführen. Die Tiefensuche mit schrittweiser Tiefenvergrößerung ist eine Suchstrategie, bei der eine Tiefensuche mit begrenzter Suchtiefe durchgeführt wird. Ist diese Tiefensuche erfolglos, so wird die maximale Suchtiefe schrittweise erhöht, und die Tiefensuche wird wiederholt. Diese Strategie stellt einen guten Kompomiß zwischen Rechenzeit und Speicherkosten bei der Suche dar [Korf, 1985a].

Der in Abbildung 5.16 dargestellte Algorithmus realisierte die Lösungsverfeinerung eines abstrakten Falles $F_a = ((s_0^a, s_m^a), \langle o_1^a, \ldots, o_m^a \rangle)$ zur Lösung des konkreten Problems $p = (s_0^c, s_n^c)$. Dieser Algorithmus ist in drei, teilweise rekursive Prozeduren aufgeteilt.

Die Prozedur $refine_{DFID}$ bekommt als Eingabe den konkreten Anfangszustand s_I^c, den konkreten Zielzustand s_G^c, die Folge von abstrakten Zwischenzuständen $S^a = (s_1^a, \ldots, s_k^a)$, die aus dem abstrakten Fall abgeleitet wurden, sowie die Menge α^*, die die Zustandsabstraktionsabbildung bestimmt. Diese Prozedur erhöht die maximale Suchtiefe für die eigentliche Tiefensuchprozedur $search_{bounded}$ bis zu einer maximalen Tiefe $Deep_{Max}$. Das Ziel für diese Tiefensuche ist entweder ein abstrakter Zustand (der erste Zustand in der Liste S^a) oder der konkrete Zielzustand, falls bereits alle abstrakten Zwischenzustände abgearbeitet sind. Durch die Anwendung der verfügbaren konkreten Operatoren und den rekursiven Aufruf der Suchprozedur wird die Tiefensuche realisiert.

5.4.5 Aufwand zur Lösungsverfeinerung

Eine Aufwandsabschätzung für die Lösungsverfeinerung haben wir bereits in Abschnitt 3.3.4 vorgenommen. Hiernach ergeben sich die Kosten für die Lösungsverfeinerung zu $C'_S(b, l_1, \ldots, l_k) = C_S(b, l_1) + \cdots + C_S(b, l_k)$, wobei b der Verzweigungsgrad des Suchraumes, $l_1, \ldots, l_k$ die Längen der entstandenen Teillösungen bei der Verfeinerung und $C_S(b, l) = (b^{l+1} - 1)/(b - 1)$ die Kosten zur Lösung eines Teilproblemes sind. Diese Aufwandsabschätzung basiert auf der Betrachtung der Größe des zu durchsuchenden Raumes und gilt daher in gleicher Weise auch für den zuvor angegebenen Algorithmus. Aufgrund der

Eingabe: Ein abstrakter Fall $F_a = ((s_0^a, s_m^a), \langle o_1^a, \ldots, o_m^a \rangle)$, ein konkretes Problem $p = (s_0^c, s_n^c)$, eine Zustandsabstraktionsabbildung α mit $\alpha(s_0^c) = s_0^a$ und $\alpha(s_n^c) = s_m^a$, sowie eine Suchtiefenbegrenzung $Deep_{Max}$.

Ausgabe: Eine Lösung $\bar{o}$ für p

procedure $refine((s_0^c, s_n^c), ((s_0^a, s_m^a), \langle o_1^a, \ldots, o_m^a \rangle))$
for $i := 1$ **to** $m-1$ **do** $s_i^a := (s_{i-1}^a \setminus Del_{o_i^a}) \cup Add_{o_i^a}$
$\alpha^* := s_0^a \cup \bigcup_{i=1}^{m} Add_{o_i^a}$
return $refine_{DFID}(s_I^c, (s_1^a, \ldots, s_{m-1}^a), \alpha^*, s_G^c)$

procedure $refine_{DFID}(s_I^c, S^a, \alpha^*, s_G^c)$
$Deep := 0$
repeat
  $search_{bounded}(s_I^c, S^a, \alpha^*, s_G^c, Deep)$
  if $search_{bounded}$ returns success(p) **then return** success(p)
  $Deep := Deep + 1$
until $Deep = Deep_{Max}$
return failure

procedure $search_{bounded}(s_I^c, S^a, \alpha^*, s_G^c, Deep)$
if $S^a = ()$ *(* Keine weiteren abstrakten Ziele: Teste konkretes Ziel *)*
  and $s_I^c = s_G^c$ **then return** success$(())$
if $S^a = (s_1^a, \ldots, s_k^a)$ *(* Wenigstens ein abstraktes Ziel *)*
  and for all $e \in s_1^a$ gilt: $\mathrm{SLD}(\mathrm{s_I^c} \cup \mathcal{R}_\mathrm{c} \cup \mathcal{A}, \mathrm{e}) \neq \emptyset$
  and for all $e \in \alpha^* \setminus s_1^a$ gilt: $\mathrm{SLD}(\mathrm{s_I^c} \cup \mathcal{R}_\mathrm{c} \cup \mathcal{A}, \mathrm{e}) = \emptyset$
  then *(* Abstrakter Zustand erreicht: Verfeinere nächsten abstrakten Operator *)*
    $refine_{DFID}(s_I^c, (s_2^a, \ldots, s_k^a), \alpha^*, s_G^c)$
    if $refine_{DFID}$ returns success(p) **then return** success(p)
if $Deep = 0$ **then return** failure
(Wende Operatoren zur Suche an *)*
for all $o^c \in \mathcal{O}_c$ **do**
  $\Omega = \mathrm{SLD}(s_I^c \cup \mathcal{R}_c, Pre_{o^c})$
  for each $\sigma \in \Omega$ **do**
    $s_{new}^c := (s_I^c \setminus (Del_{o^c}\sigma)) \cup (Add_{o^c}\sigma)$
    $search_{bounded}(s_{new}^c, S^a, \alpha^*, s_G^c, Deep - 1)$
    if $search_{bounded}$ returns success(p) **then return** success$((o^c\sigma) \circ p))$
return failure

Abbildung 5.16: Algorithmus zur Lösungsverfeinerung mit Backtracking

iterativen Tiefenvergrößerung wird dieser Suchraum zwar mehrfach abgearbeitet, jedoch ändert dies nichts an der Komplexität des Algorithmus. Diese ergibt sich zu $O(b^{l_1}+\cdots+b^{l_k})$.

Darüber hinaus haben jedoch noch eine Menge weiterer Faktoren einen Einfluß auf die Rechenzeit, die für die Lösungsverfeinerung benötigt wird. Dies sind

- die Kosten zum Test der Vorbedingung von konkreten Operatoren,
- die Anzahl der vorkommenden abstrakten essentiellen Sätze und
- die Kosten zur Ableitung eines abstrakten essentiellen Satzes durch die generische Abstraktionstheorie.

Diese Kosten haben einen linearen Einfluß auf den Aufwand zur Lösungsverfeinerung. Sie sind abhängig von der Formulierung der konkreten und der abstrakten Problemlösedomäne und sollten daher bei ihrer Entwicklung beachtet werden. Diesen Punkt werden wir in Abschnitt 8.1 erneut aufgreifen.

Beispiel 5.10 Abschließend wollen wir noch den Aufwand für die Lösungsverfeinerung der beiden in Beispiel 5.6 gezeigten abstrakten Fälle bei der Lösung des Problems $(\{\texttt{wert(2)}\},\{\texttt{wert9)}\})$ miteinander vergleichen. Hierzu nehmen wir exemplarisch an, daß ein Verzweigungsgrad von $b=5$ vorliegt, d. h. daß in jedem konkreten Zustand, neben dem `erhöhe_wert`-Operator, noch vier weitere Operatoren anwendbar sind. Der abstrakte Fall mit dem `abs_erhöhe_sprunghaft`-Operator enhält nur einen einzigen abstrakten Operator. Die Verfeinerung führt zu einem Plan, der aus 7 `erhöhe_wert`-Operatoren besteht. Daher ergeben sich die Kosten $C'_S(5,7)=97656$. Dieselben Kosten entstehen, wenn das Problem durch vollständige Suche, d. h. ohne Wiederverwendung eines Falles, gelöst wird. Verfeinern wir jedoch die Lösung des abstrakten Plans mit den beiden `abs_erhöhe_wert`-Operatoren (vgl. Beispiel 5.8), so ergeben sich die Kosten $C'_S(5,2,5)=C'_S(5,2)+C'_S(5,5)=3937$. Wir erkennen hier also einen deutlichen Vorteil durch den Einsatz dieses abstrakten Falls.

5.4.6 Bemerkungen über die Länge von Plänen

Wie bereits in der Einleitung erwähnt, liegt der Fokus dieser Arbeit auf der Effizienz des Problemlösens. Wir haben daher immer die Aufgabe betrachtet, überhaupt einen Plan für ein Problem zu finden. Die Qualität dieser Pläne spielte dabei bislang keine Rolle. An dieser Stelle wollen wir jedoch noch eine Bemerkung über die Länge der ermittelten Lösungen machen, da diese üblicherweise auf die Qualität der Lösung einen Einfluß hat. Ein Plan mit einer geringeren Anzahl von Operatoren ist in der Regel „besser“ als ein Plan mit einer größeren Operatorzahl. Dies hängt mit den Kosten zusammen, die bei der Ausführung eines Plans (z. B. der konkreten Fertigung eines Werkstückes) entstehen. Viele Suchverfahren zur Lösung von Problemen versuchen daher, eine kürzeste Lösung[3] eines Problems zu finden.

[3]In der Regel gibt es mehrere Lösungen eines Problems. Eine Lösung ist ein kürzeste Lösung, wenn es keine Lösung mit einer geringeren Anzahl von Operatoren gibt. Es ist daher möglich, daß es mehrere kürzeste Lösungen gibt.

Auch die Suchstrategie der Tiefensuche mit schrittweiser Tiefenvergrößerung garantiert das Finden einer kürzesten Lösung eines Problems. Jedoch gilt dies nicht mehr für eine durch Verfeinerung eines abstrakten Falls enstandene Lösung. Die verwendete Suchstrategie garantiert zwar, daß die Lösungen der Teilprobleme kürzeste Lösungen sind. Hieraus folgt jedoch nicht, daß die durch Kombination der Teillösungen entstehende Gesamtlösung eine kürzeste Lösung ist.

Beispiel 5.11 Betrachten wir die in Beispiel 5.10 erörterte Verfeinerung der beiden abstrakten Fälle aus Beispiel 5.6. Nehmen wir weiter an, daß es einen weiteren konkreten Operator gibt, der das Problem $(\{\texttt{wert(2)}\}, \{\texttt{wert9)}\})$ in einem Schritt löst. Die Verfeinerung des abstrakten Falls mit den beiden `abs_erhöhe_wert`-Operatoren führt dann trotzdem zu einer aus sieben `erhöhe_wert`-Operatoren bestehenden Lösung. Dies ist keine kürzeste Lösung, obwohl für beide Teilprobleme kürzeste Lösungen bestimmt wurden. Die Verfeinerung des Falls mit dem `abs_erhöhe_sprunghaft`-Operator führt hingegen zu einer kürzesten Lösung, die nur den neuen Operator enthält. Der erste abstrakte Fall führt den Problemlöser somit auf einen „Umweg“, der eine unnötig lange Lösung zur Folge hat.

Kapitel 6

Fallgeneralisierung und Spezialisierung

In diesem Kapitel soll der Ansatz zur flexiblen Wiederverwendung von Fällen durch Generalisierung und Spezialisierung im Detail erörtert werden. Wir beschreiben somit die in Abbildung 6.1 grau hinterlegten Komponenten der PARIS Architektur. Wie im vorherigen Kapitel wollen wir auch hier zunächst eine präzisere Vorstellung von Fallgeneralisierung aufbauen und danach entsprechende Verfahren zur Fallgeneralisierung und -spezialisierung vorstellen. Zur Verdeutlichung der dargestellten Begriffe und Algorithmen verwenden wir in diesem Kapitel ein sehr einfaches Beispiel aus dem Bereich der Arbeitsplanung. Für komplexere Beispiele wird erneut auf Kapitel 8 verwiesen.

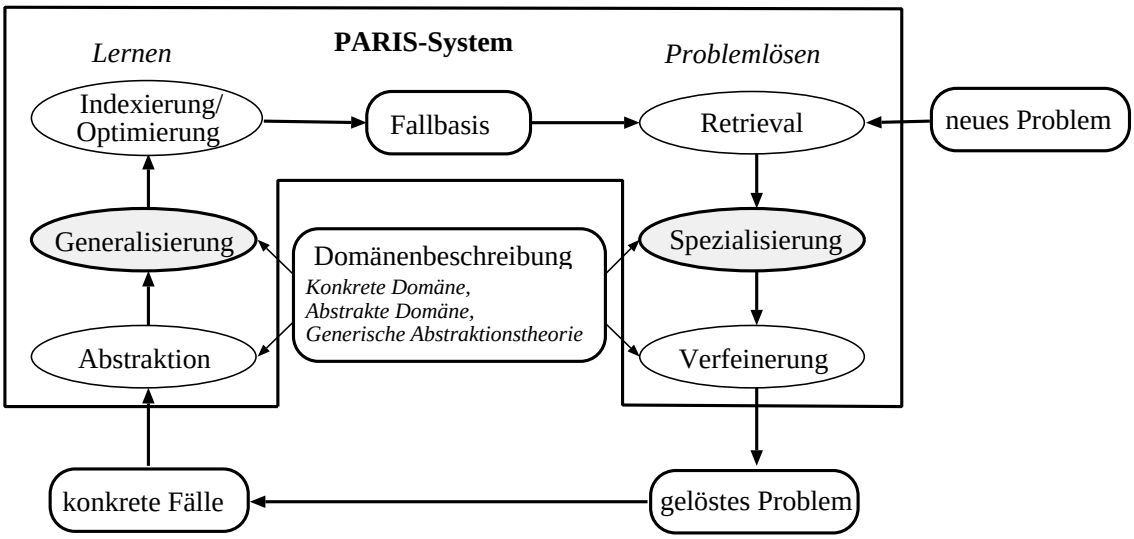

Abbildung 6.1: Die Architektur des PARIS Systems

6.1 Spezifikation der Fallgeneralisierung

Der hier vorgestellte Ansatz beschreibt die Generalisierung von Fällen auf der Grundlage einer Problemlösedomäne $\mathcal{D} = (\mathcal{L}, \mathcal{E}, \mathcal{O}, \mathcal{R})$ gemäß Definition 2.6. Hierbei ist es zunächt ohne Belang, ob es sich um eine abstrakte oder eine konkrete Problemlösedomäne handelt. Im folgenden führen wir einige grundlegende Definitionen ein, die uns letztlich zu dem Begriff des *generalisierten Falls* hinführen.

Eine Zustandsbeschreibung in einer Problemlösedomäne ist nach Definition 2.1 eine Menge von essentiellen Sätzen. Diese essentiellen Sätze werden durch Grundinstantiierung von essentiellen Satzformen gebildet, die eine Art Schema für eine Menge von essentiellen Sätzen sind. Ein *generalisierter Zustand* kann nun durch eine Menge von essentiellen Satzformen definiert werden, d. h. Variablen können vorkommen.

Definition 6.1 (Generalisierter Zustand) Eine generalisierte Zustandsbeschreibung $S(X)$ ist eine endliche Menge von teilweise instantiierten essentiellen Satzformen (siehe Bezeichnung 2.1), also $S(X) \subseteq \{e\sigma | e \in \mathcal{E} \text{ und } \sigma \text{ ist eine Substitution}\}$. Hierbei bezeichnet X genau die Menge der Variablen, die in den essentiellen Satzformen vorkommen.

Ein generalisierter Zustand steht für die Menge von Zuständen, die sich durch Instantiierung der vorkommenden Variablen ergeben. Die Definition der generalisierten Zustände wird nun benutzt, um den Begriff der *Problemklasse* zu definieren. Zunächst müssen wir jedoch noch eine Menge von Prädikaten auszeichnen, die bei der Definition einer Problemklasse benutzt werden dürfen.

Bezeichnung 6.1 (Menge zulässiger Prädikate) Wir zeichnen eine endliche Menge $\mathcal{M}$ von Prädikatssymbolen der Sprache $\mathcal{L}$ aus, die wir *zulässige* Prädikate nennen.

Bemerkung 6.1 Die Menge der zulässigen Prädikate werden wir im Verlauf dieses Kapitels dazu benutzen, um ein Operationalitätskriterium für das erklärungsbasierte Lernen zu definieren (siehe Abschnitt 6.4.3).

Definition 6.2 (Problemklasse) Eine Problemklasse $P(X_P)$ ist ein Tripel $(S_I(X_I), S_G(X_G), C(X_C))$, bestehend aus:

(i) einem generalisierten Initialzustand $S_I(X_I)$,
(ii) einem generalisierten Zielzustand $S_G(X_G)$ und
(iii) einer endlichen Konjunktion von Atomformeln, die aus Prädikaten aus $\mathcal{M}$ aufgebaut sind und mit $C(X_C)$ (*Constraints*) bezeichnet werden.

Hierbei bezeichnen X_I und X_G die Menge der Variablen, die im generalisierten Initial- bzw. Zielzustand vorkommen und X_C bezeichnet die Menge der Variablen, die in den Constraints vorkommen. Die Menge X_P bezeichnet die Menge aller Variablen der Problemklasse: $X_P = X_I \cup X_G \cup X_C$.

Bemerkung 6.2 Die Variablen im generalisierten Initial- und Zielzustand sowie in den Constraints brauchen hierbei nicht disjunkt zu sein. Die Constraints dienen dazu, genauere Einschränkungen an die vorkommenden Variablen zu beschreiben. In den Constraints dürfen daher auch Variablen vorkommen, die nicht im Initial- und Zielzustand enthalten sind. Wir werden hierauf später nochmal eingehen.

Eine Problemklasse beschreibt genau die Menge der Probleme, die sich ergeben, wenn die Variablen im generalisierten Initial- und Zielzustand so instantiiert werden, daß alle Constraints erfüllt sind:

Definition 6.3 (Probleme der Problemklasse) Ein Problem $p = (s_I, s_G)$ fällt genau dann in die Problemklasse $P(X) = (S_I(X_I), S_G(X_G), C(X_C))$ wenn es eine Substitution σ gibt, so daß gilt:

(i) $s_I = S_I(X_I)\sigma$ und
(ii) $s_G = S_G(X_G)\sigma$ und
(iii) $\mathcal{R} \vdash C(X_C)\sigma$.

Während eine Problemklasse eine Menge von Problemen repräsentiert, beschreibt eine *Lösungsklasse* eine Menge von Lösungen (Plänen). Die Lösungsklasse unterscheidet sich von einer Lösung genau dadurch, daß nicht alle Parameter der Operatoren durch Grundinstanzen belegt sein müssen. Vielmehr können in den Operatoren Variablen für bestimmte Parameter vorkommen.

Definition 6.4 (Lösungsklasse) Eine Lösungklasse $\bar{O}(X_O)$ ist eine endliche Folge von partiell instantiierten Operatoren $\langle o_1(t_{1,1}, \ldots, t_{1,k_1}), \ldots, o_n(t_{n,1}, \ldots, t_{n,k_n})\rangle$. Hierbei ist X_O genau die Menge aller Variablen, die in den Termen $t_{i,j}$ vorkommen.

Bemerkung 6.3 Eine Lösungsklasse umfaßt genau die Menge der Lösungen, die sich durch beliebige Grundinstantiierung aller Variablen der Lösungsklasse ergeben. Man beachte, daß nicht jede Instantiierung der Variablen zu einer (korrekten) Lösung führt.

Während ein Fall eine Beziehung zwischen einem speziellen Problem und einer speziellen Lösung herstellt, stellt ein *generalisierter Fall* eine Beziehung zwischen einer Problemklasse und einer Lösungsklasse her.

Definition 6.5 (Generalisierter Fall) Ein generalisierter Fall GF ist ein Paar $(P(X_P), \bar{O}(X_O))$, bestehend aus einer Problemklasse $P(X_P)$ und einer Lösungsklasse $\bar{O}(X_O)$.

Bemerkung 6.4 Die Variablenmengen X_P und X_O brauchen hierbei *nicht* disjunkt zu sein. Kommt eine Variable aus der Lösungsklasse auch in der Problemklasse vor, so beschreibt dies eine Abhängigkeit zwischen einem Problem der Problemklasse und einer Lösung der Lösungsklasse.

Wir können nun den Begriff der *Korrektheit* eines generalisierten Falles wie folgt definieren:

Definition 6.6 (Korrektheit eines generalisierten Falles) Ein generalisierter Fall $GF = (P(X_P), \bar{O}(X_O))$ mit der Problemklasse $P(X_P) = (S_I(X_I), S_G(X_G), C(X_C))$ ist genau dann korrekt in Bezug auf eine Problemlösedomäne, wenn für jede Grundsubstitutionen σ, die alle Variablen in X_P und X_O instantiiert, gilt: falls $\mathcal{R} \vdash C(X_C)\sigma$ gilt, so folgt $S_I(X_I)\sigma \overset{\bar{O}(X_O)\sigma}{\longrightarrow} S_G(X_G)\sigma$.

Bemerkung 6.5 Wird ein neues Problem p vorgegeben und liegt ein korrekter generalisierter Fall $(P(X_P), \bar{O}(X_O))$ vor, dessen Problemklasse $P(X_P)$ das Problem p umfaßt, so existiert eine Instantiierung τ der Variablen X_P, so daß $\mathcal{R} \vdash C(X_C)\tau$ gilt. Wird τ zu einer beliebigen Grundinstantiierung σ aller Variablen in X_P und X_O erweitert, so ist gesichert, daß $\bar{O}(X_O)\sigma$ eine korrekte Lösung von p ist.

Abschließend bleibt noch festzulegen, wann ein generalisierter Fall GF eine *Generalisierung* eines Falls F ist. Hierzu wollen wir lediglich fordern, daß der generalisierte Fall GF mindestens den Fall F umfaßt.

Definition 6.7 (Generalisierung, korrekte Generalisierung von Fällen) Ein generalisierter Fall $GF = (P(X_P), \bar{O}(X_O))$ mit der Problemklasse $P(X_P) = (S_I(X_I), S_G(X_G), C(X_C))$ ist eine *Generalisierung* eines Falles $F = (p, \bar{o})$, wenn es eine Grundsubstitution σ gibt, die alle Variablen in X_P und X_O so belegt, daß gilt:

(i) p ist ein Problem der Problemklasse $P(X_P)$ mit $(S_I(X_I)\sigma, S_G(X_G)\sigma) = p$ und
(ii) $\bar{O}(X_O)\sigma = \bar{o}$.

GF wird *korrekte Generalisierung* von F genannt wenn GF zusätzlich korrekt ist.

Beispiel 6.1 Die zuvor eingeführten Begriffe wollen wir nun an einem einfachen Beispiel aus dem Bereich der Arbeitsplanung verdeutlichen. Hierzu führen wir Beispiel 2.4 aus Kapitel 2 fort. Wir setzen die beiden Operatoren `chuck` und `cut` voraus, die wie folgt beschrieben sind:

Operator: `chuck(X)`
  Pre: `chuck_pos(none)`
  Del: { `chuck_pos(none)` }
  Add: { `chuck_pos(X)` }

Operator: `cut(X)`
  Pre: `chuck_pos(Y)` $\wedge$ `area(X,unpro)` $\wedge$ `Y` $\neq$ `none` $\wedge$ `no_overlap(X,Y)`
  Del: { `area(X,unpro)` }
  Add: { `area(X,pro)` }

Der `chuck`-Operator beschreibt das Aufspannen eines Werkstücks auf der Seite, die durch den Parameter angegeben ist. Der `cut`-Operator beschreibt die Bearbeitung einer Seite des Werkstücks (Abtragen von Material). Der in der Beschreibung dieses Operators vorkommende essentielle Satz `area(X,Y)` gibt an, ob die Seite `X` noch unbearbeitet ist (`Y =`

unpro) oder ob sie noch bearbeitet werden muß (Y = pro).[1] Außerden setzen wir noch die folgende universelle Implikation voraus, die das Prädikat no_overlap beschreibt:

$$\texttt{no_overlap(X,Y)} \leftarrow \texttt{X} \neq \texttt{Y}.$$

In dieser Domäne wollen wir nun den nachfolgenden Fall F betrachten:[2]

$$F = ((\{\texttt{chuck_pos(none)}, \texttt{area(r,unpro)}\}, \{\texttt{chuck_pos(l)}, \texttt{area(r,pro)}\}),\\ \langle\ \texttt{chuck(l)}, \texttt{cut(r)}\ \rangle).$$

Er beschreibt die Situation, daß man ein Werkstück auf der rechten Seite dadurch bearbeiten kann, daß man es auf der linken Seite einspannt (chuck(l)) und danach das Material auf der rechten Seite abträgt (cut(r)). Ein generalisierter Fall gemäß Definition 6.5 kann z. B. wie folgt aussehen:

$$GF = ((\{\texttt{chuck_pos(none)}, \texttt{area(X,unpro)}\}, \{\texttt{chuck_pos(Y)}, \texttt{area(X,pro)}\},\\ \texttt{Y} \neq \texttt{none} \wedge \texttt{no_overlap(X,Y)}),\\ \langle\ \texttt{chuck(Y)}, \texttt{cut(X)}\ \rangle).$$

In diesem generalisierten Fall ist die Menge der Variablen der Lösungsklasse (X und Y) vollständig in der Menge der Variablen der Problemklasse enthalten. Ist ein neues Problem der Problemklasse gegeben, so sind daher alle Variablen instantiiert und somit eine vollständige Lösung bestimmt. Darüber hinaus ist GF korrekt. Dies bedeutet, daß jede Instantiierung der Variablen X und Y, für die $\texttt{Y} \neq \texttt{none} \wedge \texttt{no_overlap(X,Y)}$ gilt, zu einem Fall führt, d. h. der Plan, der sich durch Instantiierung der Lösungsklasse ergibt, ist eine korrekte Lösung des Problems, das durch Instantiierung der Problemklasse entsteht. Außerdem ist GF eine korrekte Generalisierung von F. Ein weiteres, komplexeres Beispiel für einen generalisierten Fall werden wir in Abschnitt 8.3.2 beschreiben.

6.2 Erklärungsbasiertes Lernen zur Fallgeneralisierung

Wir wollen hier zunächst eine abstrakte Vorstellung davon vermitteln, wie erklärungsbasiertes Lernen zur Generalisierung von Fällen eingesetzt werden kann, bevor dieser Ansatz in den folgenden Abschnitten detaillierter entwickelt wird. Das Grundprinzip des erklärungsbasierten Lernens haben wir bereits in Abschnitt 3.2 erörtert und dabei die wesentlichen Begriffe eingeführt. Anhand dieser Begriffe wollen wir nun die Anwendung von erklärungsbasiertem Lernen zur Generalisierung eines Falls $F = ((s_I, s_G), \bar{o})$ erläutern.

[1] Hierbei steht unpro für unprocessed, und pro steht für processed.

[2] Hierbei steht r für die rechte Seite und l für die linke Seite.

Zielkonzept: Als Zielkonzept wird ein generalisierter Zielzustand $S_G(X_G)$ verwendet. Dieser wird so gewählt, daß der Zielzustand s_G des Falles eine Instantiierung hiervon ist.

Trainingsbeispiel: Der Initialzustand s_I ist ein Beispiel für einen Zustand, von dem aus der Zielzustand s_G durch eine Operatorfolge erreicht werden kann.

Domänentheorie: Die Problemlösedomäne kann als Domänentheorie herangezogen werden. Sie beschreibt hierbei allgemein, welche essentiellen Sätze eines Zustandes s_i, der sich durch die Anwendung des Operators o_i im Zustand s_{i-1} ergibt, gültig sind.

Operationalitätskriterium: Eine Konzeptspezialisierung wird genau dann als operational angesehen, wenn sie nur Atome der folgenden Art enthält:

(i) essentielle Satzformen, die den Initialzustand des Beispiels beschreiben,

(ii) Atomformeln, die Prädikate aus $\mathcal{M}$ enthalten und folglich als Constraints in der Problemklasse des generalisierten Falls vorkommen können.

Operationale Konzeptspezialisierung: Die operationale Konzeptspezialisierung, die das erklärungsbasierte Lernen als Ergebnis liefert, ist eine hinreichende Bedingung für das Erreichen des generalisierten Zielzustandes S_G. Diese operationale Konzeptspezialisierung ist eine Konjunktion von essentiellen Satzformen des Initialzustandes und Atomformeln, die Prädikate aus $\mathcal{M}$ enthalten.

Der Prozeß der erklärungsbasierten Fallgeneralisierung kann nun grob in drei Phasen untergliedert werden:

Phase-I: Generierung der Fallerklärung: Zunächst wird eine Erklärung für den vorliegenden Fall aufgebaut, die nachweist, daß das Trainingsbeispiel das Zielkonzept erfüllt. Diese Erklärung begründet, auf der Grundlage der Problemlösedomäne, daß der Zielzustand, vom Initialzustand durch die Anwendung der Operatorfolge erreicht wird.

Phase-II: Generalisierung der Fallerklärung: Die zuvor gebildete Fallerklärung wird generalisiert und die operationale Konzeptspezialisierung wird gebildet.

Phase-III: Bildung des generalisierten Falles Die operationale Konzeptspezialisierung wird in einen generalisierten Fall transformiert.

Die nachfolgenden Abschnitte beschreiben diese drei Phasen im Detail.

6.3 Phase-I: Generierung einer Fallerklärung

6.3.1 Der Begriff der Erklärung

Im folgenden wollen wir den Begriff der Erklärung, der bereits in der Begriffsbestimmung 3.9 eingeführt wurde, näher definieren, da er die zentrale Datenstruktur für das erklärungs-

basierte Lernen darstellt. Eine Erklärung dient dem Nachweis einer oder mehrerer Aussagen (des Zielkonzeptes) auf der Grundlage einer Domänentheorie. Die Erklärung repräsentiert das Zusammenwirken mehrerer Regeln beim Nachweis der Gültigkeit von Aussagen.

Definition 6.8 (Erklärung) Eine Erklärung Λ ist ein zusammenhängender azyklischer gerichteter Graph. Knoten, zu denen keine Kanten hinführen, werden als *Basisknoten* bezeichnet; alle anderen Knoten werden *innere Knoten* genannt. Die Knoten, von denen eine gerichtete Kante zu einem inneren Knoten K hinführt, werden *direkte Vorgänger von* K genannt (K wird auch *direkter Nachfolger* genannt). Knoten, von denen keine Kanten wegführen, werden als *Zielknoten* bezeichnet. Jeder Knoten K des Graphen ist mit einem Grundatom p markiert (Schreibweise: $\overline{K} \equiv p$), das *Aussage* genannt wird. Jeder innere Knoten K, mit den direkten Vorgängern $K_1, \ldots, K_n$, besitzt zusätzlich eine Markierung (*Begründung* genannt) der Form $P \leftarrow P_1, \ldots, P_n$, wobei P und $P_1, \ldots, P_n$ Atome sind und für die es eine Substitution σ gibt mit $P\sigma = \overline{K}$ und $P_i\sigma = \overline{K_i}$ für alle $i \in \{1, \ldots, n\}$. Wir schreiben auch: $\widehat{K} \equiv P \leftarrow P_1, \ldots, P_n$.

In einer Erklärung wird jede Aussage, mit der ein Knoten markiert ist, durch die Aussagen an den Knoten gerechtfertigt, von denen gerichtete Kanten zu diesem Knoten hinführen (siehe Abbildung 6.2). Eine Aussage an einem inneren Knoten ist immer eine Instantiierung eines allgemeineren Zusammenhangs, der als Begründung zu den gerechtfertigten Knoten notiert wird.

Bemerkung 6.6 Die in einer Erklärung vorkommenden Begründungen werden zwar wie eine Hornklausel notiert, haben jedoch nicht notwendigerweise ihre Semantik. Dies hängt damit zusammen, daß wir für die in Erklärungen vorkommenden Formeln nicht die übliche prädikatenlogische Interpretation zugrunde legen müssen. Wir gehen lediglich davon aus, daß wir einer Aussage p an einem Knoten K einen Wahrheitswert zuweisen können.

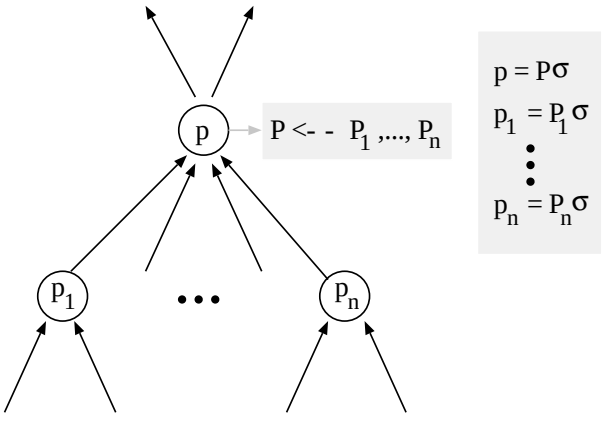

Abbildung 6.2: Ausschnitt aus einem Erklärungsgraphen

Definition 6.9 (Gültigkeit von Begründungen und Erklärungen) Sei Λ eine Erklärung. Für jeden Knoten K aus Λ sei weiterhin W_K eine Funktion (Wahrheitsfunktion), die einem Grundatom einen Wahrheitswert aus $\{wahr, falsch\}$ zuordnet. Sind $K_1, \ldots, K_n$ die direkten Vorgänger von K, dann heißt die Begründung $\widehat{K} \equiv P \leftarrow P_1, \ldots, P_n$ *gültig* bezüglich der Wahrheitsfunktionen W_K genau dann wenn für jede Grundsubstitution σ, die alle Variablen in $P \leftarrow P_1, \ldots, P_n$ belegt, folgendes gilt: falls $\forall_{i=1,\ldots,n}\ W_{K_i}(P_i\sigma) = wahr$ gilt, dann gilt auch $W_K(P\sigma) = wahr$. Eine Erklärung ist *gültig* genau dann wenn alle Begründungen gültig sind.

Bemerkung 6.7 Wir erlauben es, jedem Knoten einer Erklärung eine eigene Wahrheitsfunktion zuzuordnen. Dies werden wir später für Fallerklärungen benötigen (vgl. Abschnitt 6.3.3).

Lemma 6.1 (Gültigkeit der Aussagen einer Erklärung) Sei Λ eine gültige Erklärung bzgl. der Wahrheitsfunktionen W_K und seien $K_1, \ldots, K_n$ die Basisknoten von Λ. Es gilt: falls $\forall_{i=1,\ldots,n}\ W_{K_i}(\overline{K_i}) = wahr$ gilt, so gilt $W_K(\overline{K}) = wahr$ für jeden Knoten K aus Λ.

Beweisskizze Falls $\forall_{i=1,\ldots,n}\ W_{K_i}(\overline{K_i}) = wahr$ gilt, so gilt für jeden direkten Nachfolger K von K_i ebenfalls $W_K(\overline{K}) = wahr$ aufgrund von Definition 6.8 und 6.9. Diesen Schluß kann man fortsetzen, bis die Gültigkeit der Aussage für die Zielknoten der Erklärung gezeigt ist.

Wir wollen im folgenden diesen Erklärungsbegriff am Beispiel der Hornlogik erläutern.

Beispiel 6.2 Sei Σ eine Menge von Hornklauseln. Der Beweis dafür, daß ein Grundatom F aus Σ folgt, läßt sich auf einfache Weise als Erklärung darstellen. Hierbei entsprechen die Aussagen Grundatomen in der Hornlogik und Begründungen sind Hornklauseln aus Σ. Wir setzen nun folgende Wahrheitsfunktion voraus: $W_K(F) = wahr$ gdw $\Sigma \vdash F$. Diese Wahrheitsfunktion soll für alle Knoten verwendet werden. Eine Erklärung Λ, die nur Begründungen aus Σ enthält, ist gültig bzgl. W_K. Daher läßt sich Lemma 6.1 wie folgt anwenden: Sind $K_1, \ldots, K_n$ die Basisknoten von Λ, so gilt für jeden Knoten K aus Λ: $\Sigma \cup \{\overline{K_1}\} \cup \ldots \cup \{\overline{K_n}\} \vdash \overline{K}$. Insbesondere gilt: sind $K_1^*, \ldots, K_m^*$ die Zielknoten der Erklärung, so gilt: $\Sigma \cup \{\overline{K_1}\} \cup \ldots \cup \{\overline{K_n}\} \vdash (\overline{K_1^*} \wedge \ldots \wedge \overline{K_m^*})$

Bemerkung 6.8 Anhand von Beispiel 6.2 lassen sich bereits einige der klassischen Begriffe des erklärungsbasierten Lernens präzisieren. Hierbei ist $\{\overline{K_1}\} \cup \ldots \cup \{\overline{K_n}\}$ das *Trainingsbeispiel* B und $\overline{K_1^*} \wedge \ldots \wedge \overline{K_m^*}$ ist eine durch das Trainingsbeispiel erzeugte spezifische Instantiierung des *Zielkonzeptes*.

6.3.2 Die Fallerklärung

Erklärungen gemäß Definition 6.8 können eingesetzt werden, um die Korrektheit eines Falles zu begründen. Hierzu muß für einen Fall eine spezielle *Fallerklärung* aufgebaut werden.

Begriffsbestimmung 6.1 (Fallerklärung) Eine Fallerklärung ist eine Erklärung gemäß Definition 6.8, die die Gültigkeit jedes essentiellen Satzes im Zielzustand aufgrund der Anwendung der Operatorsequenz (Lösung) auf den Initialzustand nachweist und mit Wissen aus der Problemlösedomäne (Operatoren und universelle Implikationen) begründet.

Bemerkung 6.9 Wie bereits in Abschnitt 3.2.6 erwähnt wurde, ist ein durch erklärungsbasiertes Lernen gewonnenens Zielkonzept nie allgemeiner, als die zugrundeliegende Erklärung, d. h. die darin verwendeten Begründungen. Da wir in der Regel daran interessiert sind, einen möglichst allgemeinen, und damit möglichst wiederverwendbaren Fall zu generieren, sollten die Begründungen in der Erklärung so allgemein sein, wie es aufgrund der Domänentheorie möglich ist.

Aus Gründen der Übersichtlichkeit wird die aufzubauende Fallerklärung in zwei Arten von partiellen Erklärungen zerlegt: einer *Operatorerklärung* und einer *Vorbedingungserklärung.*

Begriffsbestimmung 6.2 (Operatorerklärung) Die Operatorerklärung ist eine Erklärung für die Zustandstransformation $s_i := s_{i-1} \setminus Del_{o_i} \cup Add_{o_i}$, die sich aufgrund der Ausführung des Operators o_i ergibt, d. h. eine Erklärung für die Gültigkeit der essentiellen Sätze in s_i.

Begriffsbestimmung 6.3 (Vorbedingungserklärung) Die Vorbedingungserklärung ist einer Erklärung dafür, daß die Vorbedingung eines Operators in dem Zustand, in dem dieser angewendet wird gilt, d. h. eine Erklärung für die Gültigkeit von $s_{i-1} \cup \mathcal{R} \vdash Pre_o$.

Die Fallerklärung enthält für jeden Operator im Plan eine zugehörige Operatorerklärung und eine Vorbedingungserklärung, die geeignet verbunden sind.

6.3.3 Verschiedene Knoten und Wahrheitsfunktionen

Die Fallerklärung wird aus drei verschiedenen Arten von Knoten aufgebaut, denen wir verschiedene Wahrheitsfunktionen zuordnen. Der Grund für ihre Einführung wird sich im Detail in den Abschnitten 6.3.4 bis 6.3.6 ergeben.

Zustandsknoten: Ein Zustandsknoten wird mit KS_i bezeichnet. Die Aussage, die einem Zustandsknoten zugeordnet ist, ist ein essentieller Satz des Zustands s_i. Für Zustandsknoten verwenden wir die Wahrheitsfunktion $W_{KS_i}(F) = wahr$ gdw $F \in s_i$.

Folgerungsknoten: Ein Folgerungsknoten wird mit KF_i bezeichnet. Die Aussage, die einem Folgerungsknoten zugeordnet ist, ist ein Atom, das sich im Zustand s_i durch Anwendung der universellen Implikationen ableiten läßt. Für Zustandsknoten verwenden wir die Wahrheitsfunktion $W_{KF_i}(F) = wahr$ gdw $\mathcal{R} \cup s_i \vdash F$.

Operatorknoten: Ein Operatorknoten wird mit KO_i bezeichnet. Die Aussage, die einem Operatorknoten zugeordnet ist, ist ein Atom der Form $Apl(o(t_1, \ldots, t_n))$. Dieses Atom besagt, daß der instantiierte Operator $o(t_1, \ldots, t_n)$ im Zustand s_{i-1} angewendet wird. Für Operatorknoten verwenden wir die Wahrheitsfunktion $W_{KO_i}(Apl(o(t_1, \ldots, t_n))) = wahr$ gdw $s_{i-1} \overset{o(t_1,\ldots,t_n)}{\longrightarrow} s_i$.

6.3.4 Aufbau der Operatorerklärung

Die Operatorerklärung weist die Gültigkeit jedes essentiellen Satzes, im Endzustand s_i (Zustand nach Operatorausführung) aufgrund der Gültigkeit der essentiellen Sätze im Ausgangszustand s_{i-1} (Zustand vor Operatorausführung) und der Operatorbeschreibung $(o(x_1, \ldots, x_n), Pre_o, Add_o, Del_o)$ für $o(x_1, \ldots, x_n)$ auf möglichst allgemeine Weise nach. Um die Erklärung so allgemein wie möglich zu gestalten, rechtfertigen die Begründungen die Effekte des uninstantiierten Operators $o(x_1, \ldots, x_n)$. Damit sind diese Begründungen für jede beliebige Instantiierung der Operatorparameter gültig.

Jeder essentielle Satz e im Ausgangs- und Endzustand wird durch einen Zustandsknoten in der Erklärung repräsentiert (siehe Abbildung 6.3). Die Knoten für den Ausgangszustand werden mit KS_{i-1} bezeichnet; die Knoten für den Endzustand werden mit KS_i bezeichnet. Die Zustandsknoten bekommen die entsprechenden essentiellen Sätze als Aussagen zugeordnet.

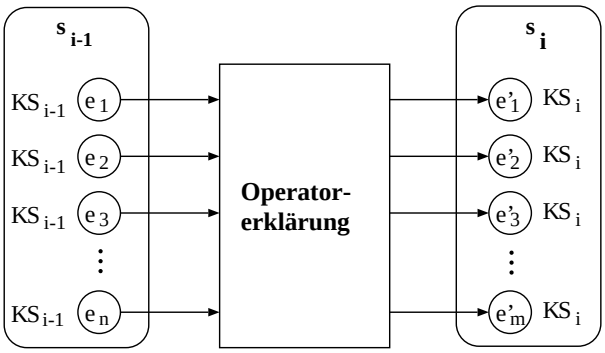

Abbildung 6.3: Schematische Darstellung einer Operatorerklärung

Für jeden Knoten im Endzustand wird eine eigene Erklärung aufgebaut. Drei verschiedene Varianten von Erklärungen kommen hierbei vor, abhängig davon, ob (a) der zu erklärende essentielle Satz durch den Operator hinzugefügt wurde, (b) durch der Operator „bedroht“ wird, oder (c) durch den Operator nicht betroffen ist. Diese Erklärungsarten werden wir nun im Detail erörtern.

Fall (a): Hinzugefügte essentielle Sätze

Wird ein essentieller Satz durch den Operator hinzugefügt, so hängt die Gültigkeit dieses Satzes von der Anwendbarkeit des Operators ab. Daher wird in die Operatorerklärung ein *Operatorknoten* KO_i eingefügt, der die Anwendung des Operators o_i mit seinen Parametern repräsentiert (siehe Abbildung 6.4).[3] Dieser Operatorknoten bekommt die Aussage $KO_i \equiv Apl(o(t_1, \ldots, t_n))$ zugeordnet. Hierbei ist Apl ein neues Prädikat. Vom Operatorknoten gehen gerichtete Kanten zu jedem Knoten hin, der einen durch diesen Operator hinzugefügten essentiellen Satz des Endzustandes repräsentiert. Jeder dieser Knoten besitzt eine Begründung der Form $A_i \leftarrow Apl(o(x_1, \ldots, x_n))$, wobei A_i der entsprechende essentielle Satz der Addliste Add_o ist.

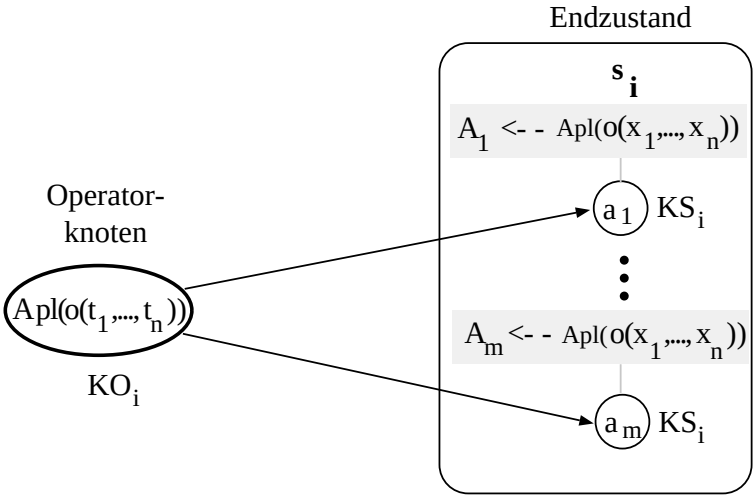

Abbildung 6.4: Operatorerklärung für hinzugefügte essentielle Sätze

Fall (b): Bedrohte essentielle Sätze

Die zweite Erklärungsvariante behandelt die essentiellen Sätze, die von der Deleteliste betroffen sein können. Ein essentieller Satz aus dem Ausgangszustand bleibt im Zielzustand gültig, falls er nicht in der instantiierten Deleteliste des Operators vorkommt. Ist ein essentieller Satz durch die Anwendung des instantiierten Operators $o(t_1, \ldots, t_n)$ gelöscht worden, so ist folglich auch kein Knoten im Endzustand vorhanden, so daß das Löschen von essentiellen Sätze nicht erklärt werden muß. Ein Operator $o(x_1, \ldots, x_n)$, der bei der gerade vorliegenden Instantiierung $o(t_1, \ldots, t_n)$ einen Satz nicht löscht, kann diesen jedoch

[3] Der Operatorknoten KO_i wird durch eine Vorbedingungserklärung gerechtfertigt (siehe Abschnitt 6.3.5).

möglicherweise bei einer abgeänderten Instantiierung löschen. Wir sagen daher, der essentielle Satz ist bedroht.[4] Da ein bedrohter Satz ggf. in der Vorbedingung von folgenden Operatoren benötigt wird, ist es erforderlich, ihn vor dem Löschen zu bewahren.

Definition 6.10 (Bedrohte essentielle Sätze) Jeder essentieller Satz e des Anfangszustands, der eine Grundinstanz der essentiellen Satzform E ist, ist durch den Operator $o(x_1, \ldots, x_n)$ *bedroht*, wenn es einen Satz D_j der Deleteliste Del_o gibt, so daß E und D_j unifizierbar sind.

Um eine möglichst allgemeine Erklärung aufzubauen, ist es erforderlich, die allgemeinste Schutzbedingung zu bestimmen, die bedrohten essentielle Sätze vor dem Löschen durch einen Operator schützt.

Definition 6.11 (Schutzbedingung für bedrohte essentielle Sätze) Sei e ein essentieller Satz des Anfangszustands, der eine Grundinstanz der essentiellen Satzform $P(s_1, \ldots, s_k)$ ist und durch den Operator $o(x_1, \ldots, x_n)$ bedroht wird. Seien weiterhin $P(s_{1,1}, \ldots, s_{1,k}), \ldots, P(s_{m,1}, \ldots, s_{m,k})$ genau die Sätze der Deleteliste des Operators, die mit $P(s_1, \ldots, s_k)$ unifizierbar sind. Die *Schutzbedingung für* $P(s_1, \ldots, s_k)$ *bei Ausführung von* $o(x_1, \ldots, x_n)$ ist folgende Konjunktion von Atomen: $f(s_1, \ldots, s_k) \neq f(s_{1,1}, \ldots, s_{1,k}) \wedge \ldots \wedge f(s_1, \ldots, s_k) \neq f(s_{m,1}, \ldots, s_{m,k})$. Hierbei ist f ein neues Funktionssymbol und $\neq$ ein 2-stelliges Prädikat, das die Ungleichheit von Termen ausdrückt.[5] Das Konjunktionsglied $f(s_1, \ldots, s_k) \neq f(s_{j,1}, \ldots, s_{j,k})$ wird mit $PROT_j(P(s_1, \ldots, s_k), o(x_1, \ldots, x_n))$ bezeichnet.

Bemerkung 6.10 In der Schutzbedingung für $P(s_1, \ldots, s_k)$ bei Ausführung von $o(x_1, \ldots, x_n)$ können nur die Variablen aus $P(s_1, \ldots, s_k)$ sowie die Variablen $x_1, \ldots, x_n$ vorkommen.

Ist diese Schutzbedingung erfüllt, so ist sichergestellt, daß der essentielle Satz e nicht durch die Operatorausführung gelöscht werden kann, da alle essentiellen Sätze in der Deleteliste von e verschieden sind.

Die Operatorerklärung für bedrohte essentielle Sätze e der Satzform E ist in Abbildung 6.5 dargestellt. Für jedes Konjunktionsglied der Schutzbedingung wird ein Folgerungsknoten KF_{i-1} eingeführt. Sei σ die Substitution, für die $e = E\sigma$ und $o(t_1, \ldots, t_n) = o(x_1, \ldots x_n)\sigma$ gilt. Der Folgerungsknoten KF_{i-1} ist ein Basisknoten in der Erklärung und bekommt die Aussage $PROT_j(E, o(x_1, \ldots, x_n))\sigma$ zugeordnet.

Die Begründung für den Knoten KS_i, der den bedrohten essentiellen Satz e im Zielzustand repräsentiert, ergibt sich wie folgt: $E \leftarrow E, Apl(o(x_1, \ldots, x_n)), PROT_1(E, o(x_1, \ldots, x_n)) \wedge \ldots \wedge PROT_m(E, o(x_1, \ldots, x_n))$.

[4] Der Begriff der Bedrohung kommt in verschiedenen Formen im Bereich der Planung vor. Er entspricht z. B. den *Threats*, die bei nicht-linearen Planern betrachtet werden.

[5] Diese Ungleichheit von Termen kann innerhalb der Hornlogik axiomatisiert werden (siehe [Lloyd, 1984], S. 70). Wir nehmen daher im folgenden an, daß eine solche Axiomatisierung vorliegt und beschreiben $\neq$ daher nicht näher. Wir setzen weiterhin voraus, daß $\neq$ ein zulässiges Prädikat ist, d. h. daß $\neq \in \mathcal{M}$ gilt.

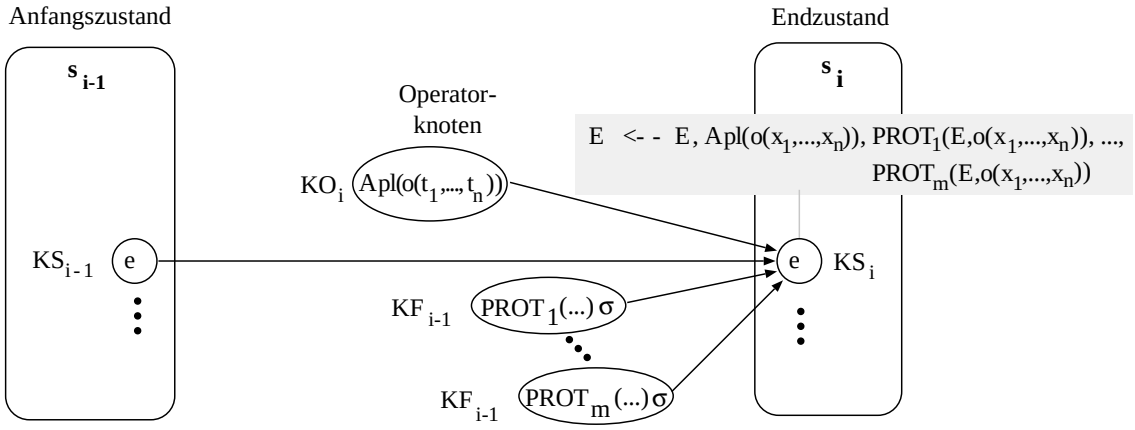

Abbildung 6.5: Operatorerklärung für bedrohte essentielle Sätze

Fall (c): Nicht-bedrohte essentielle Sätze

Die dritte Erklärungsvariante behandelt essentielle Sätze, die in keinem Fall vom Operator gelöscht werden können. Dies sind Sätze, die nicht hinzugefügt werden und auch nicht bedroht sind. Daher gilt ein solcher essentieller Satz im Endzustand genau dann, wenn er im Ausgangszustand bereits gültig war. Dies führt zu der Erklärung, die in Abbildung 6.6 dargestellt ist. Jeder Knoten, der einen solchen essentiellen Satz im Endzustand repräsentiert, erhält die Begründung $E \leftarrow E, Apl(o(x_1, \ldots, x_n))$ zugeordnet.

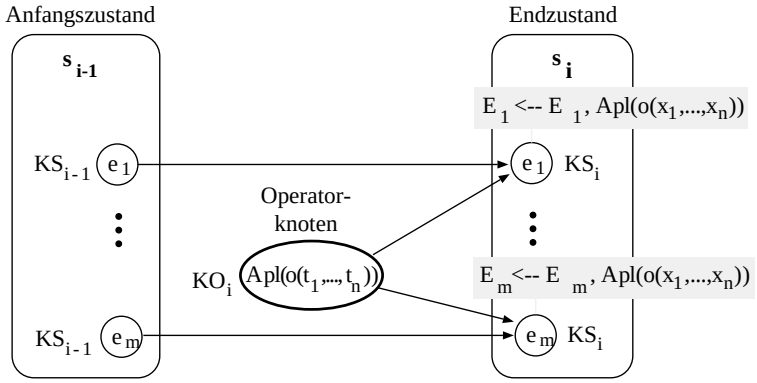

Abbildung 6.6: Operatorerklärung für nicht-bedrohte essentielle Sätze

Die vollständige Operatorerklärung, die sich durch das Zusammensetzen der beschriebenen Varianten ergibt, ist in Abbildung 6.7 nochmal zusammenfassend dargestellt. Wir führen noch folgende Bezeichnung ein:

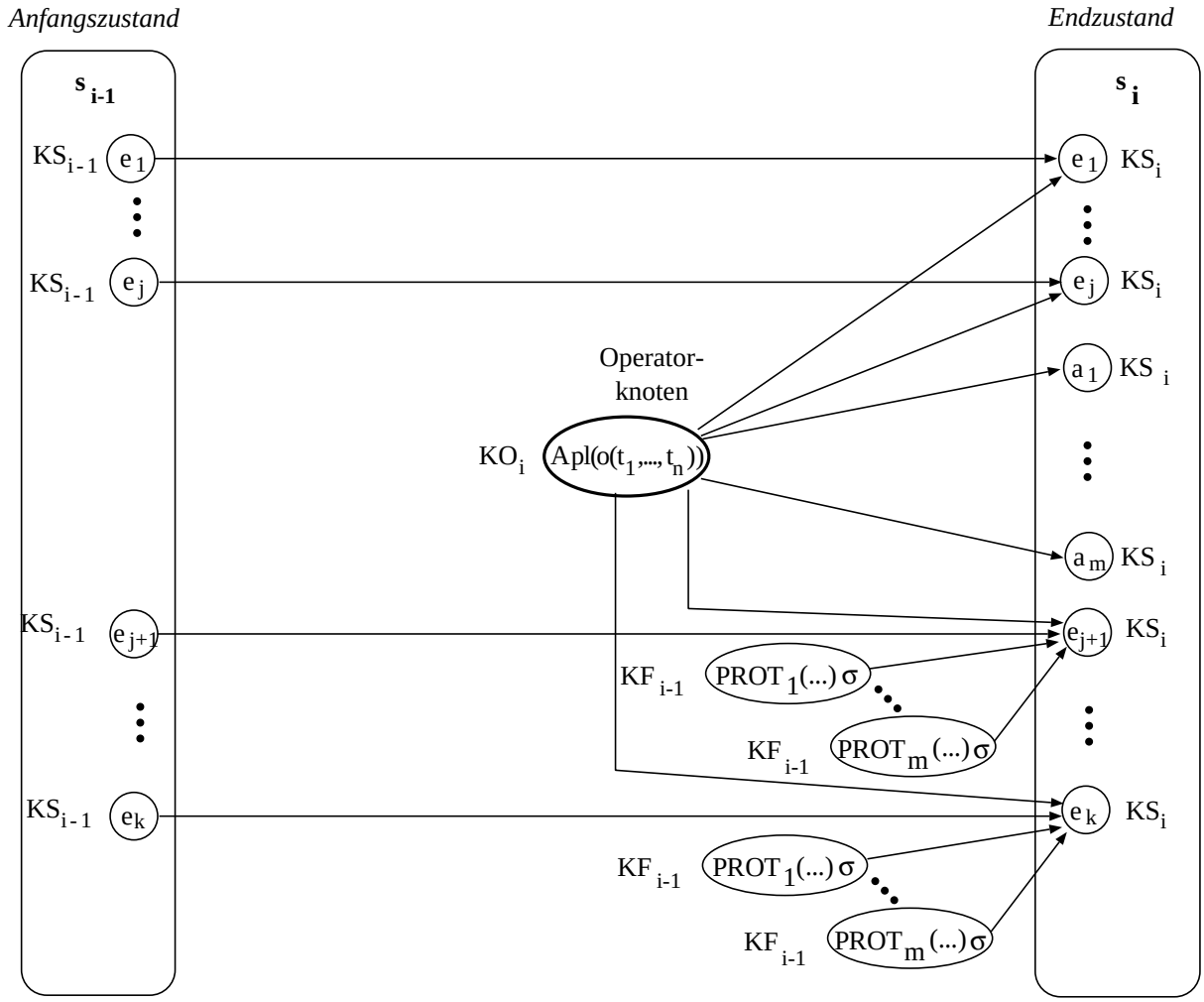

Abbildung 6.7: Detaillierte Darstellung einer Operatorerklärung

Bezeichnung 6.2 (Operatorerklärung) Eine Operatorerklärung für den Operator o wird mit $\Lambda_o(\mathcal{KS}_{i-1}, \mathcal{KS}_i, KO_i)$ bezeichnet, wobei $\mathcal{KS}_{i-1}$ die Menge der Zustandsknoten vom Typ KS_{i-1}, und $\mathcal{KS}_i$ ist die Menge der Zustandsknoten vom Typ KS_i ist. KO_i ist der Operatorknoten.

6.3.5 Aufbau der Vorbedingungserklärung

Die Vorbedingungserklärung ist eine Erklärung dafür, daß die Vorbedingung des Operators o_i im Zustand s_{i-1} erfüllt ist, also daß $s_{i-1} \cup \mathcal{R} \vdash Pre_o$ gilt. Da die Domänentheorie, die die Operatorvorbedingung definiert aus Hornregeln aufgebaut ist, wird die Gültigkeit der Vorbedingung mit Hilfe der SLD-Resolution nachgewiesen. Dies wurde bereits in den

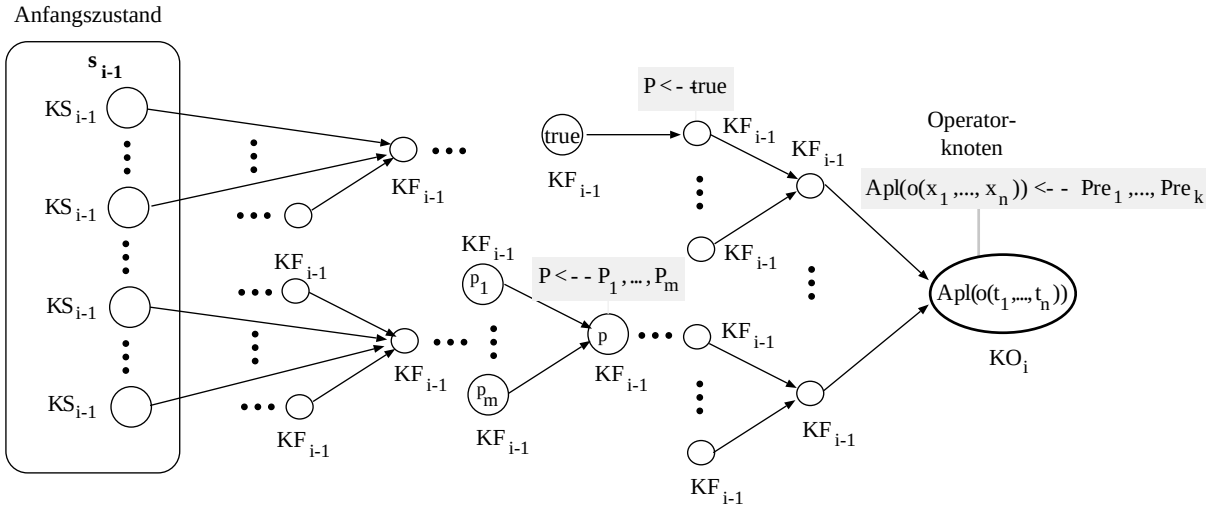

Abbildung 6.8: Schematische Darstellung einer Vorbedingungserklärung

Abschnitten 5.2.2 und 5.2.3 diskutiert. Hierbei hatten wir ein SLD-Resolutionsverfahren $\Pi = SLD(C, G)$ vorausgesetzt, daß mit Π eine Menge von Paaren, bestehend aus einer Antwortsubstitution zusammen mit einem Beweisbaum, zurückliefert. Wird diese SLD-Resolution zum Nachweis der Gültigkeit der Operatorvorbedingung verwendet, so kann der resultierende Beweisbaum direkt als Vorbedingungserklärung verwendet werden. Eine solche Vorbedingungserklärung ist schematisch in Abbildung 6.8 dargestellt. Diese Erklärung enthält für jedes Atom Pre_i in der Vorbedingung Pre_o des Operators einen entsprechenden Folgerungsknoten KF_{i-1}. Dieser Knoten besitzt als Aussage die Instantiierung des Atoms, die im Beweis der Vorbedingung des Operators vorkommt. Der Operatorknoten erhält als Begründung die allgemeinste Bedingung, die aufgrund der Operatordefinition möglich ist, nämlich: $Apl(o(x_1, \ldots, x_n)) \leftarrow Pre_1, \ldots, Pre_k$. Der Nachweis der einzelnen Atome aus der Vorbedingung erfolgt aufgrund der Hornregeln in $\mathcal{R}$, der Fakten in $\mathcal{R}$, bzw. der essentiellen Sätze in s_{i-1}. Jede angewendete Hornregel $P \leftarrow P_1, \ldots, P_m$ aus $\mathcal{R}$ wird dabei in einen Erklärungsteilgraphen umgesetzt, in den Kanten von den Knoten für die Prämissen $P_1, \ldots, P_m$ zu dem Folgerungsknoten für die Konklusion P gehen. Dieser Folgerungsknoten erhält hierbei als Aussage die im Beweis vorkommende Instantiierung von P und als Begründung die Hornklausel selbst zugeordnet. Basiert der Beweis auf einem essentiellen Satz e aus dem Zustand s_{i-1}, so wird direkt der entsprechende Zustandsknoten KS_{i-1} des Ausganszustandes verwendet. Basiert der Beweis auf einem Fakt P aus $\mathcal{R}$, so wird dieses wie die Hornregel $P \leftarrow true$ behandelt. Für die Prämisse $true$ wird ein Basisknoten mit der Aussage $true$ in die Erklärung eingeführt.[6] Wir führen noch folgende Bezeichnung für

[6] Man darf hier die Aussage $true$ nicht mit dem Wahrheitswert $wahr$ einer Wahrheitsfunktion gemäß Definition 6.9 verwechseln. Daher verwenden wir unterschiedliche Bezeichnungen. Jedoch ordnen wir einem Knoten der Vorbedingungserklärung mit der Aussage $true$ immer den Wahrheitswert $wahr$ zu (vgl. Lemma 6.2).

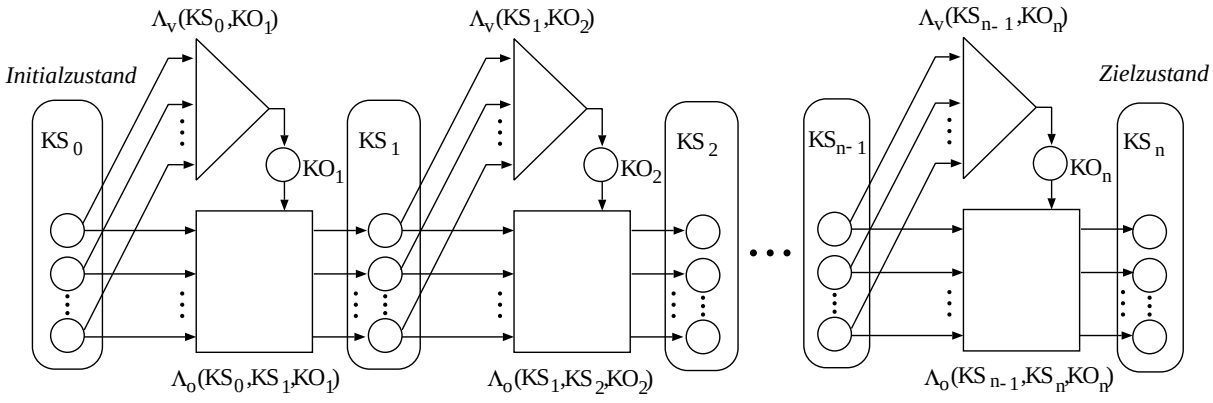

Abbildung 6.9: Schematische Darstellung einer Fallerklärung

Vorbedingungserklärungen ein:

Bezeichnung 6.3 (Vorbedingungserklärung) Eine Vorbedingungserklärung wird mit $\Lambda_v(\mathcal{KS}_{i-1}, KO_i)$ bezeichnet wobei $\mathcal{KS}_{i-1}$ die Menge der Zustandknoten KS_{i-1} ist. KO_i ist der Operatorknoten.

6.3.6 Algorithmus zur Erzeugung der Fallerklärung

Eine Fallerklärung läßt sich aus je einer Operatorerklärung $\Lambda_o(\mathcal{KS}_{i-1}, \mathcal{KS}_i, KO_i)$ und einer Vorbedingungserklärung $\Lambda_v(\mathcal{KS}_{i-1}, KO_i)$ für jeden Operator o_i in der Lösung des Falles aufbauen (siehe Abbildung 6.9). Der in Abbildung 6.10 dargestellte Algorithmus erzeugt eine solche Fallerklärung.[7] Der Reihe nach werden hierbei die Vorbedingungen der Operatoren mit Hilfe der SLD-Prozedur nachgewiesen. Aus einem Beweisbaum für die Gültigkeit der Vorbedingung wird dann eine Vorbedingungserklärung aufgebaut. Dann wird der Folgezustand aufgrund der Operatoranwendung bestimmt und eine entsprechende Operatorerklärung wird aufgebaut. Als Ergebnis ergibt sich eine vollständige Fallerklärung.

Lemma 6.2 (Gültigkeit der Fallerklärung) Seien die in Abschnitt 6.3.3 beschriebenen Wahrheitsfunktionen für die Knoten der Fallerklärung gegeben, also:

(i) $W_{KS_i}(F) = \mathit{wahr}$ gdw $F \in s_i$,
(ii) $W_{KF_i}(F) = \mathit{wahr}$ gdw $\mathcal{R} \cup s_i \vdash F$,
(iii) $W_{KO_i}(Apl(o(t_1, \ldots, t_n))) = \mathit{wahr}$ gdw $s_{i-1} \overset{o(t_1,\ldots,t_n)}{\longrightarrow} s_i$.

[7] Dieser Algorithmus ist im wesentlichen eine Erweiterung des Algorithmus zur Bestimmung der konkreten Zustandssequenz in Phase-I der Abstraktionsprozedur (siehe Abbildung 5.8).

Eingabe: Ein Fall $F = ((s_0, s_n), \langle o_1, \ldots, o_n \rangle)$.
Ausgabe: Eine Fallerklärung Λ.

$\Lambda := \emptyset$
$\mathcal{KS}_0 := \emptyset$
for each $e \in s_0$ **do**
  Erweitere Λ um einen Zustandsknoten KS_0 mit der Aussage $\overline{KS_0} = e$
  Erweitere $\mathcal{KS}_0$ um den neuen Zustandsknoten.
  end
for $i := 1$ **to** n **do**
  $\Pi := \mathrm{SLD}(s_{i-1} \cup \mathcal{R}, Pre_{o_i})$
  Sei τ ein Beweisbaum für die Gültigkeit von Pre_{o_i}, also: $(\emptyset, \tau) \in \Pi$
  Erzeuge aus τ eine Vorbedingungserklärung $\Lambda_v(\mathcal{KS}_{i-1}, KO_i)$
  Erweitere Λ um $\Lambda_v(\mathcal{KS}_{i-1}, KO_i)$
  $s_i := s_{i-1} \setminus Del_{o_i} \cup Add_{o_i}$
  Erzeuge für o_i eine Operatorerklärung $\Lambda_o(\mathcal{KS}_{i-1}, \mathcal{KS}_i, KO_i)$
  Erweitere Λ um $\Lambda_o(\mathcal{KS}_{i-1}, \mathcal{KS}_i, KO_i)$
  end

Abbildung 6.10: Algorithmus zur Erzeugung einer Fallerklärung

Es gilt: die durch Algorithmus 6.10 aufgebaute Fallerklärung ist bezüglich dieser Wahrheitsfunktionen gültig.

Beweisskizze Wir müssen zeigen, daß jede Begründung gültig ist. In der Fallerklärung kommen fünf Arten von Begründungen vor.

(i) Begründungen $A_i \leftarrow Apl(o(x_1, \ldots, x_n))$ für hinzugefügte essentielle Sätze: Sei σ eine Grundsubstitution aller Variablen der Begründung. Ist $W_{KO_i}(Apl(o(x_1, \ldots, x_n))\sigma) = wahr$, so gilt $s_{i-1} \overset{o(x_1,\ldots,x_n)\sigma}{\longrightarrow} s_i$. Dann gilt aber auch, $A_i\sigma \in s_i$ und folglich $W_{KS_i}(A_i\sigma) = wahr$.

(ii) Begründungen $E \leftarrow E, Apl(o(x_1, \ldots, x_n)), PROT_1(E, o(x_1, \ldots, x_n)) \wedge \ldots \wedge PROT_m(E, o(x_1, \ldots, x_n))$ für bedrohte essentielle Sätze: Sei σ eine Grundsubstitution aller Variablen der Begründung. Ist $W_{KO_i}(Apl(o(x_1, \ldots, x_n))\sigma) = wahr$ so gilt $s_{i-1} \overset{o(x_1,\ldots,x_n)\sigma}{\longrightarrow} s_i$. Ist $W_{KS_{i-1}}(E\sigma) = wahr$, so gilt $E\sigma \in s_{i-1}$. Ist $W_{KF_{i-1}}(PROT_j(E, o(x_1, \ldots, x_n))\sigma) = wahr$ für alle $j \in \{1, \ldots, m\}$, so kann $E\sigma$ nicht durch die Anwendung von $o(x_1, \ldots, x_n)\sigma$ gelöscht werden. Folglich gilt $E\sigma \in s_i$ und somit $W_{KS_i}(E\sigma) = wahr$.

(iii) Begründungen $E \leftarrow E, Apl(o(x_1, \ldots, x_n))$ für nicht bedrohte essentielle Sätze: Sei σ eine Grundsubstitution aller Variablen der Begründung. Ist

$W_{KO_i}(Apl(o(x_1,\ldots,x_n))\sigma) = wahr$, so gilt $s_{i-1} \overset{o(x_1,\ldots,x_n)\sigma}{\longrightarrow} s_i$. Ist $W_{KS_{i-1}}(E\sigma) = wahr$, so gilt $E\sigma \in s_{i-1}$. Da E nicht bedroht ist, gilt $E\sigma \in s_i$ und somit $W_{KS_i}(E\sigma) = wahr$.

(iv) Begründungen $Apl(o(x_1,\ldots,x_n)) \leftarrow Pre_1,\ldots,Pre_k$ für die Vorbedingung der Operatoren: Sei σ eine Grundsubstitution aller Variablen der Begründung. Gilt $W_{KF_{i-1}}(Pre_j\sigma) = wahr$ für $j \in \{1,\ldots,k\}$, so gilt $s_{i-1} \cup \mathcal{R} \vdash Pre_j\sigma$ für $j \in \{1,\ldots,k\}$. Folglich kann der Operator $o(x_1,\ldots,x_n)\sigma$ in s_{i-1} angewendet werden, d. h. $s_{i-1} \overset{o(x_1,\ldots,x_n)\sigma}{\longrightarrow} s_i$. Folglich gilt $W_{KO_i}(Apl(o(x_1,\ldots,x_n))\sigma) = wahr$.

(v) Begründungen $P \leftarrow P_1,\ldots,P_m$ aus der Vorbedingungserklärung: Sei σ eine Grundsubstitution aller Variablen der Begründung. Gilt $W_{KF_{i-1}}(P_j\sigma) = wahr$ oder $W_{KS_{i-1}}(P_j\sigma) = wahr$, so folgt $s_{i-1} \cup \mathcal{R} \vdash P_j\sigma$. Da $(P \leftarrow P_1,\ldots,P_m) \in \mathcal{R}$ gilt, folgt $s_{i-1} \cup \mathcal{R} \vdash P\sigma$ und somit $W_{KF_{i-1}}(P\sigma) = wahr$.

Bemerkung 6.11 Lemma 6.1, das die Gültigkeit von Aussagen von Erklärungen bestimmt, läßt sich nun auch auf Fallerklärungen anwenden. Bei der Fallerklärung kommen zwei Arten von Basisknoten vor:

(i) Zustandsknoten $\{K_1,\ldots,K_m\} = \mathcal{KS}_0$ des Initialzustands und
(ii) Folgerungsknoten $\{K_{m+1},\ldots,K_l\}$, die die Aussage *true* oder eine Schutzbedingung besitzen.

Es gilt nun: Wenn $\overline{K_1} \in s_0,\ldots,\overline{K_m} \in s_0$ und $\mathcal{R} \vdash \overline{K_{m+1}} \wedge \ldots \wedge \overline{K_l}$ gilt, so folgt $s_0 \overset{o_1}{\longrightarrow} s_1 \longrightarrow,\cdots,\longrightarrow s_{n-1} \overset{o_n}{\longrightarrow} s_n$ und $\overline{K^*} \in s_n$ gilt für jeden Zustandsknoten des Zielzustands $K^* \in \mathcal{KS}_n$.

Beispiel 6.3 Abbildung 6.11 zeigt die Fallerklärung für den Fall F aus Beispiel 6.1. Fettgedruckte ovale Knoten stellen Folgerungsknoten dar und fettgedruckte runde Knoten stellen Operatorknoten dar. Unter jedem inneren Knoten ist die entsprechende Begründung kursiv gedruckt abgebildet. Im Knoten selbst ist die entsprechende Aussage dargestellt. In diesem Beispiel kommt für jeden Operator eine Vorbedingungserklärung vor, sowie eine Operatorerklärung für hinzugefügte und nicht-bedrohte essentielle Sätze.

6.4 Phase-II: Generalisierung der Fallerklärung

Der Schritt, der im erklärungsbasierten Lernen nach der Generierung der Erklärung folgt, ist die Generalisierung der Erklärung. Wir sprechen hierbei von der Bildung einer *generalisierten Erklärung.*

6.4.1 Generalisierte Erklärung

Während bei einer Erklärung alle Aussagen aus Grundatomen bestehen, d. h. keine Variablen besitzen, sind bei einer *generalisierten Erklärung* beliebige Atome als Aussagen

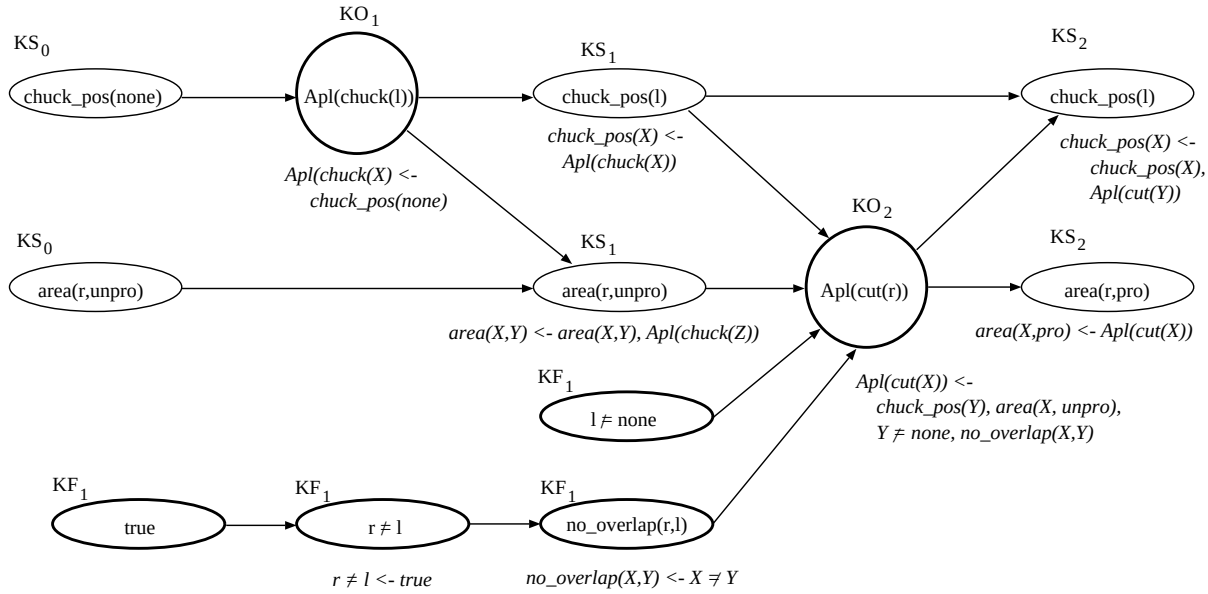

Abbildung 6.11: Beispiel für eine Fallerklärung

zugelassen, so daß Variablen vorkommen können. Die nachfolgende Definition der generalisierten Erklärung entspricht der Definition 6.8 der Erklärung, bis auf die Form der Aussagen, mit der jeder Knoten markiert ist.

Definition 6.12 (Generalisierte Erklärung) Eine *generalisierte Erklärung* Λ' ist ein zusammenhängender azyklischer gerichteter Graph. Knoten, zu denen keine Kanten hinführen, werden als *Basisknoten* bezeichnet; alle anderen Knoten werden *innere Knoten* genannt. Die Knoten, von denen eine gerichtete Kante zu einem inneren Knoten K hinführt, werden *direkte Vorgänger von* K genannt (K wird auch *direkter Nachfolger* genannt). Knoten, von denen keine Kanten wegführen, werden als *Zielknoten* bezeichnet. Jeder Knoten K des Graphen ist mit einem Atom p markiert (Schreibweise: $\overline{K} \equiv p$), das *Aussage* genannt wird. Jeder innere Knoten K, mit den direkten Vorgängern $K_1, \ldots, K_n$, besitzt zusätzlich eine Markierung (*Begründung* genannt) der Form $P \leftarrow P_1, \ldots, P_n$, wobei P und $P_1, \ldots, P_n$ Atome sind und für die es eine Substitution σ gibt mit $P\sigma = \overline{K}$ und $P_i\sigma = \overline{K_i}$ für alle $i \in \{1, \ldots, n\}$. Wir schreiben auch: $\widehat{K} \equiv P \leftarrow P_1, \ldots, P_n$.

Bemerkung 6.12 Die Definition 6.9 für die *Gültigkeit* von Begründungen und Erklärungen wird ebenfalls für generalisierte Erklärungen angewendet.

Eine generalisierte Erklärung begründet einen allgemeineren Sachverhalt als eine Erklärung. Jede beliebige Grundinstantiierung aller Variablen der Aussagen einer generalisierten Erklärung ist eine Erklärung gemäß Definition 6.8. Das für Erklärungen formulierte Lemma 6.1 läßt sich auf generalisierte Erklärungen übertragen.

Lemma 6.3 (Gültigkeit der Aussagen einer generalisierten Erklärung) Sei Λ' eine gültige generalisierte Erklärung bzgl. der Wahrheitsfunktionen W_K, seien $Kb'_1, \ldots, Kb'_n$ die Basisknoten von Λ' und sei σ eine Grundsubstitution aller Variablen, die in den Aussagen einer generalisierten Erklärung Λ' vorkommen. Es gilt: falls $\forall_{i=1,\ldots,n}\ W_{Kb'_i}(\overline{Kb'_i}\sigma) = \mathit{wahr}$ gilt, so gilt $W_{K'}(\overline{K'}\sigma) = \mathit{wahr}$ für jeden Knoten K' aus Λ'.

Beweisskizze Ist Λ' eine gültige generalisierte Erklärung, so ist die Erklärung Λ, die sich durch Anwendung von σ auf die Aussagen von Λ' ergibt, ebenfalls eine gültige Erklärung. Somit folgt mit Lemma 6.1 die gewünschte Aussage.

Beispiel 6.4 Beispiel 6.2 läßt sich wie folgt fortsetzen. Die Begründungen in der generalisierten Erklärung sind Hornklauseln aus Σ. Wir setzen nun erneut die Wahrheitsfunktion $W_K(F) = \mathit{wahr}$ gdw $\Sigma \vdash F$ voraus, so ist eine generalisierte Erklärung Λ', die nur Begründungen aus Σ enthält, gültig bzgl. W_K. Daher läßt sich Lemma 6.3 wie folgt anwenden. Sind $Kb'_1, \ldots, Kb'_n$ die Basisknoten der generalisierten Erklärung, so gilt für jeden Knoten K' der generalisierten Erklärung: $\Sigma \vdash \forall x_1, \ldots, \forall x_k(\overline{Kb'_1} \wedge \ldots \wedge \overline{Kb'_n}) \rightarrow \overline{K'}$, wobei $x_1, \ldots, x_k$ die in den Aussagen der Erklärung vorkommenden Variablen sind. Insbesondere gilt: sind $K_1^{*\prime}, \ldots, K_m^{*\prime}$ die Zielknoten der generalisierten Erklärung, so gilt: $\Sigma \vdash \forall x_1, \ldots, \forall x_k(\overline{Kb'_1} \wedge, \ldots, \wedge \overline{Kb'_n}) \rightarrow (\overline{K_1^{*\prime}} \wedge \ldots \wedge \overline{K_m^{*\prime}})$.

Bemerkung 6.13 Lemma 6.3 bietet die Grundlage für die Bildung der *operationalen Konzeptspezialisierung*, $S \rightarrow K$ die das erklärungsbasierte Generalisieren als Ergebnis liefert. Für die Hornlogik besteht diese, wie Beispiel 6.4 zeigt, nämlich genau aus der Formel $\forall x_1, \ldots, \forall x_k(\overline{Kb'_1} \wedge, \ldots, \wedge \overline{Kb'_n}) \rightarrow (\overline{K_1^{*\prime}} \wedge \ldots \wedge \overline{K_m^{*\prime}})$, wobei $x_1, \ldots, x_k$ die in den Aussagen der Erklärung vorkommenden Variablen sind.

6.4.2 Grundalgorithmus zur Generalisierung von Erklärungen

Beim erklärungsbasierten Lernen sind wir daran interessiert, eine generalisierte Erklärung aus einer (nicht-generalisierten) Erklärung zu bilden.

Definition 6.13 (Generalisierung einer Erklärung) Eine generalisierte Erklärung Λ' ist eine *Generalisierung* einer Erklärung Λ genau dann, wenn gilt:

(i) Λ' ist isomorph zu einem Teilgraphen von Λ. Der einem Knoten K von Λ entsprechende Knoten aus Λ' wird mit K' bezeichnet.
(ii) Die Zielknoten von Λ' entsprechen den Zielknoten von Λ.
(iii) Es gibt eine Substitution θ, so daß für jeden Knoten K' von Λ' gilt: $\overline{K'}\theta = \overline{K}$, d. h. die Aussagen von Λ ergeben sich durch Instantiierung der Aussagen von Λ' unter der Substitution θ.
(iv) Für jeden inneren Knoten K' aus Λ' gilt, daß die Begründung $\widehat{K'}$ der Begründung $\widehat{K}$ bis auf Variablenumbenennungen entspricht.

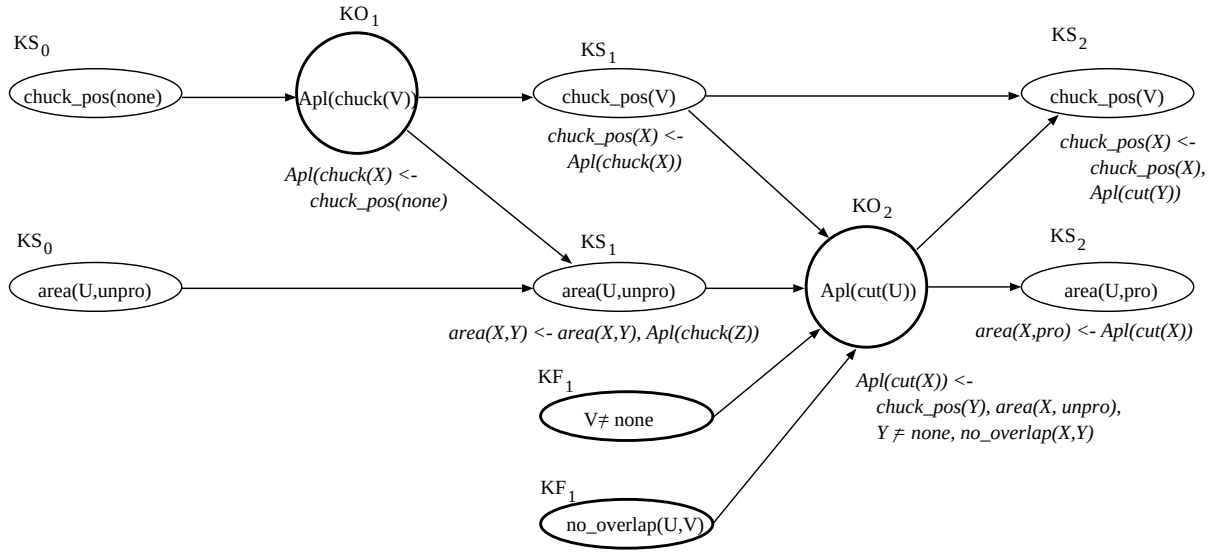

Abbildung 6.12: Beispiel für eine generalisierte Fallerklärung

Beispiel 6.5 Abbildung 6.12 stellt eine Generalisierung der Fallerklärung aus Beispiel 6.3 dar. Der hier dargestellte Graph ist offensichtlich isomorph zu einem Teilgraph der in Abbildung 6.11 gezeigten Erklärung. In der generalisierten Erklärung kommen die Knoten mit den Aussagen `true` und `r`$\neq$`l` nicht mehr vor. Die Substitution $\theta = \{V/l, U/r\}$ instantiiert jede Aussage an einem Knoten der generalisierten Erklärung so, daß die Aussage des entsprechenden Knotens der Erklärung entsteht.

Die Generalisierung einer Erklärung stellt eine Verallgemeinerung eines relevanten Ausschnitts der ursprünglichen Erklärung dar (siehe Abbildung 6.13). Hierbei bleibt die Struktur der Erklärung, die sich Aufgrund des Zusammenwirkens der Begründungen ergibt, erhalten. Insbesondere enthält die generalisierte Erklärung dieselben Begründungen wie die ursprüngliche Erklärung, wobei diese jedoch auf allgemeinere Weise zur Bildung der Aussagen instantiiert sind.

Lemma 6.4 (Generalisierungseigenschaft für Erklärungen) Ist Λ' eine Generalisierung der Erklärung Λ, sind $K_1, \ldots, K_n$ die Basisknoten von Λ, sind $Kb'_1, \ldots, Kb'_{n'}$ die Basisknoten von Λ' und sind $K^*_1, \ldots, K^*_m$ und $K^{*\prime}_1, \ldots, K^{*\prime}_m$ die Zielknoten von Λ bzw. Λ', so gilt:[8] es gibt eine Substitution θ mit

(i) $\forall_{i=1\ldots m}\, \overline{K^*_i} = \overline{K^{*\prime}_i}\theta$ und
(ii) wenn $\forall_{i=1\ldots n}\, W_{K_i}(\overline{K_i}) = true$ dann gilt $\forall_{i=1\ldots n'}\, W_{Kb_i{}'}(\overline{Kb_i'}\theta) = true$.

[8] Man beachte, daß die Zielknoten von Λ'den Zielknoten von Λ entsprechen (Subgraphisomorphie); die Basisknoten entsprechen sich jedoch nicht notwendigerweise.

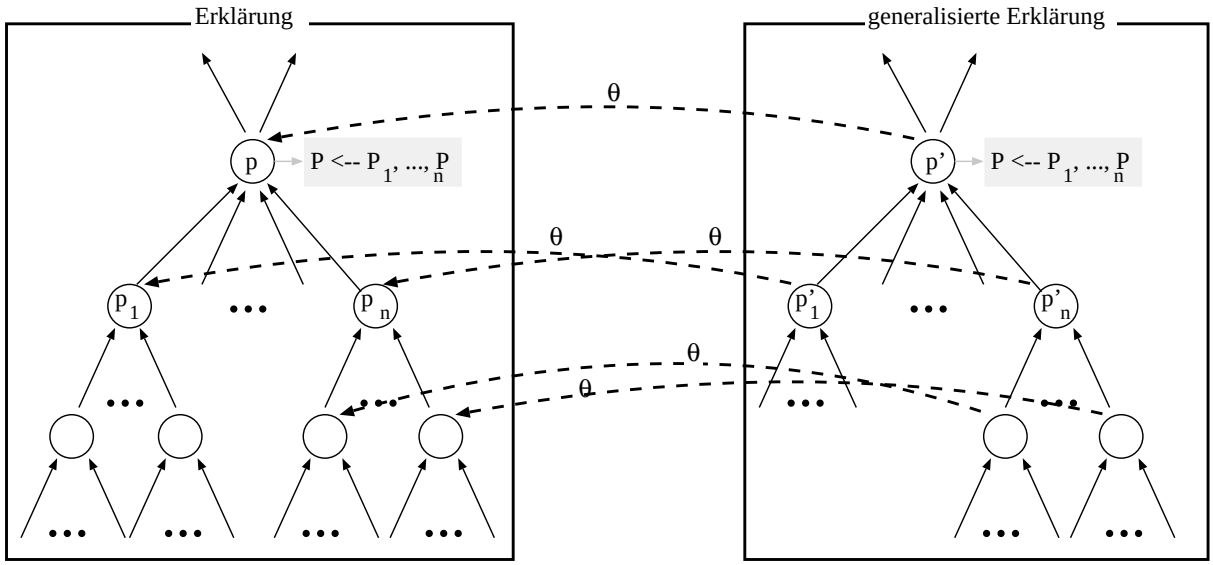

Abbildung 6.13: Beziehung zwischen einer Erklärung und einer generalisierten Erklärung

Beweisskizze Die Gültigkeit dieses Lemmas ergibt sich aus Definition 6.13, wobei die in Lemma 6.4 vorkommende Substitution θ genau der Substitution θ der Definition 6.13 entspricht. Lemma 6.1 garantiert dabei, daß die Bedingung (ii) erfüllt ist.

Bemerkung 6.14 Wie in Beispiel 6.2 und Bemerkung 6.8 erörtert wurde, ergibt sich für die Hornlogik das Trainingsbeispiel zu $B = \{\overline{K_1}, \ldots, \overline{K_n}\}$ und gemäß Beispiel 6.4 und Bemerkung 6.13 ist die operationale Konzeptspezialisierung $S \rightarrow K$ die Formel $\forall x_1, \ldots, \forall x_k(\overline{Kb'_1} \wedge, \ldots, \wedge \overline{Kb'_n}) \rightarrow \overline{K_1^{*\prime}} \wedge \ldots \wedge \overline{K_m^{*\prime}})$. Lemma 6.4 besagt nun gerade, daß das Trainingsbeispiel die so gewonnene operationale Konzeptspezialisierung umfaßt, d. h. daß $\Sigma \cup B \vdash S$ gilt.

Die Bildung einer generalisierten Erklärung wird durch ein *Operationalitätskriterium* gesteuert, daß den relevanten Ausschnitt aus einer Erklärung (Teilgraph) festlegt.

Definition 6.14 (Operationalitätskriterium) Ein Operationalitätskriterium Ω ist ein Prädikat über der Menge der Knoten des Erklärungsgraphen. Wir schreiben $\Omega(K)$ falls der Knoten K das Operationalitätskriterium erfüllt.

Erfüllt ein Knoten das Operationlitätskriterium, so ist in der generalisierten Erklärung keine weitere Rechtfertigung für die Aussagen dieses Knotens erforderlich. Operationale Knoten werden zu Basisknoten der generalisierten Erklärung.

Beispiel 6.6 In der generalisierten Erklärung aus Beispiel 6.5 ist ein Knoten genau dann operational, wenn er entweder ein Zustandsknoten KS_0 ist oder ein Folgerungsknoten, bei dem eines der Prädikate $\neq$ oder `no_overlap` vorkommt.

Der in Abbildung 6.14 dargestellte Algorithmus generalisiert eine Erklärung Λ unter Berücksichtigung des Operationalitätskriteriums Ω.[9] Die in diesem Algorithmus verwendete Funktion $rename(C)$ führt eine eindeutige Umbenennung aller in der Formel C vorkommenden Variablen durch, so daß bei mehrfachem Aufruf von *rename* jede Variable genau einmal auftritt. Der Algorithmus durchläuft die Erklärung Λ, ausgehend vom Zielknoten, rückwärts bis zu den Knoten, die das Operationalitätskriterium erfüllen. Bei diesem Durchlauf wird die generalisierte Erklärung Λ' inkrementell aufgebaut, wobei die Variablen aller vorhandenen Begründungen so umbenannt werden, daß sie für die gesamte generalisierte Erklärung eindeutig sind. Die operationalen Knoten der Erklärung Λ werden dabei zu Basisknoten der generalisierten Erklärung Λ'. Die Knoten der Erklärung Λ werden außerdem mit den korrespondierenden Knoten der generalisierten Erklärung Λ' markiert.

Die wesentliche Aufgabe des Algorithmus besteht jedoch in der Bildung der Substitution σ. Diese Substitution ist die allgemeinste Substitution, die alle Begründungen so instantiiert, daß jede Vorbedingung einer Begründung durch die Konklussion der Begründung an den jeweiligen Vorgängerknoten erfüllt wird. Zur Vereinfachung der Konstruktion dieser Substitution werden auch die Basisknoten der generalisierten Erklärung mit einer Begründung $\widehat{K}$ versehen. Bei operationalen Basisknoten besteht diese Begründung aus einer Generalisierung der entsprechenden Aussage des Basisknotens, und bei nicht-operationalen Basisknoten besteht sie aus der Aussage selbst. Da die Variablen in den einzelnen Begründungen disjunkt sind, können die entsprechenden Instantiierungen in einer einzigen Substitution σ aufgebaut werden. Ist diese Substitution gebildet, so werden abschließend die Aussagen der generalisierten Erklärung durch Anwendung der Substitution auf die jeweiligen Konklussionsteile der Begründungen gebildet.

Satz 6.1 (Korrektheit des Generalisierungsalgorithmus) Der Algorithmus aus Abbildung 6.14 bildet eine Generalisierung Λ' einer Erklärung Λ. In Λ' sind alle Knoten, die das Operationalitätskriterium erfüllen, Basisknoten.

Beweisskizze Zunächst gilt, daß Λ' tatsächlich eine generalisierte Erklärung ist, da für alle inneren Knoten K' mit $\widehat{K'} \equiv P \leftarrow P_1, \ldots, P_n$ gilt, daß $\overline{K} = P\sigma$ und $\overline{K_i} = P_i\sigma$ für genau die Substitution σ, die im Algorithmus bestimmt wird. Offensichtlich ist, daß Λ' isomorph zu einem Teilgraphen von Λ ist; jeder Knoten K in Λ ist mit dem entsprechenden Knoten K' markiert. Außerdem folgt mit Hilfe des Liftinglemmas, daß es eine Substitution θ gibt mit $\overline{K} = \overline{K'}\theta$.

6.4.3 Anwendung des Grundalgorithmus zur Generalisierung der Fallerklärung

Dieser Algorithmus zur Erklärungsgeneralisierung ist universell und kann zur Generalisierung beliebiger Erklärungen verwendet werden. Zur Anwendung auf Fallerklärungen muß

[9] Aus Gründen der Übersichtlichkeit gehen wir in diesem Algorithmus davon aus, daß jeder Zielknoten der Erklärung ein innerer Knoten ist.

Eingabe: Erklärung Λ, Operationalitätskriterium Ω
Ausgabe: Generalisierte Erklärung Λ'

Sei L die Menge der Zielknoten der Erklärung Λ.
$\sigma := \emptyset$
for each $K \in L$ **do**
  Erzeuge in Λ' einen Knoten K' und markiere K mit K'
  $\widehat{K'} := rename(\widehat{K})$
  end
while $L \neq \emptyset$ **do**
  Seien $K \in L$ und $\widehat{K} \equiv P \leftarrow P_1, \ldots, P_n$ sowie $K_1, \ldots, K_n$ Vorgänger von K in Λ.
  $L := L - \{K\}$
  for $i = 1$ **to** n **do**
    if K_i ist nicht markiert **then**
      Erzeuge in Λ' einen Knoten K'_i und markiere K_i mit K'_i
      if $\Omega(K_i)$ **then** (* K_i ist operational *)
        Sei $\overline{K_i} = q(t_1, \ldots, t_m)$, wobei q ein Prädikatssymbol ist und t_i Terme sind.
        $\widehat{K'_i} := rename(q(x_1, \ldots, x_m))$
        else (* K_i ist nicht operational *)
        if K_i ist kein Basisknoten **then**
          $\widehat{K'_i} := rename(\widehat{K_i})$
          $L := L \cup \{K_i\}$
        else $\widehat{K'_i} := \overline{K_i}$
    Sei nun K_i markiert mit K'_i
    Erzeuge in Λ' eine gerichtete Kante von K'_i nach K'.
    Sei $\widehat{K'_i} \equiv Q \leftarrow Q_1, \ldots, Q_m$ oder $\widehat{K'_i} \equiv Q$
    $\tau := mgu(P_i\sigma, Q\sigma)$
    $\sigma := \sigma\tau$
    end
  end
for each K' aus Λ' **do**
  Sei $\widehat{K'} \equiv P \leftarrow P_1, \ldots, P_n$ oder $\widehat{K'} \equiv P$
  $\overline{K'} := P\sigma$
  end

Abbildung 6.14: Grundalgorithmus zur Erklärungsgeneralisierung

lediglich noch ein geeignetes Operationalitätskriterium bestimmt werden.

Alle Aussagen an den operationalen Knoten sind genau die Aussagen, die in den Basiskno-

ten vorkommen und folglich auch in der operationalen Konzeptspezialisierung enthalten sind. Wir unterscheiden zwei Arten von operationalen Knoten:

Essentielle Sätze aus dem Initialzustand: Essentielle Sätze aus dem Initialzustand sollen im generalisierten Fall als generalisierter Initialzustand vorkommen. Daher bilden wir das folgende Operationalitätskriterium:

$$\Omega_S(K) \iff K \in \mathcal{KS}_0$$

Operationale Prädikate aus $\mathcal{M}$: Folgerungsknoten, die als Aussagen zulässige Prädikate aus $\mathcal{M}$ enthalten, sind operational, da diese Aussagen im generalisierten Fall als Constraints erlaubt sind. Jedoch müssen wir an diese Knoten die Bedingung stellen, daß sie nur durch die universelle Implikationen $\mathcal{R}$ der Problemlösedomäne ableitbar sein müssen. Wir gehen hierbei davon aus, daß das $\neq$-Prädikat, das in den Schutzbedingungen vorkommt, ebenfalls ein zulässiges Prädikat ist, d. h. $\neq \in \mathcal{M}$ gilt. Wir bilden daher das folgende Operationalitätskriterium:

$$\begin{aligned}\Omega_{\mathcal{M}}(K) \quad \iff \quad & \overline{K} \equiv p(t_1, \ldots, t_n) \wedge p \in \mathcal{M} \text{ und} \\ & \nexists K' \text{mit} \, (K' \in \mathcal{KS}_i \wedge K' \text{ist Vorgänger von } K \text{ in } \Lambda)\end{aligned}$$

Für die Generalisierung der Erklärung wird nun ein Operationalitätskriterium Ω benutzt, das wie folgt definiert ist:

$$\Omega(K) \iff \Omega_S(K) \vee \Omega_{\mathcal{M}}(K)$$

6.5 Phase-III: Bildung des generalisierten Falles

Im letzten Schritt der erklärungsbasierten Fallgeneralisierung wird aus einer generalisierten Fallerklärung ein generalisierter Fall gemäß Definition 6.5 gebildet. Hierzu müssen die einzelnen Bestandteile des generalisierten Falles aus der generalisierten Erklärung Λ' extrahiert werden. Der in Abbildung 6.15 dargestellte Algorithmus führt die hierzu erforderlichen Zuweisungen durch. Der generalisierte Initialzustand besteht aus den Aussagen der Zustandsknoten, die den Initialzustand in der Erklärung repräsentieren. Diese Knoten erfüllen das Operationalitätskriterium $\Omega_S(K)$. Der generalisierte Zielzustand besteht aus den Aussagen an den Zielknoten der generalisierten Erklärung. Die Lösungsklasse, also die Folge generalisierter Operatoren, ergibt sich aus der Folge der Aussagen an den Operatorknoten der Erklärung. Die verbleibenden Basisknoten der Erklärung, also die Knoten, die das Operationalitätskriterium $\Omega_{\mathcal{M}}(K)$ erfüllen, sind die Constraints der Problemklasse.

Eingabe: Generalisierte Fallerklärung Λ' für den Fall $F = ((s_0, s_n), \langle o_1, \ldots, o_n \rangle)$
Ausgabe: Generalisierter Fall GF

$S_I(X_I) := \{\overline{K'} | K' \in \mathcal{KS}'_0 \text{ aus } \Lambda'\}$
$S_G(X_G) := \{\overline{K'} | K' \in \mathcal{KS}'_n \text{ aus } \Lambda'\}$
$\bar{O}(X_O) := \langle o_1, \ldots, o_n \rangle$, mit KO'_i ist Operatorknoten in Λ' und $\overline{KO'_i} = Apl(o_i)$
$C(X_C) := \{\overline{K'} | K' \in \Lambda' \wedge \Omega_{\mathcal{M}}(K')\}$

$GF := ((S_I(X_I), S_G(X_G), C(X_C)), \bar{O}(X_O))$

Abbildung 6.15: Algorithmus zur Bildung des generalisierten Falles

Beispiel 6.7 Der Algorithmus zur Bildung des generalisierten Falls erzeugt aus der Fallerklärung, die in Beispiel 6.5 beschrieben ist, genau den in Beispiel 6.1 gezeigten generalisierten Fall GF. Hierbei ist die Menge der operationalen Prädikate wie folgt definiert: $\mathcal{M} = \{\neq, \texttt{no_overlap}\}$.

6.6 Eigenschaften der erklärungsbasierten Fallgeneralisierung

6.6.1 Korrektheit des Verfahrens

Das Verfahren zur erklärungsbasierten Fallgeneralisierung, bestehend aus der Erzeugung der Fallerklärung (siehe Abschnitt 6.3), der Generalisierung der Fallerklärung (siehe Abschnitt 6.4) und der Bildung des generalisierten Falles (siehe Abschnitt 6.5), führt eine korrekte Fallgeneralisierung durch.

Satz 6.2 (Korrektheit der Fallgeneralisierung) Der durch das Verfahren der Fallgeneralisierung gebildete generalisierte Fall GF ist eine korrekte Generalisierung des Falls F gemäß Definition 6.7.

Beweisskizze Es gilt, daß Λ' eine Generalisierung von Λ ist (siehe Satz 6.1). Daher existiert gemäß Definition 6.13 eine Substitution θ, so daß $\overline{K'}\theta = \overline{K}$ für alle Knoten K' aus Λ' und zugeordnete Knoten K aus Λ. Folglich gelten: $s_I = S_I(X_I)\theta$, $s_G = S_G(X_G)\theta$ und $\bar{o} = \bar{O}(X_O)\theta$ und die Constraints aus $C(X_C)\theta$ sind als Aussagen in Λ enthalten und aufgrund der Definition von $\Omega_{\mathcal{M}}$ aus $\mathcal{R}$ ableitbar. Folglich ist der generalisierte Fall GF eine Generalisierung von F. Die Korrektheit von GF folgt aus Lemma 6.3 wie folgt: Sei σ eine

Substitution aller in GF vorkommenden Variablen. Falls $s_0 = S_I(X_I)\sigma$, so gilt $W_{K'}(\overline{K'}) = \mathit{true}$ für jeden Knoten $K' \in \mathcal{KS}_0$. Gilt weiterhin $\mathcal{R} \vdash C(X_C)\sigma$, so gilt $W_{K'}(\overline{K'}) = \mathit{true}$ für jeden Basisknoten K' von Λ'. Folglich gilt $W_{K'}(\overline{K'}) = \mathit{true}$ für *jeden Knoten* $K' \in \Lambda'$. Somit gilt $s_{i-1} \xrightarrow{o_i} s_i$ für alle $i = 1, \ldots n$ und $s_n = S_G(X_G)\sigma$.

6.6.2 Komplexität des Verfahrens

Die Komplexität des Verfahrens wird im wesentlichen von der Komplexität der Generierung und Generalisierung der Fallerklärung geprägt. Wir möchten hierbei davon ausgehen, daß sich die Komplexität der SLD-Beiweisprozedur zum Beweis der Operatorvorbedingungen für eine feste Problemlösedomäne nach oben durch eine Konstante beschränken läßt, so daß die hierdurch entstehenden Beweisbäume ebenfalls eine feste Größe nicht überschreiten können. Dann ergibt sich, daß die Größe der Fallerklärung linear von der Länge des Plans abhängt. Ebenso hängt der Algorithmus zur Erklärungsgeneralisierung linear von der Größe der Fallerklärung ab.

Satz 6.3 (Komplexität des Verfahrens zur Fallgeneralisierung) Das Verfahren zur Fallgeneralisierung besitzt bei einer konstanten Domänentheorie eine Komplexität von $O(n)$, wobei n die Länge des Plans ist.

6.7 Lösungsspezialisierung

Ein neues Problem kann genau dann unter Verwendung eines generalisierten Falles gelöst werden, wenn der generalisierte Fall zur Lösung wiederverwendbar ist. Den Begriff der Wiederverwendbarkeit haben wir bereits in der Begriffsbestimmung 4.9 informell eingeführt. Formal ergibt sich die Wiederverwendbarkeitsbedingung eines generalisierten Falles wie folgt:

Definition 6.15 (Wiederverwendbarkeit eines generalisierten Falls) Ein generalisierter Fall $GF = (P(X_P), \bar{O}(X_O))$ ist genau dann zur Lösung des Problems p wiederverwendbar, wenn p ein Problem der Problemklasse $P(X_P)$ ist.

Für wiederverwendbare korrekte generalisierte Fälle (siehe Definition 6.6) ist dann garantiert, daß die sich ergebende Lösung $\bar{o} = O(X_O)\sigma$ tatsächlich eine korrekte Lösung des Problems p ist. Somit kann durch einen Wiederverwendbarkeitstest auch direkt eine Lösungsspezialisierung erfolgen.

Die Aufgabe eines Algorithmus zum Test der Wiederverwendbarkeit eines generalisierten Falles besteht im wesentlichen darin, eine geeignete Substitution σ zu finden, die den Anforderungen aus Definition 6.15 genügt. Hierbei besteht zum einen die Aufgabe, das Matching von Initial- und Zielzustand durchzuführen und zum anderen die Gültigkeit der Constraints

zu garantieren. Während Letzteres im wesentlichen durch ein SLD-Resolutionsverfahren erreicht werden kann, bedarf das Matching von Zuständen noch einer gewissen Aufmerksamkeit.

Beispiel 6.8 Betrachten wir den generalisierten Fall GF aus Beispiel 6.1, so können wir feststellen, daß dieser zur Lösung des neuen Problems p = ({`chuck_pos(none)`, `area(l,unpro)`}, {`chuck_pos(r)`, `area(l,pro)` }) wiederverwendbar ist. Hierbei ist $\sigma = \{U/l, V/r\}$ die entsprechende Substitution und $\langle$`chuck(r)`, `cut(l)`$\rangle$ der sich durch die Lösungsspezialisierung ergebende Plan für das neue Problem p. Ein komplexeres Beispiel zur Lösungsspezialisierung werden wir in Abschnitt 8.3.5 angeben.

6.7.1 Matching von Zuständen

Die Aufgabe des Zustandsmatching besteht bei einem gegebenen Zustand s und einem gegebenen generalisierten Zustand $S(X)$ darin, eine Belegung σ der Variablen X im generalisierten Zustand $S(X)$ zu finden, so daß $S(X)\sigma = s$ gilt (vgl. [Kapur & Narendran, 1986]).

Ein solcher Zustandsmatch ist in der Regel nicht eindeutig, d. h. es kann mehrere Substitutionen σ geben, die diese Bedingung erfüllen. In Abbildung 6.16 ist ein einfacher Algorithmus angegeben, der einen solchen Zustandsmatch durchführt und die Menge aller Subsitutionen σ bestimmt. Dieser Algorithmus bestimmt iterativ die Substitutionen, die jeweils ein *weiteres* Atom aus $S(X)$ mit einem Grundatom aus s unifiziert. Hierdurch wird zunächst die Menge der Substitutionen σ gebildet, für die $S(X)\sigma \subset s$ gilt. Danach wird aus dieser Menge die Menge der Substitutionen mit $S(X)\sigma = s$ herausgefiltert und als Endergebnis zurückgegeben.

Wie in [Kapur & Narendran, 1986] gezeigt wurde, ist das Matching von Termmengen NP-vollständig. Wie man leicht erkennt, wächst im Algorithmus aus Abbildung 6.16 die Menge der möglichen Substitutionen im schlimmsten Fall exponentiell mit der Menge der Atome in S(X). Hierbei spielt jedoch die Anzahl der Atome in s, die denselben Prädikatsnamen besitzen, eine entscheidende Rolle, da hierdurch die Anzahl der möglichen Matches wesentlich beschränkt wird. Im folgenden sei $\mu(s)$ die maximale Anzahl der Atome im Zustand s mit demselben Prädikatsnamen.

Lemma 6.5 (Komplexität des Zustandsmatching) Ist $\mu(s)$ die maximale Anzahl der Atome aus s, die denselben Prädikatsnamen besitzen, so liefert das Zustandsmatching maximal $\mu(s)^{|S(X)|}$ Substitutionen. Die Worstcasekomplexität des Zustandsmatching ist durch $O(|s| \cdot \mu(s)^{|S(X)|-1})$ begrenzt.

Diese Worstcasekomplexität wird jedoch in praktischen Situtationen selten erreicht. Durch geeignete Indexierung, z. B. über die Prädikatsnamen, läßt sich der Aufwand für das Zustandsmatching auf $O(\mu(s)^{|S(X)|})$ reduzieren.

Eingabe: Ein Zustand s und ein generalisierter Zustand $S(X)$
Ausgabe: Eine Menge von Substitutionen $\Omega = \{\sigma | S(X)\sigma = s\}$.

procedure *statematch*$(S(X), s)$
$\Omega' := \{\emptyset\}$
for each $E \in S(X)$ **do**
  $\Omega := \emptyset$
  for each $\sigma \in \Omega'$ **do**
    for each $e \in s$ **do**
      if es existiert eine Subsitution τ mit $e = E\sigma\tau$ **then**
        $\Omega := \Omega \cup \{\sigma\tau\}$
      end
    end
  $\Omega' := \Omega$
  end
(Es gilt nun:* $\Omega' = \{\sigma | S(X)\sigma \subseteq s\}$ **)*
$\Omega := \emptyset$
for each $\sigma \in \Omega'$ **do**
  if $S(X)\sigma = s$ **then** $\Omega := \Omega \cup \{\sigma\}$
  end
return Ω

Abbildung 6.16: Algorithmus zum Matchen eines generalisierten Zustandes

6.7.2 Algorithmus zum Test der Wiederverwendbarkeit und zur Durchführung der Lösungsspezialisierung

Der in Abbildung 6.17 dargestellte Algorithmus testet die Wiederverwendbarkeit eines generalisierten Falles. Hierbei wird auch eine Substitution σ bestimmt, die alle Variablen im generalisierten Fall instantiiert. Als Resultat liefert der Algorithmus entweder eine korrekte Lösung aus der Lösungsklasse des generalisierten Falles oder das Ergebnis, daß der generalisierte Fall nicht wiederverwendbar ist.

Zunächst bestimmt der Algorithmus alle möglichen Substitutionen (Ω_1), die einen Match des generalisierten Initialzustandes des generalisierten Falles mit dem Initialzustandes des Problems ermöglichen. Dann wird versucht, eine dieser Substitution σ_1 zu einem Match des generalisierten Zielzustandes mit dem Zielzustand des Problems zu erweitern. Im Algorithmus ist Ω_2 die Menge der Substitutionen σ_2, die eine Substitution σ_1 zu einem Match von Initial- und Zielzustand erweitert. Aus dieser Menge wird nun wiederum eine Substitution $\sigma_1\sigma_2$ bestimmt, die die Variablen X_C in den Constraints C des generalisierten Falles so instantiiert, daß die Constraints erfüllt sind. Bei diesem Erfülltheitstest bestimmt

die SLD-Resolution eine Menge von Belegungen Ω_3 der verbleibenden freien Variablen in X_C. Mit $\sigma_3 \in \Omega_3$ wählen wir eine solche Substitution aus und erhalten mit $\sigma_1\sigma_2\sigma_3$ eine Substitution für alle Variablen in X_I,X_G und X_C, so daß die Wiederverwendbarkeit des generalisierten Falles garantiert ist. Falls es Variablen in der Lösungsklasse des generalisierten Falls gibt (also Variablen aus X_O), die durch diese Substitution nicht instantiiert werden, so können für diese Variablen beliebige Grundinstanzen gewählt werden. In jedem Fall ist die resultierende Lösung eine korrekte Lösung des Problems. Wenn jedoch keine solche Substitutionen gebildet werden können, so ist der generalisierte Fall nicht zur Lösung des Problems wiederverwendbar und der Algorithmus terminiert mit *failure*.

Eingabe: Generalisierter Fall $GF = ((S_I(X_I), S_G(X_G), C(X_C)), O(X_O))$ und ein Problem $p = (s_I, s_G)$
Ausgabe: Eine Lösung $\bar{o}$ oder *failure*

$\Omega_1 := statematch(S_I(X_I), s_I)$
while $\Omega_1 \neq \emptyset$ **do**
  Sei $\sigma_1 \in \Omega_1$
  (Es gilt: $s_I = S_I(X_I)\sigma_1$ *)*
  $\Omega_1 := \Omega_1 \setminus \{\sigma_1\}$
  $\Omega_2 := statematch(S_G(X_G)\sigma_1, s_G)$
  while $\Omega_2 \neq \emptyset$ **do**
    Sei $\sigma_2 \in \Omega_2$
    (Es gilt: $s_G = S_G(X_G)\sigma_1\sigma_2$ *)*
    $\Omega_2 := \Omega_2 \setminus \{\sigma_2\}$
    $\Omega_3 := \mathrm{SLD}(\mathcal{R}, C(X_C)\sigma_1\sigma_2)$
    if $\Omega_3 \neq \emptyset$ **then**
      Sei $\sigma_3 \in \Omega_3$
      (Es gilt: $\mathcal{R} \vdash C(X_C)\sigma_1\sigma_2\sigma_3$ *)*
      $\sigma := \sigma_1\sigma_2\sigma_3$
      Erweitere σ beliebig zu einer Grundsubstitution aller Variablen
        aus $X_I \cup X_G \cup X_C \cup X_O$
      $\bar{o} := O(X_O)\sigma$
      return $success(\bar{o})$ *(* Lösung $\bar{o}$ *)*
    end
  end
return *failure (* Generalisierter Fall nicht wiederverwendbar *)*

Abbildung 6.17: Algorithmus zum Test der Wiederverwendbarkeit eines generalisierten Falles und zur Bestimmung einer Lösungsspezialisierung

6.7.3 Aufwand zur Lösungsspezialisierung

Der Aufwand zur Lösungsspezialisierung setzt sich zusammen aus den Aufwänden zum Matching der Initial- und Zielzustände (siehe Lemma 6.5) sowie dem Aufwand zur Überprüfung der Constraints. In den Constraints können jedoch freie (existenziell quantifizierte) Variablen vorkommen, so daß die SLD-Resolution in der Regel ein Backtracking über verschiedene Instantiierungen dieser Variablen durchführen muß. Folglich ergibt sich ein exponentieller Bestandteil im Gesamtaufwand, der von der Anzahl der Konjunktionsglieder in den Constraints abhängt. Das folgende Lemma faßt den Gesamtaufwand zur Lösungsspezialisierung zusammen.

Lemma 6.6 (Komplexität der Lösungsspezialisierung) Sei $C_{\mathcal{M}}$ der Aufwand, der zur Überprüfung der Gültigkeit eines operationalen Prädikates (also eines Prädikates aus der Menge $\mathcal{M}$) erforderlich ist. Sei weiterhin $\gamma_{\mathcal{M}}$ die maximale Anzahl möglicher Substitutionen σ für Variablen aus einem operationalen Prädikat P, so daß $\mathcal{R} \vdash P\sigma$ gilt. Die Worstcasekomplexität des Algorithmus zum Test Wiederverwendbarkeit eines generalisierten Falles und zur Bestimmung einer Lösungsspezialisierung ergibt sich dann zu

$$O(\mu(s_i)^{|S_I(X_I)|} \cdot \mu(s_g)^{|S_G(X_G)|} \cdot C_{\mathcal{M}} \cdot \gamma_{\mathcal{M}}{}^{|C(X_C)|-1}).$$

Aus dieser Komplexitätsbetrachtung erkennt man, daß der Aufwand für die Wiederverwendung im wesentlichen von der „Größe“ des generalisierten Falls abhängt. Hierbei spielen

- die Anzahl der Atome aus Initial- und Zielzustand mit gleichem Prädikatsnamen,
- die Anzahl der essentiellen Sätze im generalisierten Initial- und Zielzustand,
- die Anzahl der Konjunktionsglieder in den Constraints

eine wesentliche Rolle.

Kapitel 7

Organisation der Fallbasis

In diesem Kapitel wird der im PARIS-System verfolgte Ansatz zur Organisation der Fallbasis im Detail erörtert. Wir beschreiben somit die in Abbildung 7.1 grau hinterlegten Komponenten der PARIS Architektur. Die eingeführten Verfahren werden wir durch einfache Beispiele verdeutlichen. Für ein komplexeres Beispiel verweisen wir erneut auf Kapitel 8.

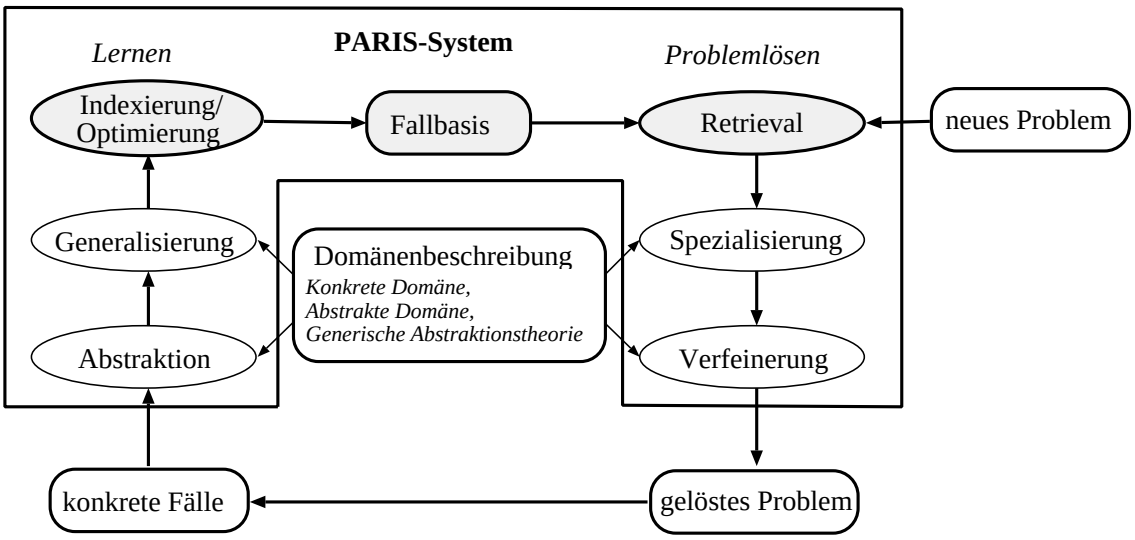

Abbildung 7.1: Die Architektur des PARIS Systems

7.1 Strukturierung der Fallbasis

7.1.1 Ausgangssituation

Wir wollen im folgenden davon ausgehen, daß aus einer Menge von vorliegenden Fällen $\mathcal{F} = \{F_1, \ldots, F_n\}$ der konkreten Problemlösedomäne durch die Fallabstraktion und anschließende Fallgeneralisierung eine Menge von generalisierten abstrakten Fällen $\Gamma = \{GAF_1, \ldots, GAF_m\}$ gebildet wurde. Hierbei entstehen in der Regel aus einem Fall F_j eine Menge von generalisierten abstrakten Fällen, die wir mit $\Gamma_j \subseteq \Gamma$ bezeichnen (siehe Abbildung 7.2). Die Gesamtmenge aller generalisierten abstrakten Fälle ergibt sich genau aus der Gesamtheit aller generalisierten abstrakten Fälle, die sich aus $\mathcal{F}$ bilden lassen, d. h. $\Gamma = \bigcup_{j=1}^{n} \Gamma_j$. Das Ziel einer guten Organisation der Fallbasis besteht darin, den Zugriff auf die generalisierten abstrakten Fälle aus Γ zu optimieren.

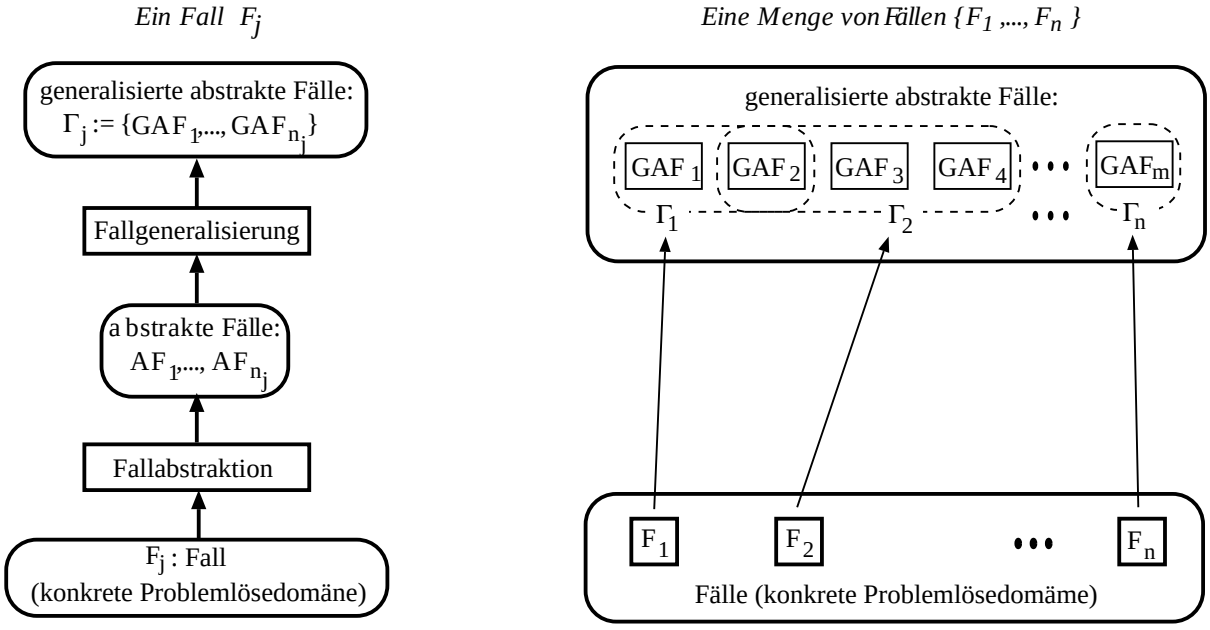

Abbildung 7.2: Gewinnung der Trainingsbeispiele

7.1.2 Wiederverwendbarkeitsrelationen für generalisierte abstrakte Fälle

Der Ansatz, den wir zur Organisation der Fallbasis Γ verwenden wollen, besteht darin, die generalisierten abstrakten Fälle mit Hilfe der Wiederverwendbarkeitsrelation (siehe Begriffsbestimmung 4.12) anzuordnen. Nachdem wir in den vorherigen Kapiteln die (schwa-

che) Wiederverwendbarkeit abstrakter Fälle (siehe Definitionen 5.8 bzw. 5.9) sowie die Wiederverwendbarkeit generalisierter Fälle (siehe Definition 6.15) eingeführt haben, können wir zunächst die (schwache) Wiederverwendbarkeit eines generalisierten abstrakten Falls GAF für ein Problem p definieren. Diese Definition ergibt sich auf natürliche Weise wie folgt:

Definition 7.1 (Wiederverwendbarkeit und schwache Wiederverwendbarkeit) Ein generalisierter abstrakter Fall GAF ist zur Lösung eines Problems p (schwach) wiederverwendbar, wenn es einen abstrakten Fall $F_a = (p_a, \bar{o_a})$ gibt, so daß gilt:

(i) F_a ist für p (schwach) wiederverwendbar gemäß Definition 5.8 (5.9) und

(ii) GAF ist für p_a wiederverwendbar gemäß Definition 6.15.

Für die Wiederverwendbarkeit von GAF schreiben wir $GAF \triangleright p$; für die schwache Wiederverwendbarkeit schreiben wir $GAF \mathrel{\triangleright\!\!\sim} p$.

Bemerkung 7.1 Die schwache Wiederverwendbarkeit ist eine notwendige Voraussetzung für die Wiederverwendbarkeit, d. h. $GAF \triangleright p$ impliziert $GAF \mathrel{\triangleright\!\!\sim} p$. Die schwache Wiederverwendbarkeit generalisierter abstrakter Fälle ist entscheidbar, die Wiederverwendbarkeit jedoch nicht.

Auf der Grundlage der Wiederverwendbarkeit generalisierter abstrakter Fälle läßt sich nun eine Ordnungsrelation auf der Menge der generalisierten abstrakten Fälle definieren, die wir zur Optimierung des Zugriffs auf die generalisierten abstrakten Fälle verwenden können.

Definition 7.2 (Wiederverwendbarkeitsrelation zwischen GAF's) Ein generalisierter abstrakter Fall GAF_1 ist *flexibler wiederverwendbar* als ein generalisierter abstrakter Fall GAF_2 (wir schreiben: $GAF_1 \sqsupseteq GAF_2$) genau dann, wenn für alle Probleme p der konkreten Problemlösedomäne gilt: falls $GAF_2 \triangleright p$ gilt, dann gilt auch $GAF_1 \triangleright p$. Wir nennen $\sqsupseteq$ *Wiederverwendbarkeitsrelation.*

Bemerkung 7.2 Die Wiederverwendbarkeitsrelation $\sqsupseteq$ ist eine reflexive Quasiordnung über der Menge der generalisierten abstrakten Fälle. Hierbei ist es möglich, daß verschiedene generalisierte abstrakte Fälle für die gleiche Menge von Problemen wiederverwendbar sind, d. h. die $\sqsupseteq$-Relation ist nicht antisymmetrisch.

Der Nutzen dieser Relation für die Organisation der Fallbasis besteht nun gerade darin, daß sie dazu beiträgt, die Anzahl der erforderlichen Tests auf Wiederverwendbarkeit zu reduzieren. Dies wird durch die folgende Eigenschaft präzisiert:

Lemma 7.1 (Reduzierung der Wiederverwendbarkeitstests) Seien zwei generalisierte abstrakte Fälle GAF_1 und GAF_2 gegeben und sei GAF_1 flexibler wiederverwendbarer als GAF_2 ($GAF_1 \sqsupseteq GAF_2$), so folgt: ist GAF_1 nicht zur Lösung von p wiederverwendbar, ($GAF_1 \not\triangleright p$), so ist auch GAF_2 nicht zur Lösung von p wiederverwendbar ($GAF_2 \not\triangleright p$). Insbesondere folgt: ist GAF_1 nicht zur Lösung von p schwach wiederverwendbar, ($GAF_1 \not\mathrel{\triangleright\!\!\sim} p$), so ist auch GAF_2 nicht zur Lösung von p wiederverwendbar ($GAF_2 \not\triangleright p$).

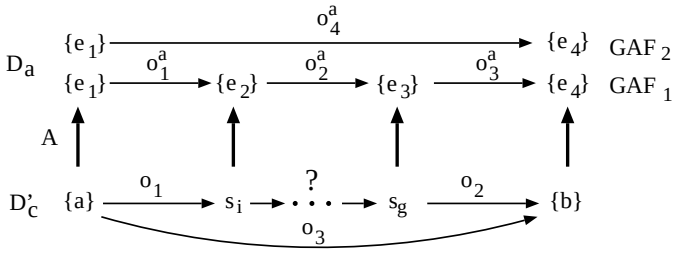

Abbildung 7.3: Hilfsabbildung zum Beweis von Lemma 7.2

Beweisskizze Die Gültigkeit dieses Lemmas ergibt sich auf triviale Weise direkt aus der Umkehrung der Definition 7.2.

Wird also während des Retrievals ein generalisierter abstrakter Fall auf schwache Wiederverwendbarkeit hin getestet und schlägt der Test fehl, so brauchen die generalisierten abstrakten Fälle der Fallbasis, die gemäß der Wiederverwendbarkeitsrelation weniger flexibel wiederverwendbar sind, nicht mehr überprüft zu werden. Hierdurch kann die Anzahl der erforderlichen Test und somit der Retrievalaufwand erheblich reduziert werden.

Hierbei stellt sich jedoch das Problem, daß die Gültigkeit der Wiederverwendbarkeitsrelation im allgemeinen unentscheidbar ist.

Lemma 7.2 (Unentscheidbarkeit der Wiederverwendbarkeitsrelation) Die Wiederverwendbarkeitsrelation ist unentscheidbar.

Beweis Wir reduzieren das Problem, ob (s_i, s_g) in $\mathcal{D}_c$ lösbar ist, auf die Frage der Gültigkeit der Wiederverwendbarkeitsrelation $GAF_1 \sqsupseteq GAF_2$ bzgl. der Problemlösedomänen $\mathcal{D}'_c$, $\mathcal{D}_a$ und der generischen Abstraktionstheorie $\mathcal{A}$. GAF_1, GAF_2, $\mathcal{D}'_c$, $\mathcal{D}_a$ und $\mathcal{A}$ werden wie folgt konstruiert (siehe Abbildung 7.3): $\mathcal{D}'_c$ entsteht aus $\mathcal{D}_c$ durch Erweiterung um drei neue Operatoren o_1, o_2, und o_3 und zwei neue essentielle Sätze a und b. Die Operatoren werden so definiert, daß sie genau die folgenden Zustandstransformationen durchführen: $\{a\} \xrightarrow{o_1} s_i$, $s_g \xrightarrow{o_2} \{b\}$ und $\{a\} \xrightarrow{o_3} \{b\}$. Die abstrakte Problemlösedomäne $\mathcal{D}_a$ enthält vier essentielle Sätze $e_1, \ldots, e_4$ sowie vier Operatoren $o_1^a, \ldots, o_4^a$, die genau die folgenden Zustandstransformationen durchführen: $\{e_1\} \xrightarrow{o_1^a} \{e_2\}$, $\{e_2\} \xrightarrow{o_2^a} \{e_3\}$, $\{e_3\} \xrightarrow{o_3^a} \{e_4\}$, $\{e_1\} \xrightarrow{o_4^a} \{e_4\}$. Die generische Abstraktionstheorie $\mathcal{A}$ enthalte folgende Klauseln: $e_1 \leftarrow a$, $e_2 \leftarrow s_i$[1], $e_3 \leftarrow s_g$ und $e_4 \leftarrow b$. Weiter konstruieren wir die folgenden beiden generalisierten abstrakten Fälle: $GAF_1 = ((\{e_1\}, \{e_4\}, \emptyset), \langle o_1^a, o_2^a, o_3^a \rangle)$ und $GAF_2 = ((\{e_1\}, \{e_4\}, \emptyset), \langle o_4^a \rangle)$. Es gilt nun: (s_i, s_g) ist in $\mathcal{D}_c$ lösbar genau dann, wenn $GAF_1 \sqsupseteq GAF_2$ gilt. Um dies zu zeigen, betrachten wir zunächst die Beweisrichtung $\Leftarrow$. Wir nehmen an, daß $GAF_1 \sqsupseteq GAF_2$ gilt. Nach Konstruktion gilt aber $GAF_2 \triangleright (\{a\}, \{b\})$, da $\{a\} \xrightarrow{o_3} \{b\}$ eine Verfeinerung

[1]Hierbei wird der Zustand s_i als Konjunktion der enthaltenen Sätze aufgefaßt.

darstellt (man beachte daß sich GAF_1 jedoch nicht zu $\{a\} \xrightarrow{o_3} \{b\}$ verfeinern läßt). Gemäß Definition 7.2 gilt dann aber auch $GAF_1 \triangleright (\{a\}, \{b\})$. Folglich ist jeder der Operatoren o_1^a, o_2^a, o_3^a verfeinerbar. Da o_2^a verfeinerbar ist, muß folglich (s_i, s_g) in $\mathcal{D}_c'$ lösbar sein und somit auch in $\mathcal{D}_c$. Abschließend betrachten wir die Beweisrichtung $\Rightarrow$. Sei (s_i, s_g) in $\mathcal{D}_c$ lösbar und somit auch in $\mathcal{D}_c'$ lösbar. Folglich gilt $GAF_1 \triangleright (\{a\}, \{b\})$. Außerdem gilt gemäß Konstruktion auch $GAF_2 \triangleright (\{a\}, \{b\})$. Da $(\{a\}, \{b\})$ gemäß Konstruktion das einzige Problem ist, für das GAF_2 wiederverwendbar ist, folgt $GAF_1 \sqsupseteq GAF_2$. Da die Lösbarkeit eines Problems bekanntermaßen unentscheidbar ist, ist auch die Wiederverwendbarkeitsrelation unentscheidbar.

7.2 Inkrementelles Lernen der Wiederverwendbarkeitshierarchie

Im folgenden stellen wir einen Ansatz vor, mit dem es möglich ist, die Wiederverwendbarkeitsrelation $\sqsubseteq$ inkrementell aus Beispielen für wiederverwendbare generalisierte abstrakte Fälle zu lernen. Hierdurch ist es zwar nicht möglich, diese Relation korrekt zu bestimmen, jedoch kann eine nützliche Approximation konstruiert werden.

7.2.1 Trainingsbeispiele zum Lernen der Wiederverwendbarkeitsrelation

Zum Lernen der Wiederverwendbarkeitsrelation sind Trainingsbeispiele für diese Relation erforderlich. In dem hier vorliegenden Szenario gibt es prinzipiell mehrere Möglichkeiten für die Bildung dieser Trainingsbeispiele.

Instanzen der $\triangleright$-Relation: Trainingsbeispiele könnten dadurch gebildet werden, daß für jeden bislang bekannten Fall $F_j \in \mathcal{F}$ und jeden bislang konstruierten generalisierten abstrakten Fall $GAF \in \Gamma$ die $\triangleright$-Relation bestimmt wird. Da die $\triangleright$-Relation unentscheidbar ist, scheidet diese Möglichkeit jedoch aus.

Instanzen der $\mathrel{\triangleright\!\!\sim}$-Relation: Alternativ zur Wiederverwendbarkeitsrelation könnte auch die schwache Wiederverwendbarkeitsrelation $\mathrel{\triangleright\!\!\sim}$ verwendet werden. Diese ist zwar keine hinreichende Bedingung für die Wiederverwendbarkeit, jedoch ist sie entscheidbar. Diese Vorgehensweise würde jedoch einen vollständigen Test für alle Paare aus $\mathcal{F} \times \Gamma$ erfordern. Außerdem müssen hierfür alle Fälle $\mathcal{F}$ gespeichert werden, um nach der Konstruktion eines neuen generalisierten abstrakten Falls die schwache Wiederverwendbarkeit mit allen bisherigen Fällen testen zu können.

Gebildete Paare (F_j, Γ_j): Eine andere Alternative für die Bildung von Trainingsbeispielen besteht in der Verwendung der Paare (F_j, Γ_j). Zu jedem Fall F_j wird die Menge Γ_i der daraus gebildeten generalisierten abstrakten Fälle betrachtet. Hierbei gilt, daß

alle generalisierten abstrakten Fälle aus Γ_j wiederverwendbar sind für das Problem p_j aus $F_j = (p_j, \bar{o_j})$, d. h. es gilt: $\forall GAF \in \Gamma_j \;\; GAF \triangleright p_j$. Dieser Ansatz hat den Vorteil, daß es nicht erforderlich ist, die Wiederverwendbarkeit separat zu überprüfen, und es ist auch nicht erforderlich die konkreten Fälle aus $\mathcal{F}$ zu speichern.

Aufgrund dieser Betrachtung ist der letzte Ansatz zur Gewinnung der Trainingsbeispiele mit besonders geringem Berechnungs- und Speicheraufwand zu realisieren. Daher wollen wir diesen Ansatz im folgenden etwas genauer betrachten.

7.2.2 Lösungserhaltende Wiederverwendbarkeit

Betrachten wir die generalisierten abstrakten Fälle Γ_j, die aus einem Fall $F_j = (p_j, \bar{o_j})$ gebildet werden, so stellen wir folgendes fest:

- Jeder generalisierte abstrakte Fall aus Γ_j ist zur Lösung von p_j wiederverwendbar.
- Die Lösung jedes generalisierten abstrakten Falls aus Γ_j kann so spezialisiert und verfeinert werden, daß die Lösung $\bar{o_j}$ entsteht.
- In Γ_j ist jeder generalisierte abstrakte Fall enthalten, der zur Lösung von p_j wiederverwendbar ist und dessen Lösung sich so spezialisieren und verfeinern läßt, daß die Lösung $\bar{o_j}$ entsteht.

Wir können also sehen, daß die generalisierten abstrakten Fälle aus Γ_j für die Lösung von p_j eine besondere Bedeutung haben, da sie es zusätzlich ermöglichen, daß durch die Wiederverwendung eine Lösung entsteht, die mit der ursprünglichen Lösung im Fall übereinstimmt. Diese Eigenschaft wollen wir nun ausnutzen, um die *lösungserhaltende Wiederverwendbarkeit* eines generalisierten abstrakten Falls zu definieren.

Definition 7.3 (Lösungserhaltende Wiederverwendbarkeit) Ein generalisierter abstrakter Fall GAF ist zur Lösung von p lösungserhaltend wiederverwendbar bzgl. $\mathcal{F}$ (wir schreiben $GAF \blacktriangleright p$), genau dann wenn:

(i) GAF zur Lösung von p wiederverwendbar ist ($GAF \triangleright p$) und
(ii) wenn es einen Fall $(p, \bar{o}) \in \mathcal{F}$ gibt, so läßt sich die Lösung von GAF so spezialisieren und verfeinern, daß $\bar{o}$ entsteht.

Bemerkung 7.3 Die lösungserhaltende Wiederverwendbarkeit ist ein Spezialfall der Wiederverwendbarkeit, d. h. es gilt: $GAF \blacktriangleright p$ impliziert $GAF \triangleright p$.

Es wird klar, daß die lösungserhaltende Wiederverwendbarkeit $\blacktriangleright$ ein besseres Kriterium für die Auswahl eines generalisierten abstrakten Falls ist, als die Wiederverwendbarkeit $\triangleright$, da für bekannte Probleme die Lösung bevorzugt wird, die schon einmal vorkam. Außerdem sind Trainingsbeispiele für die lösungserhaltende Wiederverwendbarkeitsrelation leicht zu konstruieren. Dies wird im folgenden Lemma festgehalten:

Lemma 7.3 (Trainingsbeispiele für lösungserhaltende Wiederverwendbarkeit) Ist $F_j = (p_j, \bar{o_j})$ ein Fall und ist Γ_j die Menge der daraus gebildeten generalisierten abstrakten Fälle, so gilt für jeden generalisierten abstrakten Fall GAF: $GAF \blacktriangleright p_j$ genau dann wenn $GAF \in \Gamma_j$ ist.

Wir definieren nun die Wiederverwendbarkeitsbeziehung zwischen generalisierten abstrakten Fällen mit Hilfe der lösungserhaltende Wiederverwendbarkeit. Diese Definition ersetzt die vorherige Definition 7.2.

Definition 7.4 (Wiederverwendbarkeitsbeziehung zwischen GAF's) Ein generalisierter abstrakter Fall GAF_1 ist *flexibler wiederverwendbar* als ein generalisierter abstrakter Fall GAF_2 (wir schreiben: $GAF_1 \sqsupseteq GAF_2$) genau dann, wenn für alle Probleme p der konkreten Problemlösedomäne gilt: falls $GAF_2 \blacktriangleright p$ gilt dann gilt auch $GAF_1 \blacktriangleright p$.

Für diese geänderte Wiederverwendbarkeitsbeziehung gilt nun die folgende zentrale Eigenschaft:

Lemma 7.4 (Reduzierung der Wiederverwendbarkeitstests) Seien zwei generalisierte abstrakte Fälle GAF_1 und GAF_2 gegeben und sei GAF_1 flexibler wiederverwendbar als GAF_2 gemäß Definition 7.4 ($GAF_1 \sqsupseteq GAF_2$). Es folgt: ist GAF_1 nicht zur Lösung von p schwach wiederverwendbar ($GAF_1 \not\blacktriangleright p$), so ist auch GAF_2 nicht lösungserhaltend zur Lösung von p wiederverwendbar ($GAF_2 \not\blacktriangleright p$).

Beweis Aus $GAF_1 \not\blacktriangleright p$ folgt $GAF_1 \not\triangleright p$ und hieraus folgt $GAF_1 \not\blacktriangleright p$. Somit ergibt sich mit Definition 7.4 $GAF_2 \not\blacktriangleright p$.

7.2.3 Der Hypothesenraum des Lernalgorithmus

Im folgenden versuchen wir diese Wiederverwendbarkeitsrelation zwischen den vorhandenen generalisierten abstrakten Fällen aus den Beispielen (F_j, Γ_j) zu lernen. Hierzu wird ein Lernalgorithmus vorgestellt, der folgende eingeschränkte Wiederverwendbarkeitsrelation zwischen den bekannten generalisierten abstrakten Fällen als Hypothesen für die Wiederverwendbarkeitsrelation konstruiert:

Definition 7.5 (Eingeschränkte Wiederverwendbarkeitsrelation) Seien GAF_1 und GAF_2 zwei generalisierte abstrakte Fälle und sei $\mathcal{P}$ eine Menge von Problemen der konkreten Problemlösedomäne. Wir definieren die Quasiordnung $\sqsubseteq_{\mathcal{P}}$ durch: $GAF_1 \sqsubseteq_{\mathcal{P}} GAF_2$ genau dann wenn für alle Probleme $p \in \mathcal{P}$ gilt: falls $GAF_1 \blacktriangleright p$ gilt, dann gilt auch $GAF_2 \blacktriangleright p$. Für eine gegebene Menge von Fällen $\mathcal{F}$ schreiben wir analog $GAF_1 \sqsubseteq_{\mathcal{F}} GAF_2$, falls $GAF_1 \sqsubseteq_{\mathcal{P}} GAF_2$ für die Problemmenge P aus $\mathcal{F}$ gilt.

Die Idee, die diesem Algorithmus zugrunde liegt, besteht nun darin, die eingeschränkte Wiederverwendbarkeitsrelation als Hypothese für die Wiederverwendbarkeitsrelation zu konstruieren. Die eingeschränkte Wiederverwendbarkeitsrelation kann als eine Approximation

der Wiederverwendbarkeitsrelation angesehen werden, die sich bei der Vergrößerung der Fallbasis schrittweise der Wiederverwendbarkeitsrelation annähert. Liegt eine vollständige Fallbasis $\widehat{\mathcal{F}}$ vor, so stimmt die eingeschränkte Wiederverwendbarkeitsrelation mit der Wiederverwendbarkeitsrelation überein, denn es gilt: $\sqsubseteq = \sqsubseteq_{\widehat{\mathcal{F}}}$.

7.2.4 Darstellung der Hypothesen in einer Wiederverwendbarkeitshierarchie

Im folgenden wird eine Repräsentation der eingeschränkten Wiederverwendbarkeitsrelation durch eine *Wiederverwendbarkeitshierarchie* eingeführt. Diese Hierarchie kann dann durch einen inkrementellen Lernalgorithmus, der im nachfolgenden Abschnitt beschrieben wird, aufgebaut werden.

Definition 7.6 (Wiederverwendbarkeitshierarchie) Eine Wiederverwendbarkeitshierarchie ist ein Baum mit den Knoten K_i $(i = 0, \ldots, m)$ und der Wurzel K_0. Jedem Knoten K_i ist eine Menge von abstrakten generalisierten Fällen $\mathcal{GAF}_i$ zugeordnet, die den folgenden Bedingungen genügen:

(i) $\forall_{i=0,\ldots,m} \forall_{GAF_1, GAF_2 \in \mathcal{GAF}_i} GAF_1 \sqsubseteq_{\mathcal{F}} GAF_2 \wedge GAF_2 \sqsubseteq_{\mathcal{F}} GAF_1$

(ii) Ist K_2 direkter Nachfolger von K_1, so gilt: $\forall_{GAF_1 \in \mathcal{GAF}_1} \forall_{GAF_2 \in \mathcal{GAF}_2} GAF_2 \sqsubseteq_{\mathcal{F}} GAF_1$

Diese Hierarchie ordnet also die generalisierten abstrakten Fälle gemäß der $\sqsubseteq_{\mathcal{F}}$-Quasiordnung an. Hierbei sind generalisierte abstrakte Fälle, die bzgl. dieser Quasiordnung Äquivalent sind, zusammen in einem Knoten der Hierarchie gespeichert. Die generalisierten abstrakten Fälle an einem Knoten sind immer flexibler wiederverwendbar als die an den Nachfolgerknoten.

7.2.5 Der Lernalgorithmus

Wir werden nun einen inkrementellen Algorithmus vorstellen, der für eine Menge von Fallmengen $\mathcal{F} = \{F_1, \ldots, F_n\}$ und zugehörigen generalisierten abstrakten Fällen $\Gamma_i, \ldots, \Gamma_n$ eine Wiederverwendbarkeitshierarchie aufbaut. Der Algorithmus ist inkrementell, da er bei jedem Auftreten eines neuen Beispiels (F_j, Γ_j) die bestehende Hierarchie so anpaßt, daß diese für die erweiterte Fallmenge korrekt ist. Dabei kommt dieser Algorithmus ohne die Speicherung der Fälle $\mathcal{F}$ aus. Zur Bildung der Wiederverwendbarkeitshierarchie wird im Algorithmus die folgende Eigenschaft ausgenutzt, die aus einer Umformulierung der Definition 7.6 unter Ausnutzung von Lemma 7.3 entsteht.

Lemma 7.5 (Äquivalente Definition der Wiederverwendbarkeitshierarchie) Ein Baum mit den Knoten K_i $(i = 0, \ldots, m)$ und zugehörigen generalisierten abstrakten Fällen $\mathcal{GAF}_i$ ist genau dann eine Wiederverwendbarkeitshierarchie, wenn die folgenden beiden Bedingungen gelten:

(i) $\forall_{j=1,\ldots,n} \forall_{i=0,\ldots,m} \mathcal{GAF}_i \subseteq \Gamma_j \vee \mathcal{GAF}_i \cap \Gamma_j = \emptyset$ und
(ii) Ist K_2 direkter Nachfolger von K_1, so gilt: $\forall_{j=1,\ldots,n}$ wenn $\mathcal{GAF}_2 \subseteq \Gamma_j$ dann auch $\mathcal{GAF}_1 \subseteq \Gamma_j$.

Beweisskizze Unter Berücksichtigung von Lemma 7.3 folgt, daß Bedingung (i) aus Lemma 7.5 ist eine äquivaltene Umformung von Bedingung (i) aus Definition 7.6. Ebenso ist Bedingung (ii) aus Lemma 7.5 ist eine äquivaltene Umformung von Bedingung (ii) aus Definition 7.6.

Der Lernalgorithmus erhält als Eingabe eine bestehende Wiederverwendbarkeitshierarchie K_0 sowie ein neues Trainingsbeispiel in Form des Paares (F_{neu}, Γ_{neu}). Der Algorithmus paßt die bestehende Hierarchie in zwei Schritten (*Update* und *Insert*) so an, daß das neue Beispiel berücksichtigt ist:

Update: Die bestehende Hierarchie wird so angepaßt, daß die beiden Bedingungen, die in Lemma 7.5 formuliert sind, auch für das neue Trainingsbeispiel gelten.

Insert: Die bestehende Hierarchie wird um die generalisierten abstrakten Fälle aus Γ_{neu}, die bislang noch nicht in der Wiederverwendungshierarchie gespeichert sind, erweitert.

Der Algorithmus für den Update-Schritt

Um beim Eintreffen eines neuen Beispiels zu überprüfen, ob die aktuelle Hierarchie korrekt ist, muß diese (in Vorordnung) durchlaufen werden. Hierbei wird zunächst geprüft, ob die generalisierten abstrakten Fälle an einem Knoten entweder alle für das aktuelle Beispiel lösungserhaltend wiederverwendbar sind ($\mathcal{GAF}_i \subseteq \Gamma_{neu}$) oder aber alle nicht lösungserhaltend wiederverwendbar sind ($\mathcal{GAF}_i \cap \Gamma_{neu} = \emptyset$). Liegt der Fall vor, daß ein Knoten sowohl lösungserhaltend wiederverwendbare als auch nicht lösungserhaltend wiederverwendbare Fälle enthält, so muß die Hierarchie an dieser Stelle „repariert“ werden, damit die erste Bedingung aus Lemma 7.5 wieder erfüllt ist. Hierzu wird dieser Knoten in zwei neue Knoten geteilt, wie dies in Abbildung 7.4 verdeutlicht wird. Die Fälle, die dem neuen oberen Knoten (K_{21}) zugewiesen werden, sind alle Fälle aus K_2, die für das aktuelle Problem lösungserhaltend wiederverwendbar sind. Alle Fälle, die dem neuen unteren Knoten (K_{22}) zugewiesen werden, sind alle Fälle aus K_2, die für das aktuelle Problem nicht lösungserhaltend wiederverwendbar sind. Diese Operation erzeugt einen korrekten Teilpfad K_1, K_{21}, K_{22} für das neue Beispiel und erhält die Korrektheit bezüglich aller vorherigen Beispiele.

In einem weiteren Schritt muß die Bedingung (ii) aus Lemma 7.5 überprüft werden. Hierzu wird getestet, ob es einen Knoten K_i gibt, dessen generalisierte abstrakte Fälle für das aktuelle Trainungsbeispiel lösungserhaltend wiederverwendbar sind ($\mathcal{GAF}_i \subseteq \Gamma_{neu}$), obwohl die generalisierten abstrakten Fälle seines Vorgängerknotens K_j nicht lösungserhaltend wiederverwendbar sind ($\mathcal{GAF}_j \cap \Gamma_{neu} = \emptyset$). In diesem Fall erfolgt eine Korrektur dadurch, daß der Knoten K_i herausgenommen und als Nachfolger des nächsten Vorgängers eingefügt

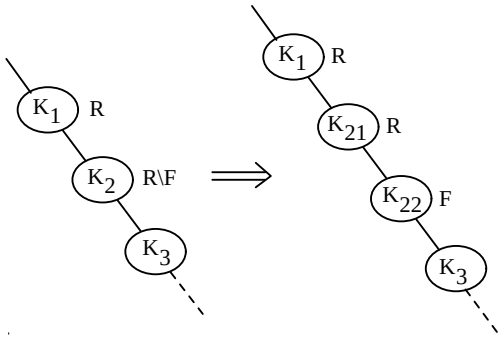

Abbildung 7.4: Das Aufteilen eines Mehrfachknotens in der Wiederverwendbarkeitshierarchie: R bezeichnet einen Knoten mit wiederverwendbaren GAF's, F bezeichnet einen Knoten mit nicht-wiederverwendbaren GAF's, und R/F bezeichnet einen Knoten, der sowohl wiederverwendbare als auch nicht-wiederverwendare GAF's enthält.

wird, dessen Fälle ebenfalls lösungserhaltend wiederverwendbar sind. Dieses „Hochziehen" des Knotens an die erste passende Stelle ist in Abbildung 7.5 beispielhaft dargestellt. Hier sind die Fälle am Knoten K_4 lösungserhaltend wiederverwendbar, obwohl die Fälle am Vorgängerknoten K_3 nicht lösungserhaltend wiederverwendbar sind. Folglich wird der Knoten K_4 entfernt und als Nachfolger von K_2 angehängt, da K_2 der nächsten Vorgängerknoten ist, dessen Fälle lösungserhaltend wiederverwendbar sind.

In Abbildung 7.6 ist der vollständige Algorithmus für den beschriebenen Update-Schritt angegeben. Dieser Algorithmus realisiert im wesentlichen die beiden zuvor beschriebenen Operationen, die jedoch aus Effizienzgründen ineinander verzahnt sind. Der Aufruf dieser Prozedur erfolgt durch: Update$(K_0, \Gamma_{neu}, true, K_0)$.

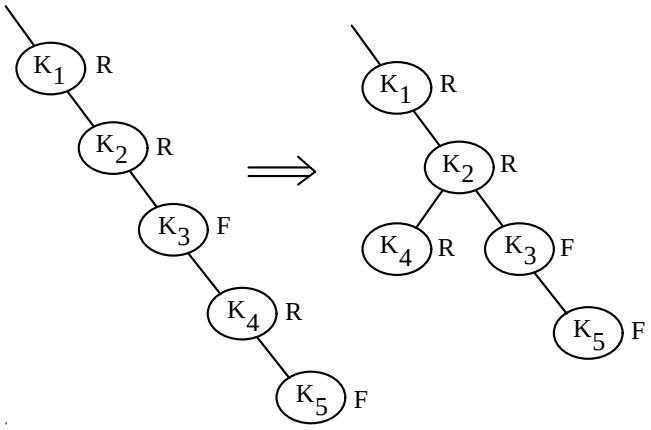

Abbildung 7.5: Das „Hochziehen" eines (Teil)-Knotens zur Korrektur der Wiederverwendbarkeitshierarchie

Eingabe: Eine Wiederverwendbarkeitshierarchie K_i, ein Trainingsbeispiel Γ_{neu}, sowie zwei Hilfsargument OK_{Flag} und K_{append}.

Ausgabe: Eine korrigierte Wiederverwendbarkeitshierarchie K_i

procedure Update($K_i, \Gamma_{neu}, Ok_{Flag}, K_{append}$)
if Ok_{Flag} **then** (* Vorheriger Knoten enthält nur wiederverwendbare Fälle *)
  if $GAF_i \subseteq \Gamma_{neu}$ **then** (* Knoten K_i enthält nur wiederverwendbare Fälle *)
    if Es existieren keine Nachfolger von K_i **then return**
    for all K_j Nachfolger von K_i **do** Update($K_j, \Gamma_{neu}, true, K_i$)
  else if $GAF_i \cap \Gamma_{neu} = \emptyset$ **then** (* Knoten K_i enthält keine wiederverwendbaren Fälle *)
    if Es existieren keine Nachfolger von K_i **then return**
    for all K_j Nachfolger von K_i **do** Update($K_j, \Gamma_{neu}, false, K_{append}$)
  else (* Knoten sowohl wiederverwendbare als auch nicht-wiederverwendbare Fälle *)
    Teile Knoten K_i in zwei neue Knoten K_u und K_n mit:
      a) Vorgänger von K_i wird Vorgänger von K_u
      b) K_n wird Nachfolger von K_u
      c) Alle Nachfolger von K_i werden Nachfolger von K_n
      d) $GAF_u := GAF_i \cap \Gamma_{neu}$
      e) $GAF_n := GAF_i - GAF_u$
    if Es existieren keine Nachfolger von K_i **then return**
    for all K_j Nachfolger von K_n **do** Update($K_j, \Gamma_{neu}, false, K_u$)
else (* Vorgängerknoten enthält keine wiederverwendbaren Fälle *)
  if $GAF_i \cap \Gamma_{neu} = \emptyset$ **then** (* Knoten K_i enthält keine wiederverwendbaren Fälle *)
    if Es existieren keine Nachfolger von K_i **then return**
    for all K_j Nachfolger von K_i **do** Update($K_j, \Gamma_{neu}, false, K_{append}$)
  else (* K_i enthält wiederverwendbare Fälle *)
    1. Erzeuge einen Knoten K_{neu} als Nachfolger von K_{append} mit:
       $GAF_{neu} := GAF_i \cap \Gamma_{neu}$
    2. Setze $GAF_i := GAF_i - \Gamma_{neu}$
    if $GAF_i = \emptyset$ **then** (* leeren Knoten entfernen *)
      1. Entferne K_i
      2. Die Nachfolger von K_i werden die Nachfolger von $K_i's$ Vorgänger
      3. **for all** K_j Nachfolger von K_i **do** Update($K_j, \Gamma_{neu}, false, K_{append}$)

Abbildung 7.6: Algorithmus zur Realisierung des Update-Schrittes

7.2.6 Der Algorithmus für den Insert-Schritt

Nachdem die Wiederverwendbarkeitshierarchie bezüglich des neuen Beispiels korrekt ist, müssen nun noch alle generalisierten abstrakten Fälle des neuen Beispiels, die noch nicht

im Baum enthalten sind, hinzugefügt werden. Dieses Einfügen kann im Prinzip an einer beliebigen Stelle geschehen, solange die Hierarchie danach weiterhin korrekt ist. Zum Bestimmen eines guten Einfügepunktes wird eine Heuristik verwendet, die einen möglichst tiefen Baum erzeugen soll. Hierzu wird, ausgehend von der Wurzel des Baumes, solange im Baum hinabgestiegen, bis die generalisierten abstrakten Fälle eines Knotens für das aktuelle Beispiel nicht mehr wiederverwendbar sind oder ein Blattknoten erreicht wurde. Ist eine solche Stelle im Baum gefunden, wird ein neuer Knoten angehängt und ihm die neuen generalisierten abstrakten Fälle zugewiesen. Der vollständige Algorithmus für das Einsortieren der neuen Fälle ist in Abbildung 7.7 dargestellt. Der Aufruf dieser Prozedur erfolgt durch Insert$(K_0, (\Gamma_{neu} - \bigcup_{i=0}^{m} \mathcal{GAF}_i), \Gamma_{neu})$.

Eingabe: Eine Wiederverwendbarkeitshierarchie mit dem Wurzelknoten K_i, das Trainingsbeispiel Γ_{neu}, sowie die Menge der generalisierten abstrakten Fälle Γ_{Rest} aus Γ_{neu} die bislang noch nicht in der Hierarchie vorgekommen sind.

Ausgabe: Eine erweiterte Wiederverwendbarkeitshierarchie.

procedure Insert$(K_i, \Gamma_{Rest}, \Gamma_{neu})$
if Es existieren keine Nachfolger von K_i **then**
  Erzeuge einen neuen Knoten K_j als Nachfolger von K_i mit:
  $GAF_j := GAF_{Rest}$
else
  Wähle einen Nachfolger K_j von K_i mit $\mid GAF_j \cap \Gamma_{neu} \mid$ maximal
  if $\mid GAF_j \cap \Gamma_{neu} \mid > 0$ **then** (* Steige tiefer in den Baum ein *)
    Insert$(K_j, \Gamma_{Rest}, \Gamma_{neu})$
  else
    Erzeuge einen Knoten K_j als Nachfolger von K_i mit:
    $GAF_j := GAF_{Rest}$

Abbildung 7.7: Algorithmus zur Realisierung des Insert-Schrittes.

7.2.7 Ein Beispiel

Wir wollen diesen Lernalgorithmus an einem einfachen Beispiel verdeutlichen. Hierzu nehmen wir an, daß die fünf Fälle $F_1, \ldots, F_5$ vorliegen. Aus jedem dieser Fälle wird eine Menge von generalisierten abstrakten Fällen generiert. Tabelle 7.1 zeigt die Zuordnung von Fällen zu den generalisierten abstrakten Fällen. Weiter wollen wir annehmen, daß die Trainingsbeispiele in der Reihenfolge F_2, F_5, F_1, F_4, F_3 eintreffen. Abbildung 7.8 zeigt die resultierende Folge von Klassifikationsbäumen.

	GAF_1	GAF_2	GAF_3	GAF_4	GAF_5
F_1	X	X			
F_2	X	X	X		
F_3	X	X	X		
F_4	X			X	
F_5	X			X	X

Tabelle 7.1: Ein Beispiel zur lösungserhaltenden Wiederverwendbarkeit generalisierter abstrakter Fälle.

Nachdem der erste Fall F_2 bearbeitet wurde, wird genau ein Knoten erzeugt, der alle generalisierten abstrakten Fälle (GAF_1, GAF_2, GAF_3), die aus diesem Planungsfall hervorgegangen sind, enthält (Baum a). Bei dem zweiten Planungsfall (F_5) stellt nun die `update` Prozedur fest, daß der bisherige Knoten sowohl wiederverwendbare Fälle (nämlich GAF_1) enthält als auch nicht wiederverwendbare Fälle (GAF_1 und GAF_2). Deshalb wird der Knoten zunächst geteilt (siehe Baum b). Danach werden beim `Insert`-Schritt die fehlenden generalisierten abstrakten Fälle GAF_4 und GAF_5 zusammen an einem Knoten angehangen (siehe Baum c). Das Bearbeiten des Planungsfalls F_1 führt nun dazu, daß der Knoten mit GAF_2 und GAF_3 geteilt wird (Baum d). Weitere generalisierte abstrakte Fälle kommen nun nicht mehr hinzu. Der Fall F_4 bewirkt nun lediglich noch ein Aufteilen eines Knotens (siehe Baum e), während der letzte Fall F_3 den Baum überhaupt nicht mehr verändert

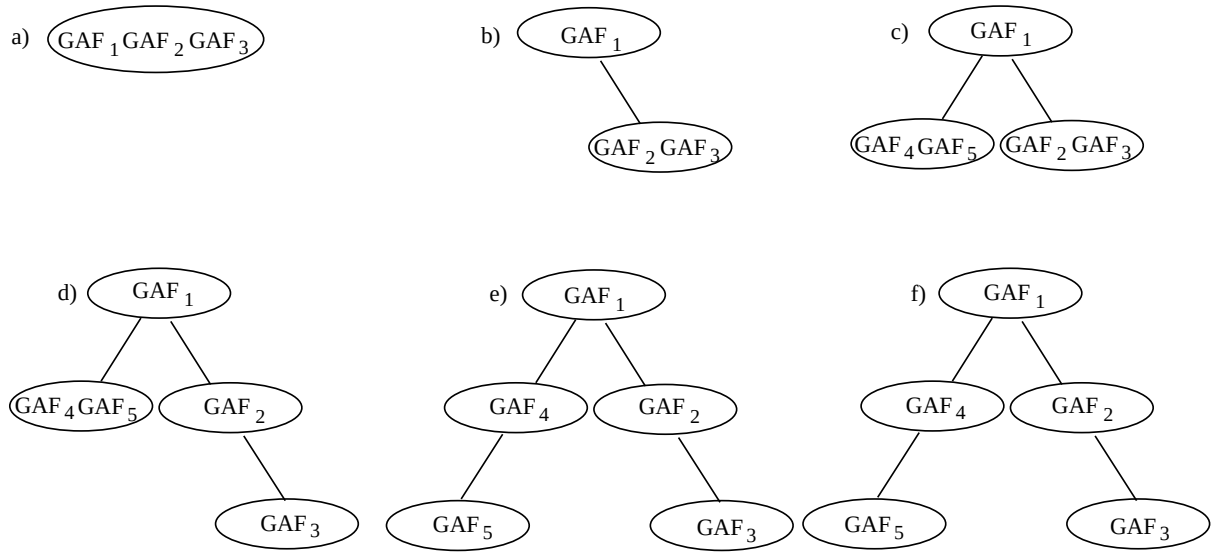

Abbildung 7.8: Eine Beispielfolge von Wiederverwendbarkeitshierarchien für die Folge F_2, F_5, F_1, F_4, F_3.

(Baum f).

Ein weiteres Beispiel, daß die inkrementelle Erweiterung einer Wiederverwendbarkeitshierarchie für die Domäne der Arbeitsplanung beschreibt, ist in Kapitel 8, Abschnitt 8.3.3 zu finden.

7.3 Ein Kostenmodell für die Wiederverwendung generalisierter abstrakter Fälle

Im vorausgehenden Abschnitt haben wir gezeigt, wie eine Fallbasis aus generalisierten abstrakten Fällen mit Hilfe der Wiederverwendbarkeitsrelation sturkturiert werden kann. Diese Strukturierung erlaubt es, die Suche nach wiederverwendbaren Fällen auf bestimmte Bereiche der Fallbasis zu beschränken. Jedoch bietet sie noch keine Möglichkeit, zwischen verschiedenen wiederverwendbaren generalisierten abstrakten Fällen zu differenzieren, bzw. diese zu bewerten. Im folgenden stellen wir daher einen Ansatz vor, der eine solche Bewertung leistet.

7.3.1 Abhängigkeit des Wiederverwendungsaufwandes von der Flexibilität der Wiederverwendbarkeit

Zunächst wollen wir versuchen, einen Zusammenhang herzustellen zwischen der Flexibilität der Wiederverwendbarkeit eines generalisierten abstrakten Falls und dem Aufwand, den dessen Wiederverwendung verursacht. Einen ähnlichen Zusammenhang haben wir bereits in Abschnitt 4.3.3 hergestellt.

Wir müssen zwischen dem Aufwand zur Überprüfung der schwachen Wiederverwendbarkeit (der die Lösungsspezialisierung bereits enthält) und dem Aufwand zur Lösungsverfeinerung unterscheiden. Im Abbildung 4.11 wurde bereits der Zusammenhang zwischen dem Abstraktionsgrad des generalisierten abstrakten Falls und den entsprechenden Aufwänden dargestellt. Dieser Zusammenhang bleibt auch erhalten, wenn anstelle des Abstraktionsgrades die Flexibilität Wiederverwendbarkeit bzgl. der $\sqsupseteq$-Relation betrachtet wird. Ist ein generalisierter abstrakter Fall GAF_1 flexibler wiederverwendbar als ein generalisierter abstrakter Fall GAF_2 ($GAF_1 \sqsupseteq GAF_2$), so liegt das daran, daß GAF_2 auf einer höheren Abstraktionsebene liegt als GAF_1 und/oder daran, daß GAF_2 stärker generalisiert ist als GAF_1.

Tabelle 7.2 zeigt, wie sich die Kosten für den Test der schwachen Wiederverwendbarkeit ($C_{\triangleright}$ genannt) und die Kosten für die Lösungsverfeinerung (C_v genannt) verhalten. Die in dieser Tabelle angegebenen Relationen beschreiben keinen exakten Zusammenhang sondern einen tendentiellen. Abbildung 7.9 faßt die beiden Kosten zusammen und zeigt das Verhalten der Gesamtkosten. Man erkennt, daß weder der generalisierte abstrakte Fall, der am flexibelsten wiederverwendbar ist, noch der generalisierte abstrakte Fall, der am unfle-

	Wiederverwendbarkeitstests	Lösungsverfeinerung
GAF_1 genereller als GAF_2	$C_{\triangleright}(GAF_1) > C_{\triangleright}(GAF_2)$	$C_v(GAF_1) = C_v(GAF_2)$
GAF_1 abstrakter als GAF_2	$C_{\triangleright}(GAF_1) \ll C_{\triangleright}(GAF_2)$	$C_v(GAF_1) > C_v(GAF_2)$
$GAF_1 \sqsupseteq GAF_2$	in der Regel: $C_{\triangleright}(GAF_1) < C_{\triangleright}(GAF_2)$	$C_v(GAF_1) \geq C_v(GAF_2)$

Tabelle 7.2: Vergleich der Kosten für den Wiederverwendbarkeitstest und die Lösungsverfeinerung in Abhängigkeit der Flexibilität der Wiederverwendbarkeit des eingesetzten generalisierten abstrakten Falls.

xibelsten wiederverwendbar ist, die geringsten Gesamtkosten aufweist. Der generalisierte abstrakte Fall, der sich mit den geringsten Gesamtkosten wiederverwenden läßt, wird sich somit auf einer mittleren Ebene befinden. Daher bietet es sich an, bei der Suche nach einem geeigneten Fall von dem am flexibelsten wiederverwendbaren Fall auszugehen und dann schrittweise weniger flexibel wiederverwendbare Fälle in Betracht zu ziehen. Das bietet den zusätzlichen Vorteil, daß der Aufwand für den Wiederverwendbarkeitstest anfänglich sehr gering ist und erst im weiteren Verlauf der Suche ansteigt.

Bei dieser Betrachtung muß jedoch der Vorteil durch die Verwendung eines besseren Falls dem hierzu erforderlichen erhöhten Suchaufwand gegenübergestellt werden. Wie bereits in Abschnitt 4.3.4 und Abbildung 4.12 gezeigt wurde, ist es nicht unbedingt sinnvoll, den generalisierten abstrakten Fall wiederzuverwenden, der den geringsten Wiederverwen-

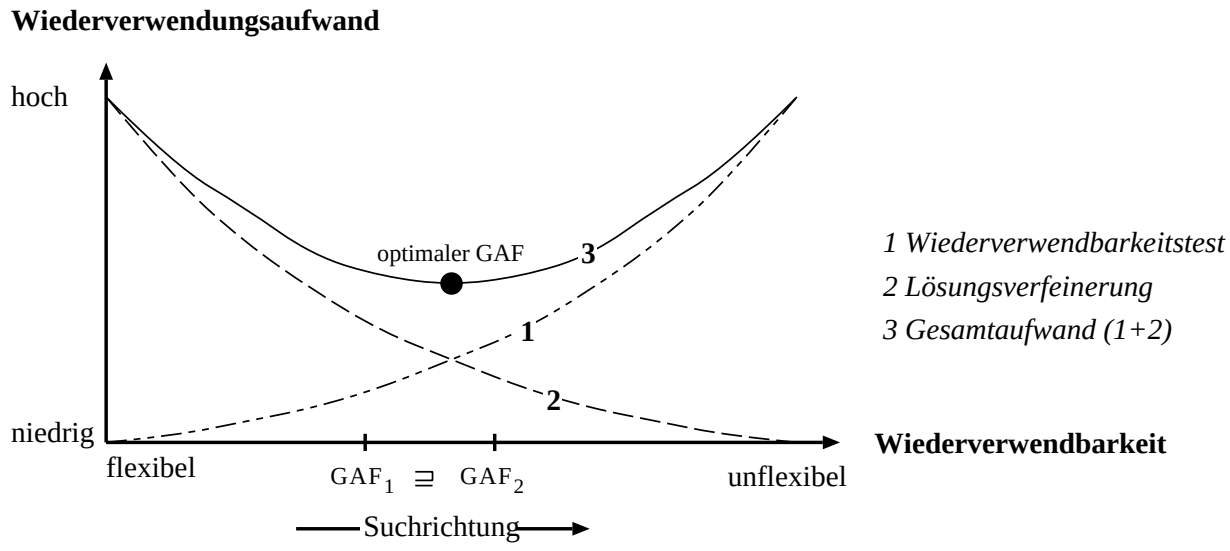

Abbildung 7.9: Aufwand bei der Wiederverwendung generalisierter abstrakter Fälle in Abhängigkeit der Flexibilität der Wiederverwendbarkeit

dungsaufwand verursacht. Es kann durchaus günstiger sein, einen suboptimalen Fall zu verwenden falls dieser in erheblich kürzerer Retrievalzeit gefunden werden kann. Diesen Zusammenhang wollen wir im folgenden genauer untersuchen und entwicklen hierzu ein detailliertes Kostenmodell, das den Retrievalaufwand sowie die Wiederverwendungskosten gleichermaßen berücksichtigt.

7.3.2 Kostenmodell

Für die Leistungsfähigkeit eines Problemlösers ist die *Problemlösezeit* entscheidend. Die Problemlösezeit ist die gesamte Zeit, von der Eingabe der Problemstellung bis zur Ausgabe der Lösung. Dabei ist die Problemlösezeit für ein einzelnes Problem in der Regel weniger bedeutsam als die mittlere Problemlösezeit für eine Menge von Problemen. Daher wollen wir im folgenden den Begriff des *Erwartungswerts für die Problemlösezeit* einführen.

Definition 7.7 (Erwartungswert für die Problemlösezeit) Der Erwartungswert für die Problemlösezeit für eine Menge von Problemen $\mathcal{P}$, denen eine bestimmten Verteilung zugrunde liegt, ergibt sich zu:

$$T_{EW}(\mathcal{P}) = \sum_{p \in \mathcal{P}} Pr(p) \cdot T_L(p).$$

Dabei ist $Pr(p)$ die Wahrscheinlichkeit, mit der das Problem p gelöst werden muß und $T_L(p)$ ist die Problemlösezeit für p.

Ziel dieses Kostenmodells ist es, eine Abschätzung für diesen Erwartungswert für bestimmte Retrievalvarianten zu geben und somit die Auswahl einer möglichst optimalen Variante zu ermöglichen. Hierzu müssen wir zunächst die grundlegende Retrievalstrategie, sowie die zu betrachtenden Varianten festlegen.

7.3.3 Retrievalstrategie

Wie bereits im vorherigen Abschnitt erörtert wurde, sollte die Suche während des Retrievals vom generalisierten abstrakten Fall mit der höchsten Wiederverwendbarkeit ausgehen und dann schrittweise weniger wiederverwendbare Fälle in Betracht zu ziehen. Diese Strategie kann durch Ausnutzung der Wiederverwendbarkeitshierarchie auf einfache Weise realisiert werden. Hierzu wird dieser Baum, ausgehend von der Wurzel, in Vorordnung durchlaufen. Für jeden betrachteten Knoten wird dann die schwache Wiederverwendbarkeit eines enthaltenen generalisierten abstrakten Falls überprüft. Ist die Wiederverwendbarkeit gegeben, so werden die Nachfolger des Knotens in einer bestimmten Reihenfolge betrachtet. Ist eine bestimmte Stufe der Hierarchie erreicht, so bricht das Retrieval mit den generalisierten abstrakten Fällen dieses Knotens ab. Generalisierte abstrakte Fälle, die unterhalb dieses Knotens in der Hierarchie liegen, werden dann nicht mehr betrachtet. Zur Realisierung

dieser Strategie muß die Wiederverwendbarkeitshierarchie um zusätzliche Informationen wie folgt erweitert werden:

Definition 7.8 (Erweiterung der Wiederverwendbarkeitshierarchie) Die Definition der Wiederverwendbarkeitshierarchie (Definition 7.6) wird wie folgt erweitert:

(i) Jeder Knoten K besitzt zusätzlich eine totale Ordnung $\succ$ (Knotenordnung genannt), die alle direkten Nachfolger $K_1, \ldots, K_r$ des Knotens K anordnet.
(ii) Jeder Knoten K besitzt zusätzlich ein Prädikat S_K , das Selektionsprädikat genannt wird.

Bemerkung 7.4 Zur Vereinfachung wollen wir nachfolgend davon ausgehen, daß die Knotenindizes so vergeben sind, daß $K_{i+1} \succ K_i$ gilt. Mit dem Seletionsprädikat S_K wird der Knoten K selektiert bzw. ausgeblendet.

Der in Abbildung 7.10 dargestellte rekursive Algorithmus, realisiert das zuvor skizzierte Retrieval. Hierbei wird das Retrieval durch die Wahl der Knotenordnung und durch die Wahl des Selektionsprädikates wesentlich beeinflußt. Die Knotenordnung bestimmt die Reihenfolge, mit der die generalisierten abstrakten Fälle durchsucht werden. Diese hat einen erheblichen Einfluß auf die Anzahl der Wiederverwendbarkeitstests, die erforderlich sind, bis ein geeigneter generalisierter abstrakter Fall gefunden ist. Das Selektionsprädikat bewirkt ein Ausblenden von Teilbäumen aus der Klassifikationshierarchie. Dies kann einerseits dazu führen, daß der Retrievalaufwand reduziert wird, andererseits jedoch auch die

---

Eingabe: Eine Wiederverwendbarkeitshierarchie K und eine Problem p.
Ausgabe: Ein Knoten, der schwach wiederverwendbare generalisierte abstrakte Fälle enthält.

procedure retrieve(K, p)
  if K hat selektierte Nachfolger **then**
    Seien $K_1, \ldots, K_r$ die selektierten Nachfolger von K mit $K_{i+1} \succ K_i$.
    $i := 0$
    repeat
      $i := i + 1$
      Wähle $GAF \in \mathcal{GAF}_i$
      until $GAF \trianglerighteq p$ oder $i = r$
    if $GAF \trianglerighteq p$ **then return** retrieve(K_i, p)
  return K

Abbildung 7.10: Algorithmus zum Retrieval von generalisierten abstrakten Fällen

Konsequenz haben, daß Fälle, die mit geringem Aufwand wiederverwendet werden können, nicht mehr gefunden werden.

7.3.4 Ermittlung des Erwartungswertes der Problemlösezeit

Im folgenden bestimmen wir den Erwartungswert der Problemlösezeit für eine gegebene Retrievalvariante (Knotenordnung und Selektionsprädikat). Hierzu müssen wir zunächst die folgenden Bezeichnungen für Zeitaufwände und Wiederverwendbarkeitswahrscheinlichkeiten einführen.

Bezeichnung 7.1 (Charakteristische Zeitaufwände und Wahrscheinlichkeiten) Sei p ein Problem und sei GAF der im Retrieval ausgewählte generalisierte abstrakte Fall aus dem Knoten K der Abstraktionshierarchie.

- Mit $T_{\triangleright}(K,p)$ bezeichnen wir den Zeitaufwand, der zum Test von $GAF \gtrsim p$ erforderlich ist.
- Mit $T_v(K,p)$ bezeichnen wir den Zeitaufwand, der zur Verfeinerung von GAF zu einer Lösung von p erforderlich ist.
- Seien $K_1,\ldots,K_r$ die direkten selektierten Nachfolger von K und seien $GAF_1,\ldots,GAF_r$ die für diese Knoten im Retrieval ausgewählten generalisierten abstrakten Fälle, dann bezeichne $P_j(K,p) = P(GAF_1 \not\gtrsim p \wedge GAF_2 \not\gtrsim p \wedge \cdots \wedge GAF_{j-1} \not\gtrsim p \wedge GAF_j \gtrsim p \,|\, GAF \gtrsim p)$ die Wahrscheinlichkeit, daß $GAF_1,\ldots,GAF_{j-1}$ nicht schwach wiederverwendbar sind, jedoch GAF_j schwach wiederverwendbar ist unter der Voraussetzung, daß GAF schwach wiederverwendbar ist.

Bemerkung 7.5 Man beachte, daß die Verteilung der Probleme, die in Definition 7.7 durch die Wahrscheinlichkeit $Pr(p)$ ausgedrückt ist, die Wahrscheinlichkeiten $P_j(K,p)$ bestimmt.

Abbildung 7.11 zeigt noch einmal die definierten Zeitaufwände und Wahrscheinlichkeiten. Anhand dieser Abbildung wollen wir nun die Berechnung des Erwartungswertes für die Problemlösezeit zur Lösung von p erörtern. Die Problemlösezeit $T_{EW}(K,p)$ zur Lösung von p mit der Wiederverwendbarkeitshierarchie K setzt sich aus den folgenden drei Komponenten zusammen:

1. *Zeitaufwand zum Test der schwachen Wiederverwendbarkeit eines generaliserten abstrakten Falls GAF aus K_i.* Dieser ergibt sich zu

$$\sum_{i=1}^{r}(T_{\triangleright}(K_i,p)\cdot(1-\sum_{j=1}^{i-1}P_j(K,p)\,)).$$

   Hierbei ist $1-(P_1(K,p)+P_2(K,p)+,\cdots,+P_{i-1}(K,p))$ die Wahrscheinlichkeit dafür, daß der generalisierte abstrakte Fall von K_i auf Wiederverwendbarkeit geprüft wird.

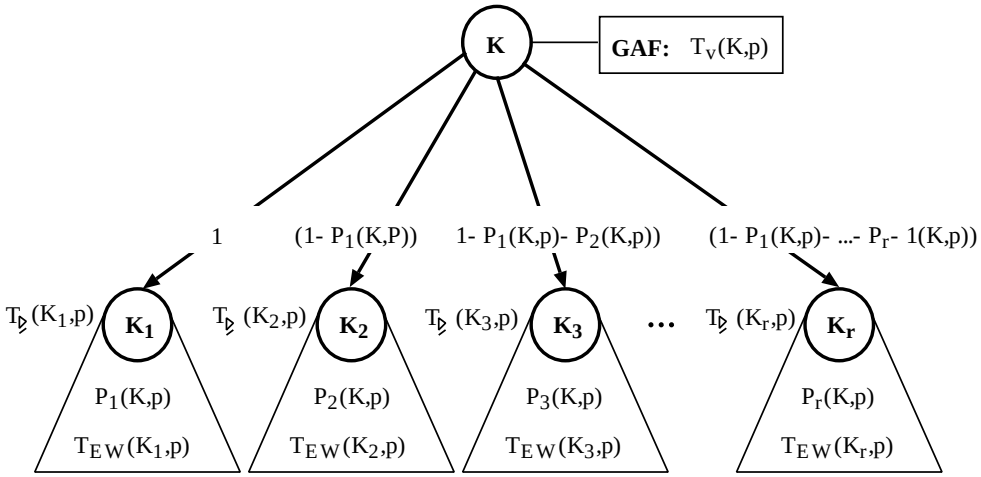

Abbildung 7.11: Zeitaufwände und Wahrscheinlichkeiten für die Betrachtung eines Knotens

Dies ist genau dann der Fall, wenn die vorhergehenden Wiederverwendbarkeitstests fehlgeschlagen sind.

2. *Zeitaufwand zur Lösung von p durch einen generaliserten abstrakten Fall im Teilbaum von K_i.* Dieser Zeitaufwand bestimmt sich rekursiv durch die Problemlösezeit $T_{EW}(K_i, p)$, gewichtet mit der Wahrscheinlichkeit $P_i(K, p)$ mit der der Teilbaum zur Lösung verwendet wird. Hieraus ergibt sich der Anteil:

$$\sum_{i=1}^{r} T_{EW}(K_i, p) \cdot P_i(K, p).$$

3. *Zeitaufwand zur Lösung von p durch einen generalisierten abstrakten Fall aus K.* Dieser Zeitaufwand beschreibt die Situation, daß keiner der Teilbäume zur Lösung verwendendet und folglich der generalisierte abstrakte Fall von K herangezogen wird. Hieraus ergibt sich der Anteil:

$$T_v(K, p) \cdot (1 - \sum_{i=1}^{r} P_i(K, p))$$

Zusammenfassend ergibt sich nun die gesamte Problemlösezeit zur Lösung von p mit der Wiederverwendbarkeitshierarchie K wie folgt:

$$T_{EW}(K,p) = \begin{cases} T_v(K,p) & K \text{ besitzt keine Nachfolger,} \\ \sum_{i=1}^{r} T_{EW}(K_i,p) \cdot P_i(K,p) + & K \text{ hat die selektierten Nachfolger} \\ \sum_{i=1}^{r} T_{\triangleright}(K_i,p) \cdot (1 - \sum_{j=1}^{i-1} P_j(K,p)) + & K_1, \ldots, K_r \text{ mit } K_{i+1} \succ K_i \\ T_v(K,p) \cdot (1 - \sum_{i=1}^{r} P_i(K,p)) & \end{cases}$$

In diese Formel gehen die Selektionsprädikate dadurch ein, daß sie festlegen, welche Knoten K_i als Nachfolger von K betrachtet werden. Die Knotenordnung beeinflußt die eingehenden Wahrscheinlichkeiten $P_i(K,p)$ (siehe Bezeichnung 7.1).

7.3.5 Schätzung des Erwartungswertes

Die a priori Bestimmung der Zeitaufwände $T_{\triangleright}$ und T_v sowie der Wahrscheinlichkeiten P_i ist für neue Probleme kaum möglich. Jedoch ist eine *Schätzung* dieser Werte auf der Grundlage der Problemlösung für Probleme aus den bisher bekannten Fällen $\mathcal{F}$ möglich. Hierzu werden die Zeitaufwände gemessen, die bei der Wiederverwendung eines generalisierten abstrakten Falls entstehen.

Zunächst müssen wir für jeden Knoten K der Wiederverwendbarkeitshierarchie die Teilmenge $\mathcal{F}_K$ der Fälle $\mathcal{F}$ bestimmen, deren Probleme durch lösungserhaltende Wiederverwendung eines generalisierten abstrakten Falls aus diesem Knoten gelöst werden können, d. h. $\mathcal{F}_K = \{(p,\bar{o}) | (p,\bar{o}) \in \mathcal{F} \text{ und } GAF \blacktriangleright p\}$.

Bezeichnung 7.2 (Schätzung von Zeitaufwänden und Wahrscheinlichkeit) Ist K ein Knoten der Wiederverwendbarkeitshierarchie und ist GAF ein generalisierter abstrakter Fall aus K.

(i) Ist $t_{\triangleright}(GAF,p)$ der Zeitaufwand zum Test der Wiederverwendbarkeit von GAF für $p \in \mathcal{F}_K$, so nennen wir

$$\widehat{T_{\triangleright}}(K) = \frac{1}{|\mathcal{F}_K|} \cdot \sum_{(p,\bar{o}) \in \mathcal{F}_K} t_{\triangleright}(GAF,p)$$

eine Schätzung für $T_{\triangleright}(K,p)$.

(ii) Ist $t_v(GAF,p)$ der Zeitaufwand zur Verfeinerung von $p \in \mathcal{F}_K$, so nennen wir

$$\widehat{T_v}(K) = \frac{1}{|\mathcal{F}_K|} \cdot \sum_{(p,\bar{o}) \in \mathcal{F}_K} t_v(GAF,p)$$

eine Schätzung für $T_v(K,p)$.

(iii) Besitzt der Knoten K die selektierten Nachfolger $K_1, \ldots, K_r$, mit $K_{i+1} \succ K_i$, so nennen wir

$$\widehat{P}_i(K) = \frac{|\mathcal{F}_{K_i} - \bigcup_{j=1}^{i-1} \mathcal{F}_{K_j}|}{|\mathcal{F}_K|}$$

eine Schätzung für $P_i(K, p)$.

Diese Schätzungen lassen sich einfach berechnen. Hierzu ist es lediglich erforderlich, für jeden generalisierten abstrakten Fall GAF, der durch die Fallabstraktion und -generalisierung aus einem Fall $F = (p, \bar{o})$ entstanden ist, die entsprechenden Zeitaufwände $t_{\triangleright}(GAF, p)$ und $t_v(GAF, p)$ zu messen.

Aufgrund dieser Schätzungen läßt sich nun eine Schätzung für den Erwartungswert für die Zeit zur Lösung eines Problems mit der Wiederverwendbarkeitshierarchie K wie folgt bestimmen:

$$\widehat{T_{EW}}(K) = \begin{cases} \widehat{T}_v(K) & K \text{ besitzt keine Nachfolger,} \\ \sum_{i=1}^{r} \widehat{T_{EW}}(K_i) \cdot \widehat{P}_i(K) + & K \text{ hat die selektierten Nachfolger} \\ \sum_{i=1}^{r} \widehat{T_{\triangleright}}(K_i) \cdot (1 - \sum_{j=1}^{i-1} \widehat{P}_j(K)) + & K_1, \ldots, K_r \text{ mit } K_{i+1} \succ K_i \\ \widehat{T}_v(K) \cdot (1 - \sum_{i=1}^{r} \widehat{P}_i(K)) & \end{cases}$$

7.4 Ein Beispiel

Im folgenden möchten wir das eingeführte Kostenmodell in einem Beispiel einsetzen, um verschiedene Varianten einer Wiederverwendbarkeitsshierarchie zu vergleichen. Hierzu führen wir das Beispiel aus Abschnitt 7.2.7 fort. Wir gehen von den in Tabelle 7.1 dargestellten Fällen $F_1, \ldots, F_5$ der konkreten Problemlösedomäne sowie den daraus erzeugten generalisierten abstrakten Fällen $GAF_1, \ldots, GAF_5$ aus, so daß sich die in Abbildung 7.8(f) gezeigte Wiederverwendbarkeitshierarchie ergibt. Abbildung 7.12 zeigt nochmals diese Wiederverwendbarkeitshierarchie. Um das Kostenmodell auf dieses Beispiel anzuwenden, wollen wir annehmen, daß die Zeitaufwände $t_{\triangleright}(GAF, p)$ und $t_v(GAF, p)$ durch entsprechende Zeitmessungen bestimmt wurden. Die Tabellen die neben den Knoten in Abbildung 7.12 dargestellt sind, zeigen die Werte, von denen wir im folgenden ausgehen wollen. Zusätzlich sind in den Tabellen bereits die berechneten Schätzungen $T_{EW}(K)$ eingetragen. Wir möchten nun den Erwartungswert für verschiedene Retrievalvarianten bestimmen und miteinander vergleichen. Diese Varianten unterscheiden sich darin, ob Knoten der Hierarchie ausgeblendet und wie die Knoten K_2 und K_4 geordnet sind.

Variante 1: Alle Knoten sind selektiert und es gilt $K_2 \succ K_4$

Es ergeben sich die folgenden Schätzungen für die Erwartungswerte für die Knoten in der Hierarchie:

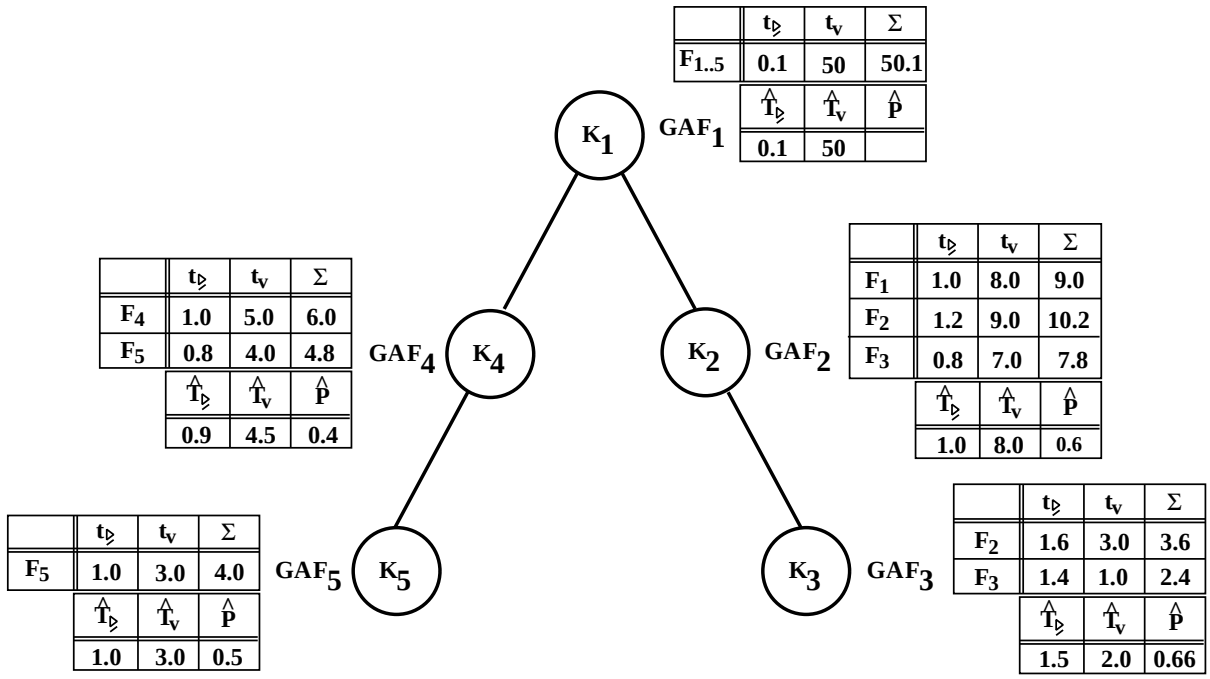

Abbildung 7.12: Wiederverwendbarkeitshierarchie für das Beispiel aus Tabelle 7.1 zusammen mit ermittelten Zeiten für den Wiederverwendbarkeitstest und die Lösungsverfeinerung sowie den daraus berechneten Schätzungen für die Erwartungswerte.

$$\widehat{T_{EW}}(K_5) = 3.0$$
$$\widehat{T_{EW}}(K_3) = 2.0$$
$$\widehat{T_{EW}}(K_4) = 3.0 \cdot 0.5 \ + \ 1 \cdot (1-0) \ + \ 4.5 \cdot (1-0.5) = 4.75$$
$$\widehat{T_{EW}}(K_2) = 2.0 \cdot 0.66 \ + \ 1.5 \cdot (1-0) \ + \ 8.0 \cdot (1-0.66) = 5.46$$
$$\widehat{T_{EW}}(K_1) = (4.75 \cdot 0.4 + 5.46 \cdot 0.6) \ + \ (0.9 \cdot (1-0) + 1 \cdot (1-0.4)) + 50 \cdot (1-1) = 6.676$$

Der Erwartungswert für die Problemlösezeit mit dieser Retrievalvariante beträgt also $\widehat{T_{EW}}(K_1) = 6.676$.

Variante 2: Der Knoten K_5 ist ausgeblendet und es gilt $K_2 \succ K_4$

Wird nun der Knoten K_5 ausgeblendent, so besitzt K_4 keine selektierten Nachfolger mehr und die Schätzungen für die Erwartungswerte ändern sich wie folgt:

$$\widehat{T_{EW}}(K_4) = 4.5$$
$$\widehat{T_{EW}}(K_1) = (4.5 \cdot 0.4 + 5.46 \cdot 0.6) \ + \ (0.9 \cdot (1-0) + 1 \cdot (1-0.4)) + 50 \cdot (1-1) = 6.576$$

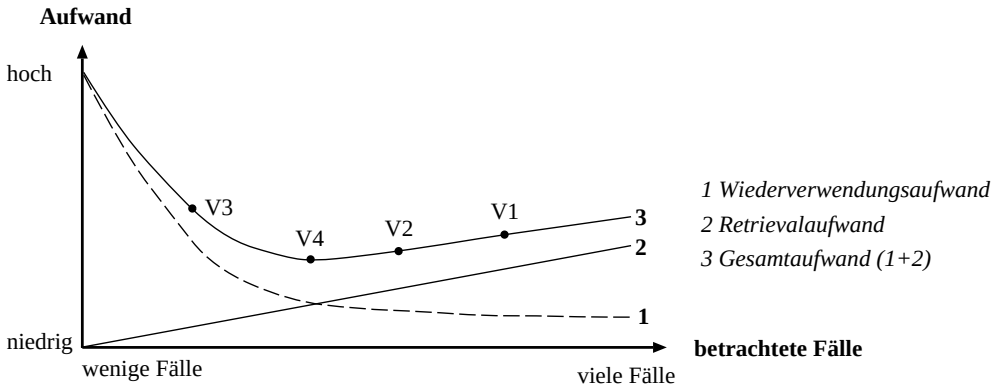

Abbildung 7.13: Interpretation der Beispielvarianten 1 bis 4

Wir können also feststellen, daß die Schätzung für den Erwartungswert zu einem besseren Ergebnis führt. Hier ist es also insgesamt günstiger, den Knoten K_5 nicht zu betrachten. Dies wird auch sofort einsichtig, wenn man die in Abbildung 7.12 gezeigten Zeitaufwände für die Knoten K_5 und K_4 betrachtet. Die Wiederverwendung von GAF_5 ist für F_5 zwar insgesamt günstiger ($t_{\triangleright} + t_v = 4.0$) als die Wiederverwendung von GAF_4 ($t_{\triangleright} + t_v = 4.8$), jedoch rechtfertigt dieser Vorteil nicht den erhöhten Aufwand für den erforderlichen Test der Wiederverwendbarkeit von GAF_5 ($t_{\triangleright} = 1.0$). Daher ist es günstiger, das Retrieval an dieser Stelle vorzeitig abzubrechen. Interpretiert man dieses Ergebnis anhand von Abbildung 7.13, so erkennt man, daß bei Variante 1 Fälle auswählt werden, die „rechts“ des optimalen Falls liegen.

Variante 3: Der Knoten K_3 ist ausgeblendet und es gilt $K_2 \succ K_4$

Wenn wir nun die Retrievalvariante betrachten, bei der nur der Knoten K_3 ausgeblendet ist, so ergeben sich die folgenden Änderungen der Schätzungen der Erwartungswerte im Vergleich zur Variante 1.

$$\widehat{T_{EW}}(K_2) = 8.0$$
$$\widehat{T_{EW}}(K_1) = (4.75 \cdot 0.4 + 8.0 \cdot 0.6) \; + \; (0.9 \cdot (1-0) + 1 \cdot (1-0.4)) + 50 \cdot (1-1) = 8.180$$

Durch das Ausblenden von K_3 verschlechtert sich also das Retrievalverhalten. Der Knoten K_3 enthält einen nützlichen generalisierten abstrakten Fall, der zu einer erheblich kürzeren Wiederverwendungszeit führt als der generalisierte abstrakte Fall aus K_2. In diesem Fall lohnt sich also der Wiederverwendbarkeitstest für K_3. Interpretiert man dieses Ergebnis anhand von Abbildung 7.13, so erkennt man, daß Variante 3 Fälle auswählt, die „links“ des optimalen Falls liegen, d. h. das Retrieval bricht zu früh ab.

Variante 4: Der Knoten K_5 ist ausgeblendet und es gilt $K_4 \succ K_2$

Mit dieser Variante wollen wir noch den Einfluß der Knotenordnung demonstrieren. Im Unterschied zu Variante 2 wird hier der Knoten K_2 vor dem Knoten K_4 getestet. Die Schätzungen für die Erwartungswerte ändern sich im Vergleich zu Variante 2 wie folgt:

$$\widehat{T_{EW}}(K_1) = (4.5 \cdot 0.4 + 5.46 \cdot 0.6) \; + \; (1.0 \cdot (1-0) + 0.9 \cdot (1-0.6)) + 50 \cdot (1-1) = 6.516$$

Hierdurch kann also das Retrievalverhalten im Vergleich zur Variante 2 nochmals verbessert werden. In diesem Fall rührt die Verbesserung daher, daß der Knoten K_2 mit einer höheren Wahrscheinlichkeit wiederverwendbare Fälle enthält als K_4. Daher führt die hier gewähle Knotenordnung zu besseren Ergebnissen.

Alle anderen hier nicht aufgeführten Retrievalvarianten führen zu schlechteren Schätzungen, so daß Variante 4 hier die optimale Variante darstellt (siehe Abbildung 7.13).

Weitere Beispiele aus dem Bereich der Arbeitsplanung, die die Verwendung dieses Kostenmodells erläutern, sind in Kapitel 8 Abschnitte 8.3.3 und 8.3.7 zu finden.

7.5 Einsatz des Kostenmodells zur Retrievaloptimierung

Das im vorherigen Abschnitt gezeigte Beispiel hat bereits angedeutet, wie die Schätzung des Erwartungswertes zur Optimierung des Retrievals eingesetzt werden kann. Die Aufgabe besteht dabei im wesentlichen darin, eine Retrievalvariante zu bestimmen, die die beste Schätzung für den Erwartungswert für die Problemlösezeit liefert. Hierbei stellt sich die Frage, wie eine optimale Knotenordnung und eine optimale Wahl der Selektionsprädikate gefunden werden kann. Nachfolgend wird ein Ansatz für eine solche Optimierung skizziert; eine detaillierte Darstellung der entsprechenden Algorithmen kann in [Wilke, 1994] nachgelesen werden.

7.5.1 Optimierung der Knotenordnung

Die Knotenordnung hat einen Einfluß auf die Schätzung der Problemlösezeit. Hierbei ist zunächst zu berücksichtigen, daß die Knotenordnung die Wahrscheinlichkeiten für die Wahl eines Knotens verändert. Gemäß Bezeichnung 7.2 ist

$$\widehat{P}_i(K) = \frac{|\mathcal{F}_{K_i} - \bigcup_{j=1}^{i-1} \mathcal{F}_{K_j}|}{|\mathcal{F}_K|},$$

wobei die Knoten gemäß $K_{i+1} \succ K_i$ geordnet sind. Eine geänderte Knotenordnung kann jedoch nur dann zu geänderten Wahrscheinlichkeiten führen, wenn es zwei Knoten K_i und

K_j gibt, für die $\mathcal{F}_{K_j} \cap \mathcal{F}_{K_j} \neq \emptyset$ gilt. Falls alle Fallmengen der Knoten paarweise disjunkt sind, so ist die Schätzung der Wahrscheinlichkeiten jedoch unabhängig von der Knotenordnung. Die Knotenordnung wirkt sich jedoch selbst in diesem Fall auf die Schätzung des Erwartungswertes der Problemlösezeit aus. Wir betrachten hierzu die Formel:

$$\widehat{T_{EW}}(K) = \sum_{i=1}^{r} \widehat{T_{EW}}(K_i) \cdot \widehat{P}_i(K) + \underbrace{\sum_{i=1}^{r} \widehat{T_{\triangleright}}(K_i) \cdot (1 - \sum_{j=1}^{i-1} \widehat{P}_j(K))}_{\text{durch} \succ \text{ beeinflußt}} + \widehat{T}_v(K) \cdot (1 - \sum_{i=1}^{r} \widehat{P}_i(K)).$$

Sind die Wahrscheinlichkeiten $\widehat{P}_i(K)$ nicht durch die Knotenordnung beeinflußt, so wirkt sich die Knotenordnung $\succ$ lediglich auf den zweiten Summanden dieser Formel aus, so daß die gesamte Schätzung ein Minimum liefert, wenn dieser Summand minimal ist. Dieser Summand wird jedoch genau dann minimal wenn gilt:

$$K_{i+1} \succ K_i \text{ gdw } \frac{\widehat{T_{\triangleright}}(K_{i+1})}{\widehat{P_{i+1}}(K)} \geq \frac{\widehat{T_{\triangleright}}(K_i)}{\widehat{P}_i(K)}.$$

Hieraus folgt, daß wir eine Optimierung der Knotenordnung auf einfache Weise erreichen können, wenn wir vereinfacht annehmen, daß die Fallmengen $\mathcal{F}_K$ von je zwei Geschwisterknoten disjunkt sind.

7.5.2 Optimierung der Selektionsprädikate

Während sich die Optimierung der Knotenordnung unter bestimmten Voraussetzungen als besonders einfach gestaltet, erfordert die Optimierung der Selektionsprädikate einen Suchprozeß im Raum der möglichen Kombinationen von selektierten Nachfolgerknoten. Jedoch ist es möglich, die Optimierung für die Nachfolger eines Knotens unabhängig von der Optimierung der anderen Knoten durchzuführen, ohne daß dadurch die globale Optimalität verloren geht.

Optimierung durch vollständige Suche

Um das globale Minimum für die Schätzung des Erwartungswertes der Problemlösezeit zu finden, müssen alle möglichen Kombinationen von selektierten Nachfolgerknoten bestimmt und für jede Kombination die Schätzung $T_{EW}(K)$ ermittelt werden. Danach kann die Kombination bestimmt werden, bei der die Schätzung ein Minimum einnimmt. Ist r die Anzahl der Nachfolger eine Knotens, so besitzt diese Optimierung eine Zeitkomplexität von $O(2^r)$.

Optimierung durch Hillclimbing

Besitzt ein Knoten eine große Zahl von Nachfolgern, so ist der Aufwand für eine vollständige Suche nicht mehr tragbar. In diesem Fall kann durch einen Hillclimbing-Ansatz zumindest

ein lokales Minimum in akzeptabler Zeit gefunden werden. Ein solcher Hillclimbingansatz besitzt dabei nur eine Zeitkomplexität von $O(r^2)$ und ist damit problemlos realisierbar.

Teil III

Anwendung, Bewertung und Diskussion

Kapitel 8

Anwendungsbeispiel und empirische Untersuchungen

Die in Teil-II dieser Arbeit beschriebene PARIS-Architektur wurde vollständig prototypisch implementiert. Diese Implementierung wurde in der Programmiersprache PROLOG unter Verwendung der SWI-PROLOG Programmierumgebung [Wielemaker, 1992] erstellt. Benutzungsoberflächen wurden mit der objekt-orientierten Prologerweiterung PCE [Wielemaker & Anjewierden, 1992] erstellt. Details der Implementierung der wesentlichen Module des Systems sind in [Bergmann, 1990; Surmann, 1993; Wilke, 1993; Wilke, 1994] zu finden.

In diesem Kapitel wird die Anwendung des PARIS-Systems auf den Bereich der Arbeitsplanung für rotationssymmetrische Drehteile erörtert. Hierzu werden zunächst eine Reihe von Kriterien vorgestellt, die bei der Entwicklung einer Problemlösedomäne – insbesondere der abstrakten Problemlösedomäne – beachtet werden müssen. Danach wird eine geeignete Repräsentation der konkreten und abstrakten Problemlösedomäne für einen Ausschnitt aus dem Bereich der Arbeitsplanung für rotationssymmetrische Drehteile erarbeitet. Die Funktionsweise der einzelnen Komponenten des Systems wird sodann an einem durchgängigen Beispiel demonstriert.

Außerdem wird die Effizienz des Problemlösens durch die Wiederverwendung generalisierter abstrakter Fälle untersucht und mit einem generischen Problemlöser verglichen. Da das genaue Problemlöseverhalten von einer Vielzahl von Parametern abhängt, ist eine aussagekräftige theoretische Analyse kaum möglich. Deshalb führen wir eine Reihe von empirischen Untersuchungen durch, die verschiedene Aspekte des hier verfolgten Ansatzes systematisch beleuchten und somit eine Gesamtbewertung ermöglichen. Diese empirischen Untersuchungen ergänzen die bereits in früheren Arbeiten [Bergmann, 1993; Bergmann, 1995; Bergmann & Wilke, 1995a; Bergmann & Wilke, 1995c] publizierten experimentellen Befunde entscheidend.

8.1 Kriterien für die Entwicklung von Problemlösedomänen

Wie für jedes wissensbasierte System spielt auch für das PARIS-System die genaue Formulierung der Problemlösedomäne eine entscheidende Rolle. Hierbei kommt es jedoch weniger auf die Formulierung der konkreten Problemlösedomäne an, als vielmehr auf die verwendete abstrakte Problemlösedomäne und die generische Abstraktionstheorie. Dies liegt darin begründet, daß die Wiederverwendung nicht direkt auf der konkreten Problemlösedomäne stattfindet; Generalisierung, Fallauswahl und Spezialisierung beziehen sich ausschließlich auf die abstrakte Ebene. Lediglich für die Lösungsverfeinerung ist die Beziehung zwischen der abstrakten und der konkreten Problemlösedomäne von Bedeutung. Daher läßt sich die konkrete Problemlösedomäne mit Methoden des Knowledge Engineering entwickeln, wobei Kriterien wie Verständlichkeit, Einfachheit und Wartbarkeit im Vordergrund stehen sollten. Hierzu eignen sich beispielsweise modellbasierte Wissensakquisitionsmethoden. MIKADO [Schmidt, 1994] ist eine solche Methode, die ebenfalls Fälle als Wissensquelle in den Wissensakquisitionsprozeß mit einbezieht und sogar speziell für die Akquisition von Strips-Operatoren ausgelegt werden kann.

Die Formulierung der abstrakten Problemlösedomäne sowie der generischen Abstraktionstheorie bedarf jedoch einer genaueren Betrachtung. Sie hat einen bedeutenden Einfluß auf die Flexibilität und Effizienz der Wiederverwendung. Daher ist es wünschenswert, einen Satz von Kriterien zur Verfügung zu haben, die beschreiben, wie eine „gute“ Domänendefinition aussieht. Strenge Kriterien, die quantitative Aussagen über die zu erwartenden Ergebnisse erlauben sind hierbei jedoch kaum zu erzielen. Jedoch ist es möglich eine Menge von allgemeinen Faktoren zu bestimmen, die einen Einfluß auf das Problemlöseverhalten haben. Im folgenden werden diese Faktoren beschrieben, und ihr tendenzieller Einfluß auf das Problemlöseverhalten des resultierenden Systems wird erörtert.

8.1.1 Vollständigkeit der abstrakten Domänentheorie

Die konkrete Problemlösedomäne legt fest, welche Probleme und Lösungen überhaupt repräsentierbar sind. Damit alle repräsentierbaren konkreten Fälle abstrahiert werden können, muß die abstrakte Problemlösedomäne die notwendigen abstrakten essentiellen Sätze und Operatoren enthalten. Dies ist eine Art von Vollständigkeitsanforderung. Fehlen in der abstrakten Problemlösedomäne abstrakte Operatoren, um einen konkreten Plan vollständig zu abstrahieren, so schlägt der Fallabstraktionsalgorithmus fehl, d. h. es wird kein abstrakter Fall generiert. Falls dies geschieht, kann der vorliegende Fall nicht wiederverwendet werden.

Kriterium 8.1 Die abstrakte Domänentheorie sollte vollständig sein in Bezug auf die repräsentierbaren konkreten Fälle.

8.1.2 Unabhängigkeit der abstrakten Operatoren

Folgt man Korfs [1987] Analyse des hierarchischen Problemlösens (siehe auch Abschnitt 3.3.4), so führt die Verwendung von Abstraktion beim Problemlösen zur einer signifikanten Komplexitätsreduktion falls kein Backtracking zwischen der Lösung der einzelnen Teilprobleme stattfindet. Ist Backtracking erforderlich, so kann der Komplexitätsvorteil ganz oder teilweise durch das Backtracking aufgezehrt werden. Diese Analyse gilt in gleicher Weise auch für die Verfeinerung abstrakter Fälle. Daher sollte eine abstrakte Problemlösedomäne so formuliert sein, daß Backtracking bei der Lösungsverfeinerung soweit wie möglich ausgeschlossen werden kann. Auf Backtracking kann vollständig verzichtet werden, falls alle abstrakten Operatoren unabhängig voneinander verfeinert werden können (siehe Definition 5.11). Dieses Kriterium ist jedoch zu restriktiv. Es ist nicht zu erwarten, daß sich für praktische Probleme Domänentheorien entwickeln lassen, bei denen alle abstrakten Operatoren unabhängig verfeinerbar sind. Dennoch sollte man versuchen, abstrakte Operatoren zu definieren die *weitgehend unabhängig verfeinerbar* sind. Ein abstrakter Operator o^a ist weitgehend unabhängig verfeinerbar, wenn für „die meisten" Zustände s^c, $\tilde{s}^c \in \mathcal{S}_c$ gilt: falls $\alpha(s^c) \xrightarrow{o^a} \alpha(\tilde{s}^c)$ gilt, dann existiert eine Folge von konkreten Operatoren $\langle o_1^c, \ldots, o_k^c \rangle$, so daß $s^c \xrightarrow{o_1^c} \ldots \xrightarrow{o_k^c} \tilde{s}^c$ gilt.

Kriterium 8.2 Abstrakte Operatoren sollen weitgehend unabhängig verfeinerbar sein.

8.1.3 Teilzieldistanz abstrakter Operatoren

Die Teilzieldistanz (engl. subgoal distance) ist ein Parameter, der ebenfalls von Korf [1987] in der Analyse des hierarchischen Problemlösens verwendet wurde. Sie ist definiert als die maximale Länge einer konkreten Operatorsequenz, die zur Verfeinerung eines abstrakten Operators erforderlich ist (siehe Abbildung 8.1). Je größer die Teilzieldistanz ist, um so größer ist der zu durchsuchende Suchraum bei der Lösungsverfeinerung. Daher sollte eine gute abstrakte Problemlösedomäne möglichst abstrakte Operatoren mit einer geringen Teilzeildistanz enthalten.

Kriterium 8.3 Abstrakte Operatoren sollten eine geringe Teilzieldistanz aufweisen.

8.1.4 Generalität von Operatordefinitionen

Die Repräsentationssprache für Problemlösedomänen bietet viele Variationsmöglichkeiten bei der Definition eines abstrakten Operators. Ein Aspekt, der für die erklärungsbasierte Fallgeneralisierung eine große Rolle spielt, ist das Vorkommen von Variablen bei Operatordefinitionen. Meist kann ein Operator durch eine allgemeine Regel (oder Operatordefinition) unter Verwendung von Variablen definiert werden. Alternativ ist aber auch die Repräsentation durch mehrere spezielle Inferenzregeln möglich. Für das erklärungsbasierte

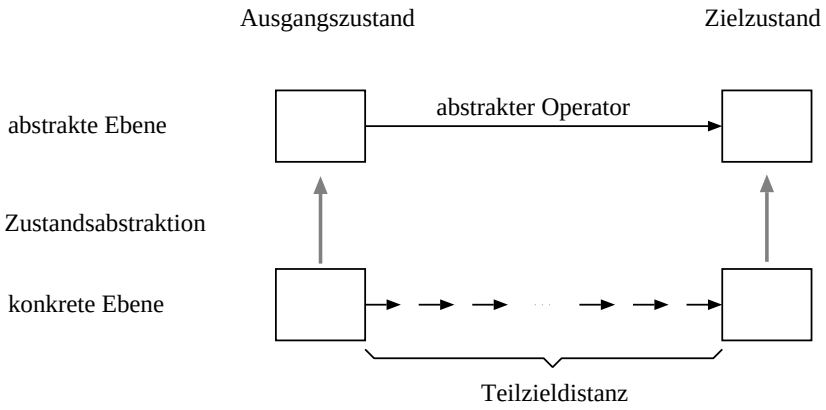

Abbildung 8.1: Teilzieldistanz eines abstrakten Operators

Generalisieren ist jedoch die allgemeinere Repräsentation von Vorteil, da entstehende Generalisierungen immer spezieller sind als die Regeln der Domänentheorie (siehe Abschnitt 3.2.6). Übertragen auf die Fallgeneralisierung bedeutet dies, daß die Allgemeinheit der generalisierten Fälle, d. h. die Größe der vom generalisierten Fall abgedeckten Problemklasse (siehe Definition 6.2), durch die Allgemeinheit der Operatordefinitionen entscheidend mitbestimmt wird. Sie hat somit einen direkten Einfluß auf die Flexibilität der Wiederverwendbarkeit.

Kriterium 8.4 Um eine hohe Flexibilität zu erzielen, sollten abstrakte Operatoren durch möglichst allgemeine Operatordefinitionen und Inferenzregeln definiert werden.

8.1.5 Komplexität der generischen Abstraktionstheorie

Ein weiterer wichtiger Faktor, der die Effizienz der Wiederverwendung bestimmt, ist die „Komplexität“ der generischen Abstraktionstheorie. Die Komplexität bezeichnet hierbei Aufwand, der erforderlich ist, um Inferenzen aufgrund dieser Theorie ziehen zu können. Die Abstraktionstheorie muß während der Lösungsverfeinerung auf jeden, bei der Suche auf der konkreten Ebene, entstehenden Zustand angewendet werden. Je komplexer die generische Abstraktionstheorie ist, desto aufwendiger ist die Berechnung der Zustandsabstraktionen (siehe auch Abschnitt 5.4.5). Daher sollte die generische Abstraktionetheorie möglichst auf aufwendige Inferenzen verzichten und z. B. Backtracking weitgehend vermeiden.

Kriterium 8.5 Die Komplexität der generischen Abstraktionstheorie sollte so gering wie möglich sein.

8.2 Eine Repräsentation für die Arbeitsplanung von Werkstücken

Im folgenden wollen wir eine Repräsentation eines Ausschnitts der Domäne der Arbeitsplanung von rotationssymmetrischen Werkstücken vorstellen. Hierbei werden die in den Abschnitten 2.1.2 und 8.3.1 eingeführten Abstraktionsebenen weiter detailliert. Bei dieser Repräsentation werden die zuvor eingeführten Kriterien berücksichtigt. Eine vollständige Beschreibung dieser Problemlösedomäne kann dem Anhang des Artikels [Bergmann & Wilke, 1995a] entnommen werden.

8.2.1 Die konkrete Problemlösedomäne

Die konkrete Problemlösedomäne repräsentiert alle notwendigen Details der Problemstellung und Lösung, die zur Fertigung eines Werkstückes auf einer Drehbank erforderlich sind.

Die Zustandsbeschreibung

Ein Zustand auf der konkreten Ebene gibt die exakte Geometrie des Werkstückes einschließlich aller Abmessungen wieder. Hierzu wird das gesamte Werkstück in verschiedene *atomare Bearbeitungsbereiche* zerlegt. Jeder dieser atomaren Bearbeitungsbereiche wird bei der Fertigung des Werkstückes als Einheit behandelt; der Bereich wird entweder vollständig gefertigt oder aber gar nicht. Betrachten wir eine zweidimensionale Schnittzeichnung eines Werkstückes (siehe Abbildung 8.2), so liegen diese Bearbeitungsbereiche wie ein Gitter über

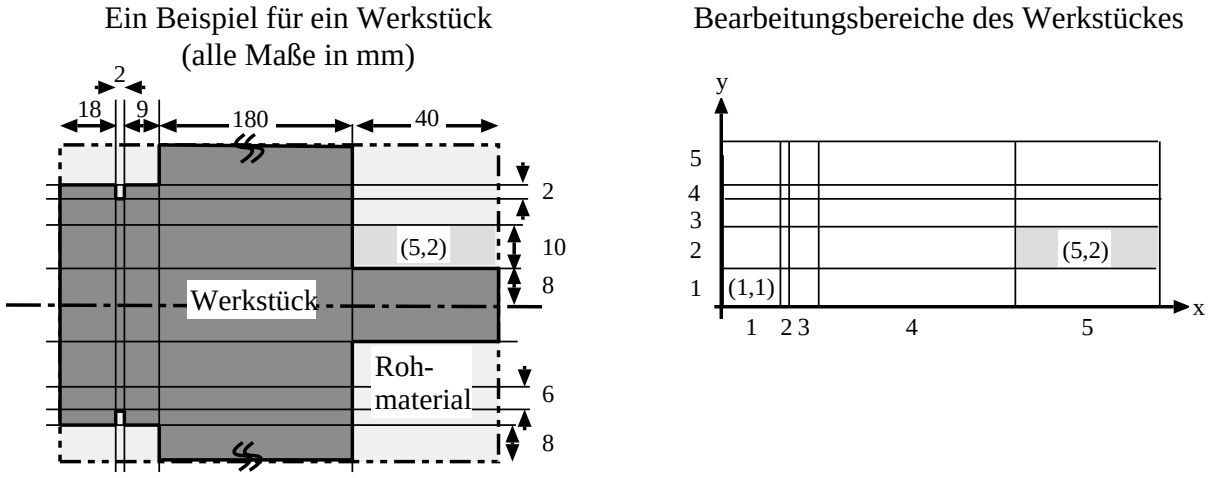

Abbildung 8.2: Beispiel für die Zerlegung eines Werkstückes in Bearbeitungsbereiche

dem Werkstück. Jede Konturveränderung (Änderung des Durchmessers des Werkstücks) führt zu der Einführung einer Gitterline in Richtung der x- und y-Koordinate. Jeder einzelne Bearbeitungsbereich kann nun durch die Angabe der Gitterkoordinate bezeichnet werden (z. B. $(5, 2)$ in Abbildung 8.2). Hierdurch ist der betreffenden Bereich des Werkstückes exakt spezifiziert. Zur Repräsentation der Bearbeitungsbereiche werden essentielle Sätze der folgenden Form in die Zustandsbeschreibung aufgenommen:

Prädikat	Beschreibung
grid_xpos *grid_ypos*	Die Prädikate *grid_xpos*($x_{grid}, x_{start}, x_{size}$) und *grid_ypos*($y_{grid}, y_{start}, y_{size}$) definieren die geometrische Position und Ausdehnung des Gitters in x- bzw. y-Richtung. Das erste Argument gibt die Gitterkoordinate an, das zweite definiert den Startpunkt des Bereiches und das dritte spezifiziert dessen Ausdehnung. Das Gitter für das Beispiel aus Abbildung 8.2 kann durch die folgenden essentiellen Sätze beschrieben werden: *grid_xpos(1,0,18), grid_xpos(2,18,2), grid_xpos(3,20,9), grid_xpos(4,29,180), grid_xpos(5,209,40)* und *grid_ypos(1,0,8), ..., grid_ypos(5,26,8).*

Der aktuelle Bearbeitungszustand eines Werkstücks kann durch die Angabe von *Statusinformation* für jeden Bearbeitungsbereich beschrieben werden. Hierzu dienen essentielle Sätze der folgenden Form:

Prädikat	Beschreibung
mat	Essentielle Sätze der Form *mat*($x_{grid}, y_{grid}, status$) beschreiben den Status des Bearbeitungsbereiches mit den Koordinaten (x_{grid}, y_{grid}). Das Argument *status* kann durch eine der drei Konstanten *raw*, *workpiece* oder *none* instantiiert werden. Der Status *raw* besagt, daß das aktuelle Werkstück im entsprechenden Bearbeitungsbereich aus Material des Rohlings besteht, das noch durch geeignete Bearbeitungsschritte abgetragen werden muß. Der Status *workpiece* bezeichnet Bereiche, die im Zielwerkstück aus Material bestehen und folglich nicht abgetragen werden dürfen. Der Status *none* besagt, daß der Bearbeitungsbereich kein Material enthält.

Die essentiellen Sätze der Form *grid_xpos* und *grid_ypos* werden durch die Ausführung von Bearbeitungsoperatoren nicht verändert; jedoch ändert sich der Status von Bearbeitungsbereichen, falls durch eine Fertigungsoperation Material vom Werkstück abgetragen wird. In dem in Abbildung 8.2 gezeigten Beispiel ändert sich z. B. der Status des Bearbeitungsbereiches $(5, 2)$. Während im Initialzustand (zylindischer Rohling) *mat(5,2,raw)* gilt, ist im Zielzustand der essentielle Satz *mat(5,2,none)* gültig.

Neben der Beschreibung des Werkstückes muß der konkrete Zustand außerdem die Informationen über die Art der verwendeten Aufspannung sowie das gerade verwendete Werkzeug enthalten. Hierzu werden die folgenden essentiellen Satzformen verwendet.

Prädikat	Beschreibung
chuck_pos	Die essentielle Satzform *chuck_pos(side)* besagt, daß das Werkstück gerade auf einer bestimmten Seite (*right* oder *left*) eingespannt ist bzw. daß das Werkstück gerade nicht eingespannt ist (*none*).
cut_tool *cut_direction*	Die essentiellen Satzformen *cut_tool(id)* und *cut_direction(dir)* bezeichnen das gerade verwendete Schneidwerkzeug durch die Angabe einer Werkzeugidentifikation *id*. Das Argument *dir* bezeichnet dabei die Richtung, mit der sich das Schneidwerkzeug bewegt um Material abzutragen.

Die Operatoren

Die Fertigung eines Werkstückes wollen wir, wie bereits in Abschnitt 8.3.1 eingeführt, durch vier verschiedene Operatoren beschreiben: einen Operator zur Aufspannung eines Werkstückes, einen Operator zum Ausspannen des Werkstückes, einen Operator zur Werkzeugauswahl sowie einen Operator zur Fertigung eines Bearbeitungsbereiches. Die entsprechenden Operatoren und Parameter werden wie folgt bezeichnet:

Operator	Beschreibung
chuck	Der Operator *chuck(side)* beschreibt die Aufspannung auf der Seite des Werkstückes, die mit dem Argument *side* (Instantiierungen: *left* oder *right*) spezifiziert ist.
unchuck	Der Operator *unchuck* beschreibt das Ausspannen des Werkstückes.
use_tool	Mit *use_tool(dir, id)* wird das Werkzeug mit der Identifikation *id* für die folgenden Bearbeitungsschritte ausgewählt. Das Argument *dir* gibt hierbei die Richtung an, in der sich das Werkzeug zur Bearbeitung bewegt.
cut	Der Operator *cut*(x_{grid}, y_{grid}) besagt, daß der Bereich (x_{grid}, y_{grid}) bearbeitet, d. h. das entsprechende Rohmaterial entfernt wird. Dieser Operator besitzt eine Reihe von Vorbedingungen: das Werkstück muß aufgespannt sein, ein verwendbares Werkzeug muß bereits ausgewählt sein und der Bearbeitungsbereich muß für das Werkzeug zugänglich sein.

Mit diesen Operatoren läßt sich nun ein Plan zur Fertigung des in Abbildung 8.2 dargestellten Werkstückes erstellen (siehe Abbildung 8.3). Hierbei wird das Werkstück zunächst auf der linken Seite eingespannt (Schritt 1), ein Werkzeug wird ausgewählt, das es erlaubt, das Werkstück von rechts nach links zu bearbeiten (Schritt 2), und es werden vier horizontale Schnitte durchgeführt (Schritte 3 bis 6). Danach wird das Werkstück auf der rechten Seite eingespannt (Schritte 7 und 8), und mit einem anderen Werkzeug (Schritt 9) lassen sich

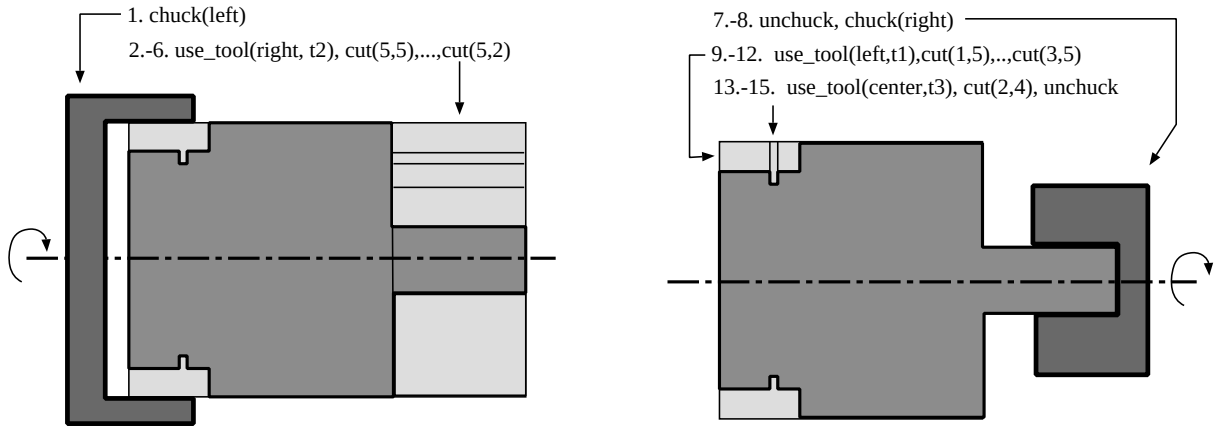

Abbildung 8.3: Ein Plan zur Fertigung des Werkstückes aus Abbildung 8.2

nun drei weitere Bearbeitungsbereiche bearbeiten (Schritte 10 bis 12). Einige dieser Bereiche ließen sich vorher nicht bearbeiten, da sie durch die vorherige Aufspannung abgedeckt waren. Abschließend wird mit einem speziellen Werkzeug (Schritt 13) die Nut gefertigt (Schritt 14). Schließlich wird das Werkstück ausgespannt (Schritt 15),

8.2.2 Die abstrakte Problemlösedomäne

Während wir in der konkreten Problemlösedomäne die geometrischen Abmessungen der Kontur des Werkstückes dargestellt haben, soll die abstrakte Problemlösedomäne hiervon in geeingeter Weise abstrahieren. Hierzu werden wir im folgenden lediglich zwischen der linken, der mittleren und der rechten Seite des Werkstückes unterscheiden. Hierdurch werden die atomaren Bearbeitungsbereiche zu drei komplexen Bearbeitungsbereichen zusammengefaßt. Außerdem wollen wir von der genauen Form der Werkstückoberfläche abstrahieren und lediglich zwischen großflächigen Bearbeitungen und kleineren Konturelementen (z. B. Nuten) unterscheiden.

Die Zustandsbeschreibung

Zustände beschreiben wir in der abstrakten Problemlösedomäne durch die folgenden essentiellen Sätze:

Prädikat	Beschreibung
abs_area_state	Ein essentieller Satz der Form *abs_area_state(area, status)* beschreibt den Zustand des komplexen Bearbeitungsbereiches *area* (*left*, *middle* oder *right*). Das Argument *status* kann dabei drei verschiedene Zustände anzeigen: a) die Bearbeitung dieses Bereiches erfordert weiterhin großflächige Bearbeitungen (*todo*), b) die Bearbeitung des Bereiches erfordert lediglich die Bearbeitung von kleineren Konturelementen (*rough*), c) der Bereich erfordert keine (weitere) Bearbeitung (*ready*).
abs_small_parts	Ein essentieller Satz der Form *abs_small_parts(area)* besagt, daß im komplexen Bearbeitungsbereich *area* kleine Konturelemente vorkommen.
abs_chuck_pos	Ein essentieller Satz der Form *abs_chuck_pos(side)* gibt die Seite (*side*) an, auf der das Werkstück gerade eingespannt ist.
abs_chuckable_wp	Ein essentieller Satz der Form *abs_chuckable_wp(side)* spezifiziert, ob das Werkstück auch dann noch auf der Seite *side* aufgespannt werden kann, wenn diese Seite bereits vollständig bearbeitet ist.

Die Operatoren

Die Operatoren der abstrakten Problemlösedomäne manipulieren die komplexen Bearbeitungsbereiche. Sie fassen somit die Bearbeitung mehrerer atomarer Bearbeitungsbereiche zusammen. Wir verwenden hierzu die folgenden Operatoren:

Operator	Beschreibung
set_fixation	Der Operator *set_fixation(side)* bezeichnet die Aufspannung des Werkstücks auf der Seite *side*. Im Vergleich zum Operator *chuck* der konkreten Problemlösedomäne ignoriert dieser Operator die Vorbedingungen für die Aufspannung, die sich auf die exakte Geometrie des Werkstückes an der Aufspannfläche beziehen.
process_rough	Der Operator *process_rough(area)* beschreibt die Grobbearbeitung des komplexen Bearbeitungsbereiches *area*. Hierdurch werden alle großflächigen Konturelemente dieses Bereiches gefertigt.

Operator	Beschreibung
process_fine	Der Operator *process_fine(area)* beschreibt die Feinbearbeitung des komplexen Bearbeitungsbereiches *area*. Dieser Operator verlangt in seiner Vorbedingung, daß alle großflächigen Konturelemente des Bereiches bereits vollständig gefertig sind.
process_ready	Der Operator *process_ready(area)* beschreibt die vollständige Bearbeitung eines Bearbeitungsbereiches *area* unabhängig von seinem vorherigen Zustand. Dieser Operator ist somit abstrakter als die beiden Operatoren *process_rough* und *process_fine*.

8.2.3 Beurteilung der Kriterien

Bei dieser Repräsentation der Domäne wurden die zuvor eingeführten Kriterien, die eine „gute“ Repräsentation der abstrakten Problemlösedomäne ermöglichen sollen, berücksichtigt. Im Detail wurden die Kriterien folgendermaßen beachtet:

bf Vollständigkeit (Kriterium 8.1): Die entwickelte abstrakte Problemlösedomäne ist vollständig, d. h. die verfügbaren abstrakten Operatoren sind ausreichend, um jeden (korrekten) Fall, der auf der konkreten Ebene formulierbar ist, zu abstrahieren.

Weitgehende Unabhängigkeit (Kriterium 8.2): Die abstrakten Operatoren sind nicht vollständig unabhängig verfeinerbar. Alle abstrakten Operatoren enthalten lediglich Approximationen der Bedingungen, die erfüllt sein müssen, um die Verfeinerbarkeit zu garantieren. Jedoch wurden diese Approximationen so gewählt, daß die Operatoren als weitgehend unabhängig verfeinerbar angesehen werden können. Eine exakte Formulierung der Operatoren würde eine generische Abstraktionstheorie erfordern, die hochgradig komplex wäre und somit das Kriterium 8.5 verletzen würde.

Geringe Teilzieldistanz (Kriterium 8.3): Die Teilzieldistanz der abstrakten Operatoren ist stark von dem vorliegenden Problem abhängig. Während die Teilzieldistanz beim Operator *set_fixation* immer zwischen eins und zwei liegt, wird die Teilzieldistanz der Bearbeitungsoperatoren durch die Anzahl der atomaren Bearbeitungsbereiche bestimmt, die zu einem komplexen Bearbeitungsbereich zusammengefaßt werden. Sei c die Anzahl dieser atomaren Bearbeitungsbereiche, so kann die maximal vorkommende Teilzieldistanz des Operators *process_ready* mit $2 \cdot c$ abgeschätzt werden. Die Teilzieldistanzen der Operatoren *process_rough* und *process_fine* sind geringer als die des Operators *process_ready* (die Teilzieldistanz des Operators *process_ready* ist etwa so groß wie die Summe der Teilzieldistanzen der Operatoren *process_rough* und *process_fine*), d. h. abstrakte Fälle, die den Operator *process_ready* enthalten erfordern einen höheren Verfeinerungsaufwand als abstrakte Fälle, die die Operatoren *process_rough* und *process_fine* enthalten.

Hohe Generalität (Kriterium 8.4): Die abstrakten Operatoren wurden so allgemein wie möglich definiert. Die Operatordefinitionen und Inferenzregeln enhalten soweit wie

möglich Variablen und verknüpfen möglichst allgemeine Bedingungen.

Geringe Komplexität (Kriterium 8.5): Die generische Abstraktionstheorie definiert die abstrakten essentiellen Sätze durch essentielle Sätze der konkreten Problemlösedomäne. Die abstrakten essentiellen Sätze wurden dabei so gewählt, daß sie sich ohne aufwendige Inferenzen aus den konkreten Zuständen ableiten lassen. Außerdem wurde durch eine effiziente Formulierung der Hornklauseln (z. B. durch die Verwendung des „Cut“[1]) Backtracking innerhalb dieses Inferenzprozesses weitgehend verhindert.

8.3 Ein Beispiel

Im folgenden wollen wir die Funktionsweise der einzelnen Komponenten an einem Beispiel demonstrieren. Wir setzten hierzu die zuvor beschriebene Repräsentation der Problemlösedomäne voraus. Wir verfolgen nun beispielhaft die Wiederverwendung des in Abbildung 8.3 gezeigten Falles.

8.3.1 Fallabstraktion

Zunächst betrachten wir den Prozeß der Fallabstraktion. Abbildung 8.4 zeigt vier abstrakte Fälle $AF_1, \ldots, AF_4$, die aus dem vorliegenden konkreten Fall F_1 gebildet werden. Zu jedem der abstrakten Fälle ist die entsprechende Zustandsabstraktionsabbildung angegeben. Die abstrakten Fälle unterscheiden sich in der Zusammenfassung von konkreten Operatoren zu abstrakten Operatoren. Hierbei liegen die abstrakten Fälle AF_1 und AF_2 auf einer höheren Abstraktionebene als die Fälle AF_3 und AF_4. Der wesentliche Unterschied zwischen den beiden Abstraktionsebenen besteht darin, daß auf der Abstraktionsebene auf der die Fälle AF_3 und AF_4 liegen, zwischen der Grob- und der Feinbearbeitung der linken Seite des Werkstückes unterschieden wird, während die Fälle AF_1 und AF_2 nicht zwischen diesen differenzieren und hierfür den abstrakteren Operator *process_ready* verwenden. Der Unterschied zwischen den abstrakten Fälle AF_1 und AF_2 bzw. AF_3 und AF_4 besteht lediglich darin, daß das Umspannen des Werkstückes bei den abstrakten Fällen AF_1 und AF_3 durch einen einzigen abstrakten Operator realisiert wird, während dies in den abstrakten Fällen AF_2 und AF_4 in zwei abstrakte Operatoren zerlegt wird.

8.3.2 Fallgeneralisierung

Im Anschluß an die Fallabstraktion wird die Fallgeneralisierung für jeden der in Abbildung 8.4 gezeigten abstrakten Fälle durchgeführt. Hieraus ergeben sich 4 generaliserte abstrakte Fälle. Abbildung 8.5 zeigt beispielhaft das Ergebnis der Generalisierung des abstrakten

[1] Ein Cut (!) ist ein in der Programmiersprache PROLOG vorkommendendes Sonderprädikat, mit dem Teile des Suchraums, der beim Inferenzprozeß durchsucht wird, gezielt abgeschnitten werden können.

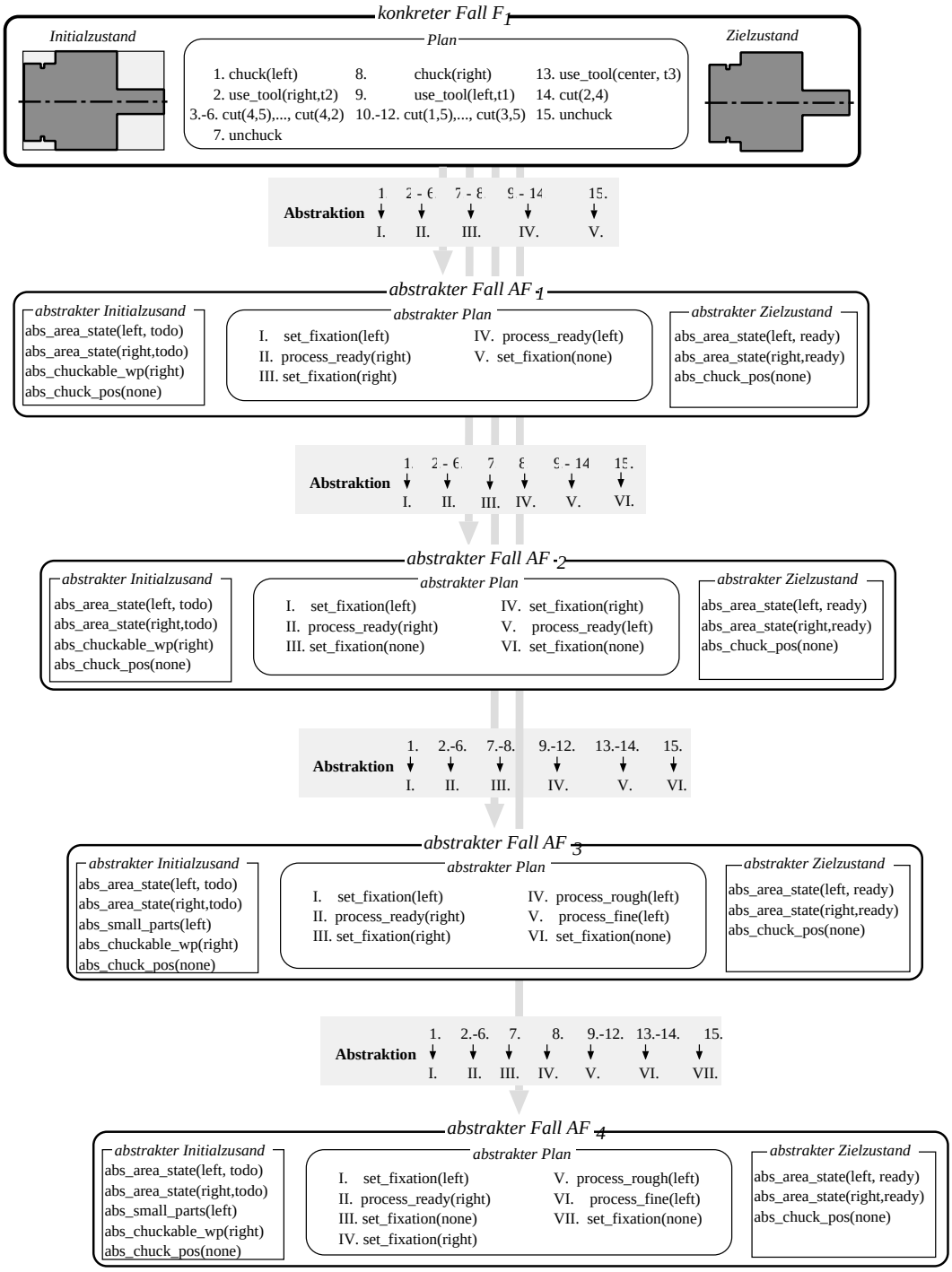

Abbildung 8.4: Ein Beispiel für die Fallabstraktion des konkreten Falls aus Abbildung 8.3

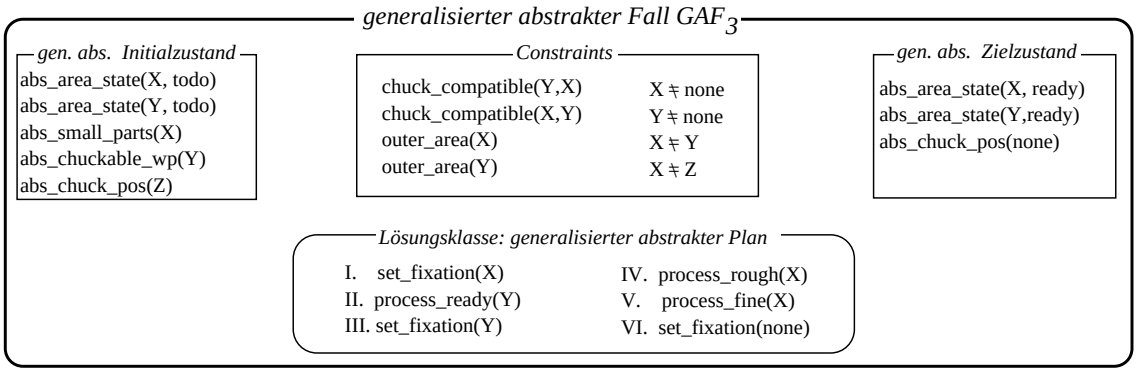

Abbildung 8.5: Ergebnis der Generalisierung des abstrakten Falles AF_3

Falls AF_3. Die Problemklasse des generalisierten abstrakten Falls besteht aus den gezeigten generalisierten Initial- und Zielzuständen sowie den Constraints. Hierbei sind Variablen durch Großbuchstaben dargestellt. In diesem Beispiel wird über die im Fall angegebenen Seiten (*left* und *right*) des Werkstückes generalisiert. Die angegebenen Constraints sind erforderlich, um zuzusichern, daß bei jeder Instantiierung der Variablen der entstehende instantiierte Plan zu einer korrekten Lösung führt. Die Prädikate, die in den Constraints vorkommen, stammen aus den Vorbedingungen der verwendeten abstrakten Operatoren. So sagt z. B. das Prädikat *outer_area(X)*, daß die Variable X einen äußeren komplexen Bearbeitungsbereich (*left* oder *right*, nicht jedoch *middle*) bezeichen muß. Dieses Prädikat wird in der Vorbedingung des Operators *set_fixation* verwendet und besagt, daß eine Aufspannung generell nur an den äußeren Seiten des Werkstücks vorgenommen werden kann. Die Beziehungen zwischen der Problemstellung der Lösung werden hier durch die Verwendung gemeinsamer Variablen realisiert. Die Instantiierung aller Variablen der Problemklasse liefert eine vollständig instantiierte Lösung.

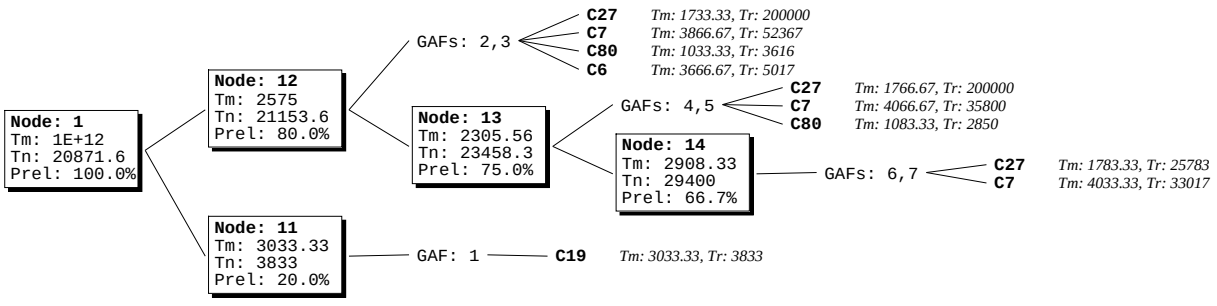

Abbildung 8.6: Eine Wiederverwendbarkeitshierarchie vor der Aufnahme der neuen Fälle

8.3.3 Erweiterung der Fallbasis

Der nächste Schritt besteht nun in der Aufnahme der vier generalisierten abstrakten Fälle in die Fallbasis und der Anpassung der Wiederverwendbarkeitshierarchie, damit auf diese Fälle zugegriffen werden kann. Zur besseren Verdeutlichung dieser Vorgehensweise wollen wir im folgenden davon ausgehen, daß die Fallbasis bereits einige Fälle enthält. Abbildung 8.6 zeigt eine mögliche Wiederverwendbarkeitshierarchie *vor* der Aufnahme der neuen generalisierten abstrakten Fälle. Hierbei bezeichen die Rechtecke die Knoten der Hierarchie. Zu jedem Knoten sind die ermittelten Schätzungen der charakteristischen Zeitaufwände und Wahrscheinlichkeiten angegeben[2]. Außerdem ist für jeden Knoten (außer dem Wurzelknoten) die Menge zugeordneter generalisierter abstrakter Fälle dargestellt. Diese zeigen wiederum Verweise auf die konkreten Fälle, aus denen dieses gebildet wurden. Beispielsweise besitzt der Knoten *Node 12* die beiden generalisierten abstrakten Fälle GAF_2 und GAF_3. Diese sind aus den konkreten Fällen $C27, C7, C80$ und $C6$ hervorgegangen. Für jeden Fall sind außerdem die ermittelten Zeiten zur Problemlösung durch Wiederverwendung eines generalisierten abstrakten Falls aus dem Knoten der Wiederverwendbarkeitshierarchie angegeben[3]. So erfordert z. B. der Test der Wiederverwendbarkeit eines generaliserten abstrakten Falls (z. B. GAF_2) aus Knoten *Node 12* für das Problem aus Fall $C7$ eine Zeit von 3866.67 ms und die Lösungsverfeinerung selbst erfordert nochmals 52367 ms.

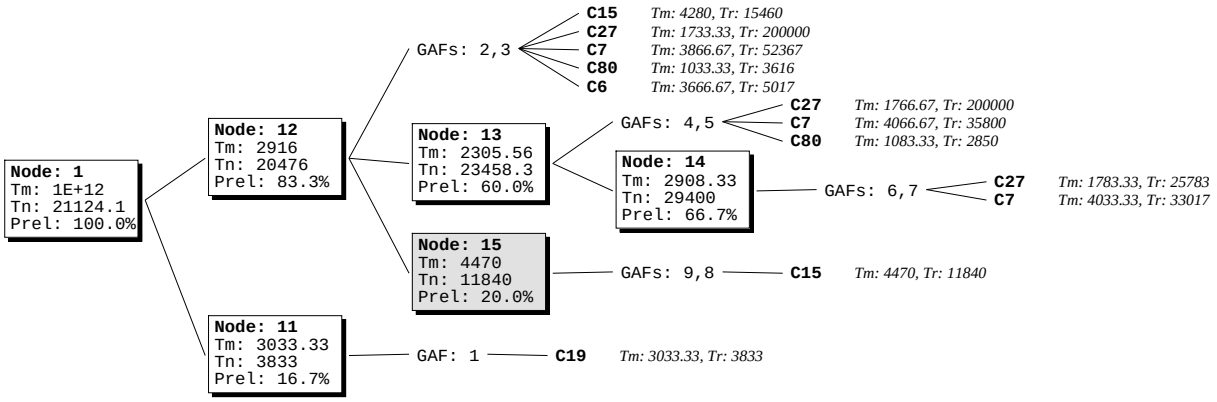

Abbildung 8.7: Wiederverwendbarkeitshierarchie nach der Aufnahme der neuen Fälle

Diese Wiederverwendbarkeitshierarchie soll nun um die zuvor gebildeten generalisierten abstrakten Fälle erweitert werden. Als Ergebnis entsteht die Wiederverwendbarkeitshierarchie die in Abbildung 8.7 dargestellt ist. In dieser Hierarchie ist der neue Fall mit $C15$

[2]Der Wert Tm bezeichnet die Schätzung des Zeitaufwandes zum Test der Wiederverwendbarkeit $\widehat{T_{\triangleright}}(K)$, Tn bezeichnet die Schätzung des Erwartungswertes der Problemlösezeit für den Knoten $\widehat{T_{EW}}(K)$ und $Prel$ bezeichnet die Schätzung der bedingten Wahrscheinlichkeit für das Betrachten des Knotens $\widehat{P}_i(K)$. Die Zeitangaben sind jeweils in Millisekunden

[3]Hierbei bezeichnet Tm die Zeit $t_{\triangleright}(GAF, p)$ für das Problem p aus dem Fall und Tr bezeichnet die Zeit $t_v(GAF, p)$.

bezeichnet. Von den vier gebildeten generalisierten abstrakten Fälle waren bereits zwei (die hier mit GAF_2 und GAF_3 bezeichnet sind) in der Fallbasis enthalten. Die beiden anderen werden als neue generalisierte abstrakte Fälle GAF_8 und GAF_9 in die Fallbasis aufgenommen. Hierzu wird die Hierarchie um den neuen Knoten (*Node 15*) erweitert. Jedoch führt die nachfolgende Retrievaloptimierung zum Ausblenden dieses neuen Knotens, was durch den grauen Knotenhintergrund angezeigt ist. Die Ursache hierfür liegt darin, daß der Optimierungsansatz aufgrund der bislang vorliegenden Fallbasis auf die Verteilung der in der Zukunft zu lösenden Probleme schließt. Im vorliegenden Fall läßt sich das Problem aus Fall $C15$ sowohl mit den generalisierten abstrakten Fällen aus bereits vorhandenen Knoten *Node 12* als auch mit den generalisierten abstrakten Fällen aus dem neuen Knoten *Node 15* lösen. Wie aus den angegebenen Problemlösezeiten zu ersehen ist, beträgt der Vorteil durch die Verwendung von Knoten *Node 15* lediglich 3620 ms (= 15460 ms - 11840 ms). Dieser Vorteil steht jedoch einem zusätzlichen Aufwand von 4470 ms zum Test der Wiederverwendbarkeit des Knotens *Node 15* gegenüber. Daher ist es bei dieser Problemverteilung generell günstiger, den neuen Knoten nicht zu betrachten.

Durch die Aufnahme weiterer Fälle kann sich jedoch die Annahme über die Verteilung der Probleme verändern und der zuvor ausgeblendete Knoten wieder aktiviert werden. Abbildung 8.8 zeigt ein solches Beispiel. Hierbei wird angenommen, daß ein weiterer konkreter Fall $C79$ vorliegt. Die Aufnahme der aus diesem Fall gebildeten generalisierten abstrakten Fälle führt zum einem zur Erweiterung der Hierarchie um den Knoten *Node 16* und zum anderen dazu, daß der Knoten *Node 15* wieder aktiviert wird. Der Grund für die Aktivierung dieses Knotens liegt darin, daß die enthaltenen generalisierten abstrakten Fälle ebenfalls zur Lösung des Problems aus Fälle $C79$ verwendet werden können. Der Vorteil der Verwendung dieses Knotens im Vergleich zum ebenfalls verwendbaren Knoten *Node 12* ist relativ groß (106960 ms = 195880 ms - 88920 ms). Dieser Vorteil überwiegt den Mehraufwand für den Test der Wiederverwendbarkeit des Knotens und führt folglich zur

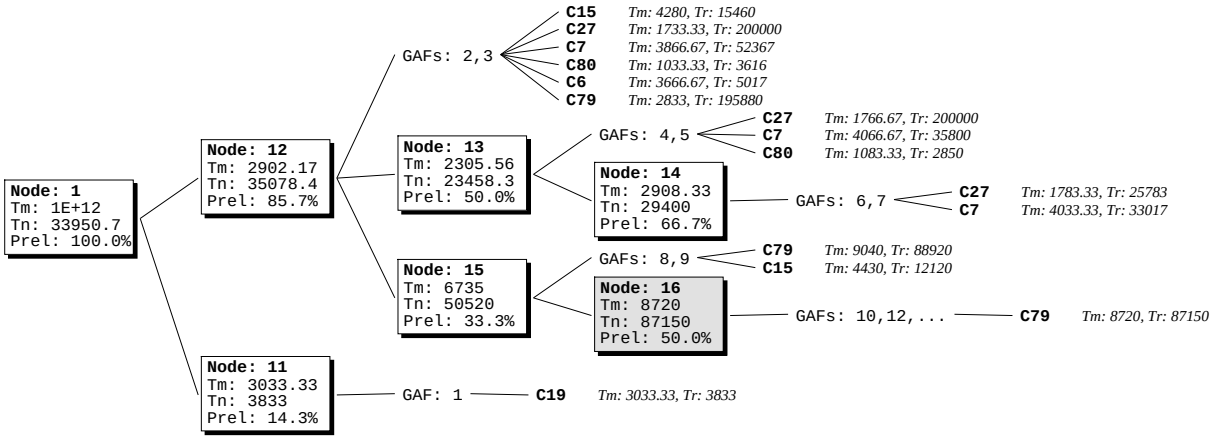

Abbildung 8.8: Wiederverwendbarkeitshierarchie nach der Aufnahme weiterer Fälle

Aktivierung dieses Knotens.

8.3.4 Retrieval eines Falles zur Lösung eines neuen Problems

Wir setzen nun dieses Beispiel mit der Betrachtung der Problemlösephase fort. Hierzu nehmen wir an, daß das neue Problem, das in Abbildung 8.9 dargestellt ist, gelöst werden soll. Betrachtet man dieses Problem zunächst auf der konkreten Problemlösedomäne, so ergibt sich ein erheblicher Unterschied zum Fall aus Abbildung 8.2. Abbildung 8.9 zeigt aber neben dem konkreten Problem auch die abstrahierte Problemstellung (abstrakter Initial- und Zielzustand). Diese verwenden wir im Retrieval und suchen nach einem generalisierten abstrakten Fall aus der Fallbasis, mit „passender" Problembeschreibung. Hierzu wird die Wiederverwendbarkeitshierarchie (siehe Abbildung 8.8) in Vorordnung durchlaufen bis ein Knoten erreicht wird, der keine wiederverwendbaren Nachfolger besitzt. In diesem Fall führt das genau zum Knoten *Node 15* und somit zur Wiederverwendung des in 8.5 gezeigten generalisierten abstrakten Falls.

8.3.5 Lösungsspezialisierung

Bereits während des Retrievals wird die schwache Wiederverwendbarkeit (Definition 7.1) von generaliserten abstrakten Fällen der Fallbasis für das vorliegende Problem getestet. Hierbei werden die Variablen, die im generalisierten abstrakten Fall vorkommenden, ge-

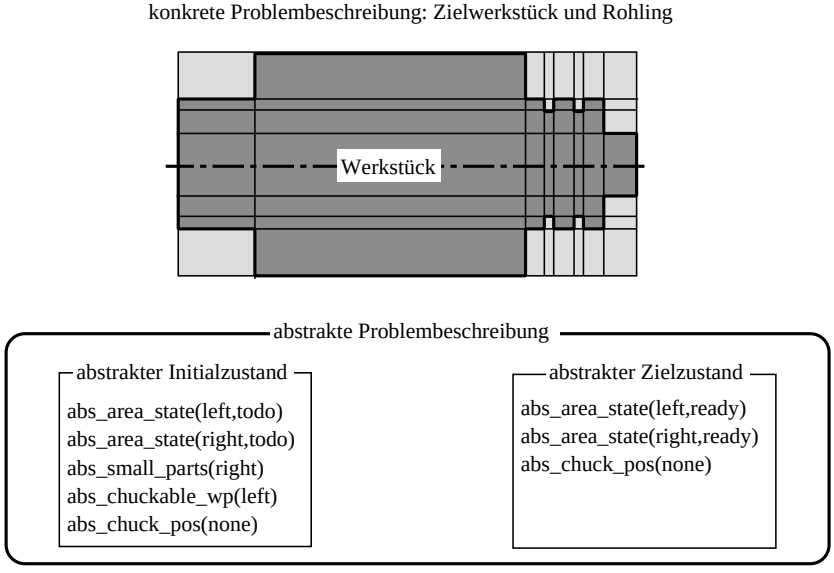

Abbildung 8.9: Neues Problem (Zielwerkstück und Rohling) auf konkreter und abstrakter Ebene

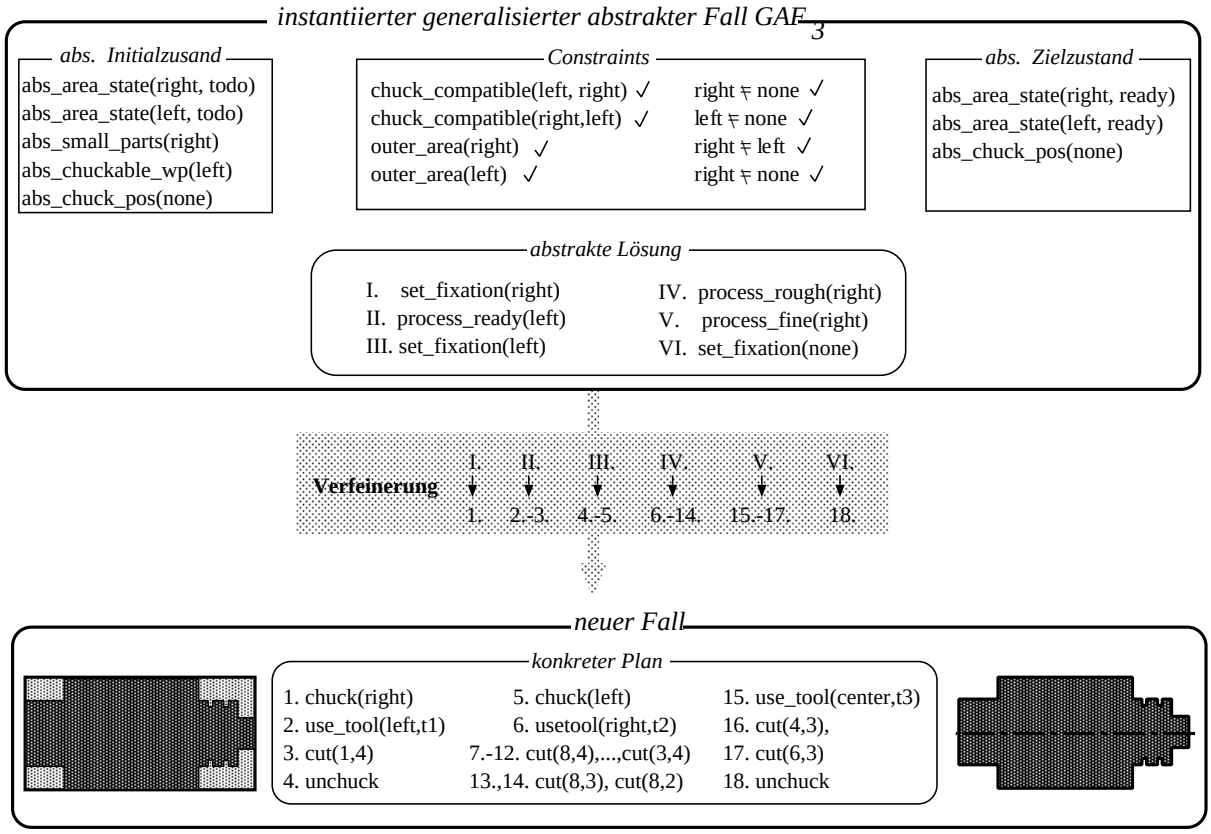

Abbildung 8.10: Instantiierter generalisierter abstrakter Plan aus Abbildung 8.5 und Lösungsverfeinerung

eignet instantiiert, und die Gültigkeit der Constraints wird getestet. Dies führt für den ausgewählten generalisierten abstrakten Fall (siehe Abbildung 8.5) zu der folgenden Instantiierung: $\{X/right, Y/left, Z/none\}$. Der obere Abschnitt von Abbildung 8.10 zeigt die entsprechende Instantiierung der Problemstellung und die sich daraus ergebende abstrakte Lösung. Man kann sich leicht davon überzeugen, daß diese Instantiierung alle Constraints erfüllt.

8.3.6 Lösungsverfeinerung

Im letzten Schritt muß nun die Verfeinerung der abstrakten Lösung zu einer konkreten Lösung des konkreten Problems vorgenommen werden. Der untere Teil von Abbildung 8.10 veranschaulicht diesen Prozeß. Der abstrakte Plan definiert eine Sequenz von 6 Teilproblemen, die auf der konkreten Problemlösedomäne durch Suche gelöst werden. Jedes dieser Teilprobleme führt zu einer Teillösung. Schließlich werden diese Teillösungen zur Gesamtlösung zusammengefaßt. Somit ist das neue Problem gelöst worden.

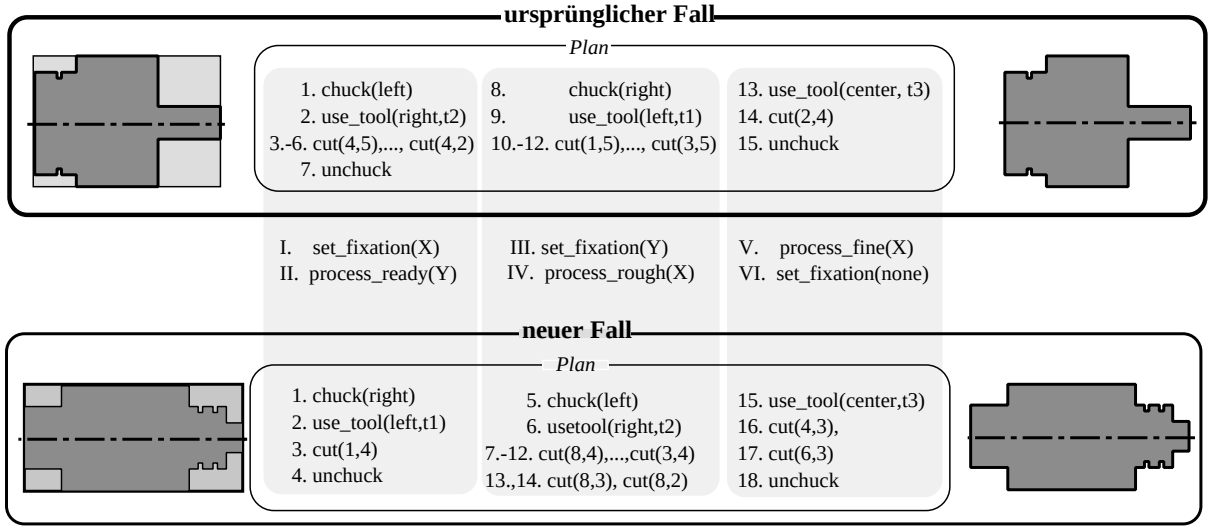

Abbildung 8.11: Vergleich von Ursprungsfall und neuer Lösung

An diesem Beispiel erkennt man, daß die Wiederverwendung generalisierter abstrakter Fälle zu einer Lösung führt, die von der Lösung im konkreten Ausgangsfall vollkommen verschieden ist. Abbildung 8.11 stellt diese beiden Fälle noch einmal einander gegenüber. Im Vergleich erkennt man die folgenden Veränderungen in der Lösung:

- Die rechte und linke Seite des Werkstückes ist vertauscht. Die Operationen die sich im Ursprungsfall auf die rechte Seite bezogen haben, bearbeiten nun die linke Seite und umgekehrt.
- Die Bearbeitungsoperationen beziehen sich auf andere Bearbeitungsbereiche mit vollständig anderen Abmessungen.
- Die komplexen Bearbeitungsbereiche haben in beiden Fällen eine unterschiedliche Zusammensetzung. Während z. B. der rechte komplexe Bearbeitungsbereich im Ursprungsfall aus 4 atomaren Bearbeitungsbereichen besteht, enthält der korrespondierende linke komplexe Bearbeitungsbereich im neuen Fall lediglich einen atomaren Bearbeitungsbereich.

8.3.7 Mehrfach iteriertes Lernen und Problemlösen

Als Ergebnis der Lösungsverfeinerung ist also ein neuer Fall entstanden. Dieser stellt die gewünschte Lösung des vorliegenden Problems dar. Darüber hinaus ist dieser neue Fall wieder eine Eingabe in die Lernphase von PARIS. So können möglicherweise neue generalisierte abstrakte Pläne gelernt werden. Außerdem führt der neue Fall gegebenfalls zur veränderten

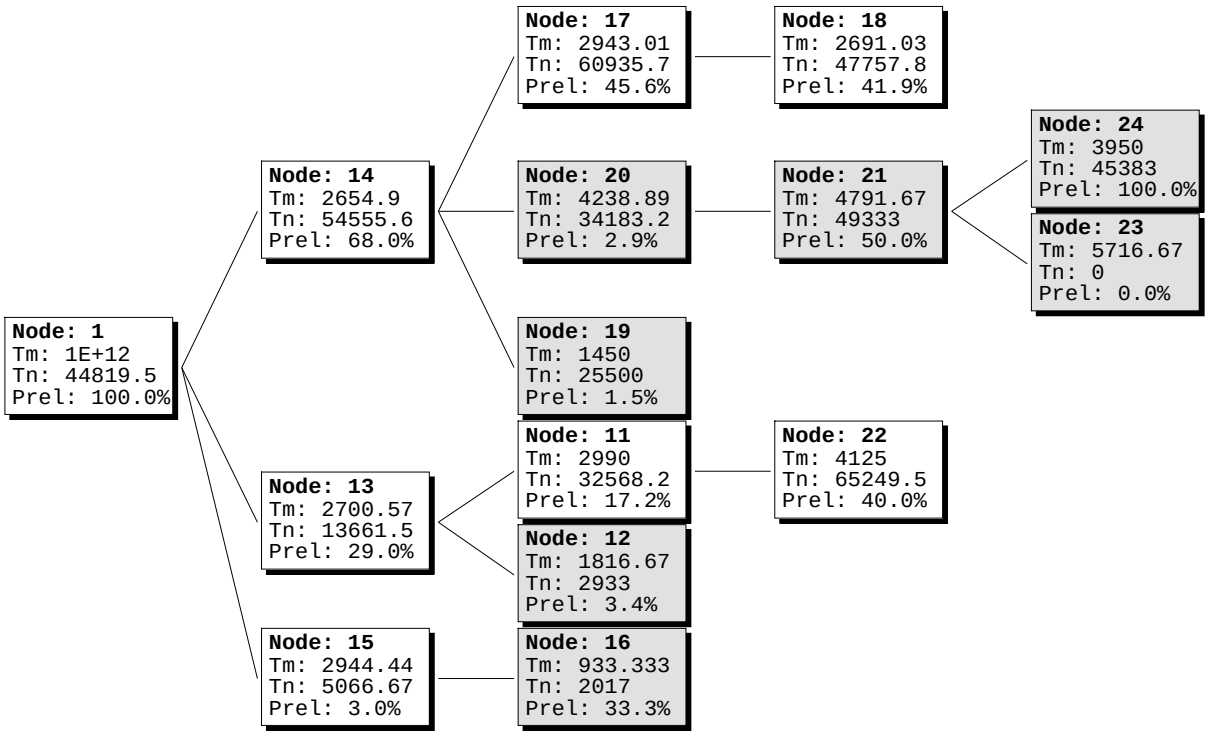

Abbildung 8.12: Ein Beispiel für eine Wiederverwendbarkeitshierarchie nach dem Lernen von 100 konkreten Fällen.

Abschätzung der Problemverteilung und der charakteristischen Zeitaufwände in der Wiederverwendbarkeitshierarchie. Hierdurch kann diese erweitert werden oder ausgeblendete Knoten können aktiviert bzw. aktivierte Knoten können ausgeblendet werden. Als Beispiel zeigt Abbildung 8.12 die Wiederverwendbarkeitshierarchie, die nach dem Lernen von 100 verschiedenen konkreten Fällen entstanden ist.

8.4 Experimentelle Untersuchungen

Während der vorherige Abschnitt den PARIS-Ansatz an einem Einzelfall demonstriert hat, soll dieser nun systematisch für eine größere Menge von Fallbeispielen evaluiert werden. Hierzu wird eine Reihe von Experimenten im strengen Sinne der Statistik entwickelt und durchgeführt. Ziel dieser experimentellen Untersuchung ist es, empirische Belege für eine Reihe von Hypothesen zu gewinnen. Hierbei geht es insbesondere darum, zu überprüfen, inwieweit die in Abschnitt 1.1.1 skizzierten Anforderungen an die Flexibilität und Effizienz der Wiederverwendung erfüllt werden können. Hierbei ist natürlich zu beachten, daß eine solche empirische Untersuchung immer nur Aussagen über einen sehr eng begrenz-

ten Anwendungsbereich des Systems geben können. Dies liegt vorallem daran, daß das Problemlöseverhalten des Systems von einer Reihe von Parametern abhängt. Diese sind:

- die Problemlösedomäne,
- die *Repräsentation* der konkreten und abstrakten Problemlösedomäne und der generischen Abstraktionstheorie,
- die Auswahl und Verteilung der vorliegenden Fälle
- die Verteilung der zu lösenden Probleme.

Naturgemäß läßt eine experimentelle Untersuchung nur eine begrenzte Variationen dieser Parameter zu. Dies muß bei der Interpretation der Ergebnisse beachtet werden.

8.4.1 Hypothesen

Zunächst sollen die zu untersuchenden Hypothesen dargelegt werden.

Effizienzsteigerung

Die grundsätzliche Motivation für den PARIS-Ansatz ist ein erwarteter Effizienzvorteil durch Wiederverwendung im Vergleich zu rein generativem Problemlösen. Effizienzvorteil bedeutet hierbei, daß Probleme innerhalb kürzerer Rechenzeit gelöst werden können. Eng damit verbunden ist auch ein erwarteter Kompetenzvorteil, d. h. das Lösen einer größeren Menge von Problemen innerhalb einer vorgegebener Rechenzeitschranke. Daraus leitet sich auch die erste Hypothese für das PARIS-System ab.

Hypothese 8.1 Durch die Wiederverwendung generalisierter abstrakter Fälle ergibt sich ein Effizienzvorteil gegenüber generativem Problemlösen.

Abstraktion und Generalisierung

Der Wiederverwendungsansatz des PARIS-Systems basiert auf den Abstraktions- und Generalisierungstechniken sowie insbesondere auf deren Kombination. Abstraktion und Generalisierung führen auf verschiedene Weisen zu einer Flexibilisierung der Wiederverwendung. Daher erwarten wir, daß sich durch die Kombination von Abstraktion und Generalisierung bessere Wiederverwendungsergebnisse erzielen lassen als durch die Wiederverwendung rein generalisierter bzw. rein abstrakter Fälle. Hierbei erwarten wir a) eine größere Flexibilität durch die Kombination beider Verfahren und b) einen größeren Effizienzvorteil. Ein weiterer wesentlicher Schwerpunkt des Abstraktionsansatzes liegt auf dem Wechsel der Repräsentationssprache. Dies schätzen wir als sehr wesenlich für eine die Komplexitätsreduktion ein. Aus diesen Betrachtungen leiten wir die folgenden drei Hypothesen ab:

Hypothese 8.2 Durch die Wiederverwendung generalisierter abstrakter Fälle läßt sich eine höhere *Flexibilität* erreichen als durch die Wiederverwendug rein generalisierter oder rein abstrakter Fälle.

Hypothese 8.3 Durch die Wiederverwendung generalisierter abstrakter Fälle läßt sich eine höhere *Effizienz* erreichen als durch die Wiederverwendug rein generalisierter oder rein abstrakter Fälle.

Hypothese 8.4 Ohne den Wechsel der Repräsentationssprache ist eine Effizienzsteigerung nicht zu erreichen.

Retrieval mit Wiederverwendbarkeithierarchien

Ein wesentlicher Teil dieser Arbeit behandelt das Retrieval generalisierter abstrakter Fälle durch eine Wiederverwendbarkeitshierarchie. Diesem relativ aufwendigen Verfahren liegt die Annahme zugrunde, daß hierdurch eine erhöhte Effizient im Vergleich zu einem naiven linearen Retrieval erreicht werden kann. Eine weitere Verbesserung erwarten wir durch die Optimierung des Retrievals auf der Grundlage von Schätzungen der Problemverteilung und von charakteristischen Wiederverwendbarkeitsparametern. Hieraus leiten wir die folgenden Hypothesen ab:

Hypothese 8.5 Das Retrieval durch eine Wiederverwendbarkeitshierarchie führt zu einem Effizienzvorteil im Vergleich zu einem naiven linearen Retrieval.

Hypothese 8.6 Das Retrieval durch eine optimierte Wiederverwendbarkeitshierarchie führt zu einem Effizienzvorteil im Vergleich zu einem Retrieval mit einer nicht-optimierten Wiederverwendbarkeitshierarchie.

Einfluß der Problemlösedomäne

Die vorgestellten Wiederverwendungstechniken werden wesentlich durch die verwendete abstrakte Problemlösedomäne gesteuert. Die abstrakte Domänentheorie (und natürlich auch die entsprechende generische Abstraktionstheorie) spielen hierbei eine wichtige Rolle. Sie legt die Abstraktionsebenen fest, auf denen die Wiederverwendung stattfinden kann. Hierdurch wird bestimmt, welche Merkmale der Lösung relevant sind und beibehalten werden sollen und welche Merkmale ignoriert und ggf. neu generiert werden müssen. Somit ist zu erwarten, daß diese einen möglicherweise erheblichen Einfluß auf die Flexibilität und Effizienz der Wiederverwendung besitzen. Für die in Abschnitt 8.2 dargestellte Problemlösedomäne erwarten wir gute Wiederverwendungsresultate, da die zuvor skizzierten Kriterien weitgehend erfüllt werden konnten. Jedoch vermuten wir, daß dieser Effizienzvorteil bei Verwendung einer Problemlösedomäne, die diese Kriterien weniger gut erfüllt, verloren

gehen kann. Dies gilt insbesondere dann, wenn abstrakte Operatoren nicht mehr weitgehend unabhänig verfeinerbar sind (Kriterium 8.2). Diese Überlegung führt zur folgenden Hypothese.

Hypothese 8.7 Der Effizienzvorteil durch die Wiederverwendung hängt wesentlich von der Formulierung der verwendeten Problemlösedomäne ab. Eine Veränderung (insbesondere ein Verzicht auf die Gültigkeit von Kriterium 8.2) kann zu einem geringeren Effizienzvorteil führen.

8.4.2 Durchführung der Experimente

Die Experimente werden in dem eingeführten Bereich aus der Arbeitsplanung durchgeführt und bestehen aus zwei Phasen. In der ersten Phase wird das PARIS-System mit einer Menge von *Trainingsfällen* trainiert, d. h. es werden die entsprechenden generalisierten abstrakten Fälle gebildet und in der Fallbasis gespeichert. In der zweiten Phase wird das so entstandene System mit einer Reihe von *Testfällen* bewertet. Dabei wird jedes Problem aus den Testfällen durch Wiederverwendung von Fällen aus der Fallbasis gelöst, und die hierzu benötigte Rechenzeit wird ermittelt. In der Testphase wird für die Lösung jedes einzelnen Problems eine Zeitschranke von 200 CPU-Sekunden eingeführt. Kann ein Problem innerhalb dieser Zeitschranke nicht gelöst werden, so wird es als ungelöst gewertet. Die so ermittelten Meßwerte werden sodann statistisch aufbereitet, um die Hypothesen zu testen. Um die Vergleichbarkeit der Meßwerte zu garantieren, werden alle Experimente auf SPARC-ELC Computern mit gleichem Speicherausbau durchgeführt. Als Rechenzeit wird die reine CPU-Zeit, die der Problemlöseprozeß benötigt, herangezogen.

Erzeugung der Fallbasis

Das Vorhandensein einer Fallbasis ist die Grundvoraussetzung zur Durchführung der Experimente. Hierzu haben wir eine Fallbasis aus 100 Fällen zufallsgesteuert generiert. Jeder Fall beinhaltet eine Problemstellung zusammen mit einer korrekten Lösung. Bei der automatischen Generierung der Fallbasis wird zunächst eine Folge von hintereinander anwendbaren Operatoren zufällig ausgewählt und danach die entstehenden Initial- und Zielzustände berechnet. Um zu garantierten, daß die Fälle möglichst gute Lösungen repräsentieren, wird nachträglich durch vollständige Suche versucht, die Anzahl der in der Zufallslösung vorkommenden Operatoren zu reduzieren. Hierdurch sind Fälle verschiedener Komplexität mit einer Lösungslänge von 4 bis 18 Operatoren entstanden. Die Fälle sind in fünf Schwierigkeitsklassen, die über die Lösungslänge definiert sind, eingeteilt. Um eine Gleichverteilung der Fälle in Bezug auf ihre Schwierigkeit zu erreichen, werden die 100 verwendeten Fälle so ausgewählt, daß genau 20 von jeder Klasse enthalten sind.

Auswahl der Trainings- und Testfälle

Aus dieser Grundmenge von 100 Fällen haben wir verschiedene Trainings- und Testfallmengen zufallsgesteuert ausgewählt:

- 10 disjunkte Trainingsfallmengen mit je 5 Fällen (je ein Fall pro Schwierigkeitsklasse),
- 10 disjunkte Trainingsfallmengen mit je 10 Fällen (je zwei Fälle pro Schwierigkeitsklasse).

Für jede der 20 Trainingsfallmengen ergibt sich eine zugehörige Testfallmenge aus den verbleibenden 95 bzw. 90 Fällen der Grundmenge. Hierdurch ist sichergestellt, daß die Trainings- und Testfallmengen disjunkt sind.

Variation der Problemlösedomäne

Wir setzen hierbei auf der in Abschnitt 8.2 gezeigten Repräsentation auf. Um die Wiederverwendung generalisierter abstrakter Fälle der Wiederverwendung rein generalisierter bzw. rein abstrakter Fälle gegenüberzustellen, haben wir jedoch verschiedene Varianten dieser Problemlösedomäne entwickelt. Um hierbei die Vergleichbarkeit zu gewährleisten, ist in allen Varianten die konkrete Problemlösedomäne identisch.

Rein generalisierende Problemlösedomäne: Hierbei wird auf die Fähigkeit zur Abstraktion vollständig verzichtet. Die abstrakte Problemlösedomäne wird hierzu mit der konkreten Problemlösedomäne gleichgesetzt, und die generische Abstraktionstheorie wird auf die Identität reduziert. Dies führt dazu, daß der Fallabstraktionsalgorithmus ebenfalls auf die Identität reduziert wird.

Rein abstrahierende Problemlösedomäne: Hierbei wird die konkrete Domänentheorie und die generische Abstraktionstheorie unverändert beibehalten, jedoch wird die abstrakte Problemlösedomäne so umformuliert, daß die verwendeten Operatordefinitionen und Inferenzregeln so speziell wie möglich sind und keine Variablen enthalten. Dies führt dazu, daß das Kriterium der Generalität der Operatordefinition (Kriterium 8.4) verletzt ist. Hierdurch wird die Fallgeneralisierung faktisch ausgeschaltet, so daß sich die Wiederverwendung auf abstrakte, nichtgeneralisierte Fälle beschränkt.

Vereinfachte Problemlösedomäne: Diese Problemlösedomäne stellt eine vereinfachte Variante der ursprünglichen Problemlösedomäne dar. Hierbei verzichten wir auf den abstrakten Operator zur Aufspannung (*set_fixation*). Ebenso wird die Vorbedingung, daß das Werkstück vor der Bearbeitung aufgespannt sein muß, bei den verbleibenden Operatoren entfernt. Durch diese Modifikation sind die verbleibenden abstrakten Operatoren nicht mehr weitgehend unabhängig verfeinerbar, d. h. Kriterium 8.2 ist nicht mehr erfüllt.

Die rein generalisierende bzw. rein abstrahierende Problemlösedomäne erlaubt es, den Vorteil der Wiederverwendung generalisierter abstrakter Fälle im Vergleich zu rein generalisierten bzw. rein abstrakten Fällen (vgl. Hpothesen 8.2 und 8.3) zu beurteilen. Alle drei

Domänentheorien ermöglichen die Bestimmung der Abhängigkeit des Verfahrens von der Repräsentation der Problemlösedomäne (vgl. Hypothese 8.7).

8.4.3 Statistische Aufbereitung der Daten

In der Testphase der Experimente werden die Problemlösezeiten für die Probleme aus der Testfallmenge bestimmt. Zum Test der zuvor formulierten Hypothesen ist es erforderlich, die Problemlösezeiten unter verschiedenen experimentellen Bedingungen (z. B. Problemlösezeit bei vollständiger Suche vs. Problemlösezeit bei Wiederverwendung) miteinander zu vergleichen. Ein solcher Vergleich kann auf verschiedene Weisen erfolgen.

Vergleich der mittleren Problemlösezeit

Man bildet für jedes Experiment die über alle Testprobleme gemittelte Problemlösezeit. Die gemittelten Problemlösezeiten, die sich bei verschiedenen experimentellen Bedingungen ergeben, können dann miteinander verglichen werden. Als problematisch erweist sich jedoch hierbei die Behandlung der Testprobleme, die innerhalb der Rechenzeitschranken nicht gelöst werden können. Werden diese Probleme bei der Mittelwertbildung vollkommen ignoriert, so sind die Mittelwerte verschiedener Experimente nicht miteinander vergleichbar, da sie sich auf unterschiedliche Teilmengen der Stichprobe (gelöste Testprobleme) beziehen. Dieses Problem läßt sich abschwächen, wenn für ungelöste Probleme die verwendete Zeitschranke als Problemlösezeit in die Mittelwertbildung eingeht.

Paarweiser Vergleich individueller Problemlösezeiten

Ein alternativer Ansatz wurde von Etzioni & Etzioni [1994] zur Bewertung von Speedup-Lernverfahren vorgeschlagen. Hierbei wird der Vergleich zweier experimenteller Bedingungen E_1 und E_2 auf die individuellen Problemlösezeiten t_1 und t_2 *der einzelnen Testprobleme* zurückgeführt. Für jedes Testproblem wird untersucht, unter welcher Bedingung sich eine kürzere Problemlösezeit ergibt. Um zu testen, ob E_1 generell zu kürzeren Problemlösezeiten führt als E_2, können diese Beobachtungen mit einem einseitigen Vorzeichentest (siehe z. B. [Sachs, 1988], S. 67 ff.) analysiert werden. Dieser Test ist geeignet, da er keine Voraussetzung an die Verteilung der Grundgesamtheit stellt. Der Vorzeichentest überprüft die Nullhypothese

$$H_0 : P(t_1 < t_2) \leq P(t_2 < t_1),$$

also die Hypothese, daß für ein beliebiges Problem die Wahrscheinlichkeit für einen Vorteil von E_1 (also $t_1 < t2$) kleiner oder gleich der Wahrscheinlichkeit für einen Vorteil von E_2 (also $t_2 < t_1$) ist. Die Behandlung der Testprobleme, die bei eine der beiden Bedingungen (E_1 oder E_2) nicht innerhalb der Zeitschranke gelöst werden können, ist offensichtlich; sie werden jeweils als Vorteil für die erfolgreiche Bedingung gewertet. Für Testprobleme, die unter keiner der beiden Bedingungen innerhalb der Zeitschranke lösbar sind, verfolgen wir

eine konservative Auswertungsstrategie und werten diese Beispiele als Unterstützung für die Nullhypothese.

Während durch den Vergleich der gemittelten Problemlösezeiten ein Anhaltspunkt für die Größe der Performanzsteigerung gewonnen werden kann, berücksichtigt der Vorzeichentest die quantitative Größe der Differenzen der Problemlösezeiten nicht. Eine weitere Möglichkeit dies zusätzlich zu berücksichtigen besteht darin, einen Rangtest (siehe z. B. [Keryszig, 1988], S. 343 ff.) durchzuführen. Hierbei werden die Differenzen der Problemlösezeit dem Betrag nach geordnet und ein entsprechender Rang zugewiesen. Dieser Test überprüft dann die Nullhypothese, daß die Summe der positiven Ränge geringer oder gleich der Summe der negativen Ränge ist. Da dieser Test jedoch aufwendiger ist als der Vorzeichentest, verzichten wir so weit wie möglich hierauf.

8.4.4 Experiment 1: Problemlösen ohne Wiederverwendung

Um eine Vergleichsbasis für die Effizienz der Wiederverwendung zu schaffen, wird zunächst das Problemlöseverhalten *ohne* Wiederverwendung von Fällen untersucht. Hierzu wird jedes der 100 Probleme durch vollständige Suche auf der konkreten Problemlösedomäne bearbeitet. Die bereits bei der Lösungsverfeinerung verwendete Suchstrategie der Tiefensuche mit schrittweiser Tiefenvergrößerung (siehe Abschnitt 5.4.4) wird auch hierbei gewählt, um die Vergleichbarkeit der Methoden zu gewährleisten. Für die Lösung jedes Problems wird eine Rechenzeitschranke von 200 CPU-Sekunden eingeführt. Probleme, die nicht innerhalb dieser Zeitschranke gelöst werden können, werden als ungelöst gewertet.

Ergebnisse

Von den 100 Problemen konnten insgesamt 29 gelöst werden. Alle Probleme die eine Lösung von mehr als 8 Operatoren erfordern, waren nicht innerhalb der Rechenzeitschranke lösbar. Abbildung 8.13 zeigt die durchschnittliche Problemlösezeit in Abhängigkeit von der Problemgröße. Als Maß für die Problemgröße dient hierbei die Länge der Lösung.[4] Die gezeigten Problemlösezeiten sind die Mittelwerte für die gelösten Probleme mit gleicher Lösungslänge. Man beachte, daß die Ordinatenachse einen logarithmischen Maßstab aufweist. Für die Probleme mit einer Lösungslänge von mehr als 8 Operatoren wurden die Meßwerte der gelösten Probleme extrapoliert. Geht man hier von dem für diese Suchstrategie typischen exponentiellen Zeitverhalten der Form $t(n) = c \cdot b^n$ aus, so ergibt sich ein Verzweigungsgrad von $b = 2$ und ein Faktor von $c = 739$ ms.

[4] Alle Probleme sind den zuvor generierten Fällen entnommen, so daß für jedes Problem eine entsprechende Lösung existiert, deren Länge zur Beurteilung der Problemkomplexität herangezogen werden kann.

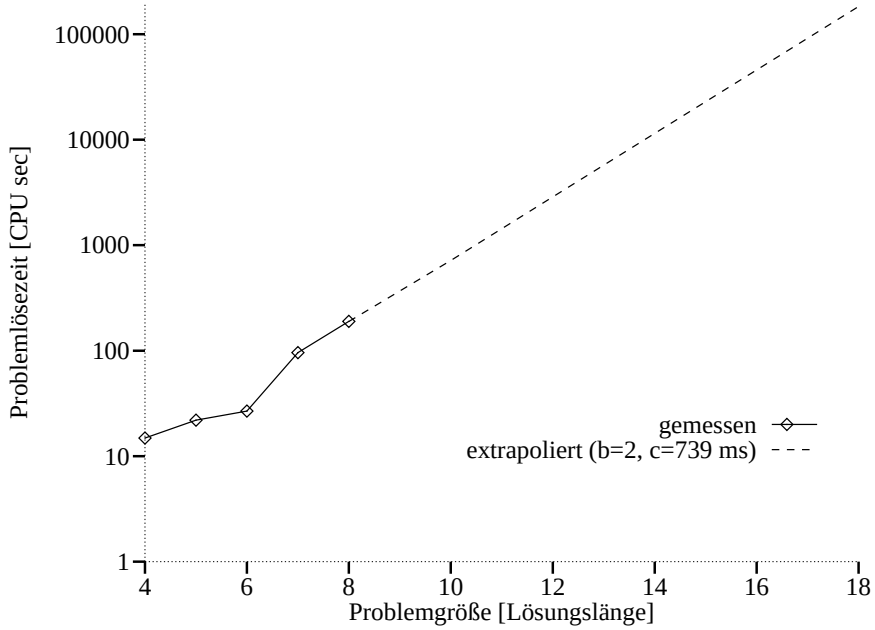

Abbildung 8.13: Problemlösen ohne Wiederverwendung von Fällen. Gezeigt wird die durchschnittliche Problemlösezeit in Abhängigkeit der Problemgröße. Für Probleme, die eine Lösung von mehr als 8 Operatoren erfordern, sind extrapolierte Werte angegeben.

8.4.5 Experiment 2: Flexibilität der Wiederverwendung

Dieses Experiment untersucht die Flexibilität der Wiederverwendung für die Grundgesamtheit der 100 vorliegenden konkreten Fälle. Als Maß für die Flexibilität der Wiederverwendbarkeit eines konkreten Falles wird die Anzahl der Probleme verwendet, die durch die Wiederverwendung dieses Falles effizienter lösbar sind als Problemlösen ohne Wiederverwendung. Für jeden konkreten Fall wird hier untersucht, wieviele der 100 Probleme durch die Wiederverwendung von generalisierten abstrakten Fällen, die aus diesem einen konkreten Fall gewonnenen wurden, in kürzerer Zeit lösbar sind als ohne Wiederverwendung. Ziel dieses Experimentes ist es insbesondere, die Hypothese 8.2 zu überprüfen, d. h. festzustellen, ob sich durch die Kombination von Abstraktion und Generalisierung eine stärkere Flexibilisierung erreichen läßt, als durch die Verwendung einer einzelnen Methode. Hierzu wird die Flexibilität der Wiederverwendung zusätzlich für die rein generalisierende und die rein abstrahierende Problemlösedomäne ermittelt.

Ergebnisse

Abbildung 8.14 zeigt die Wiederverwendbarkeit in Abhängigkeit der konkreten Fälle. Für jede verwendete Problemlösedomäne (rein generalisierend, rein abstrahierend und abstra-

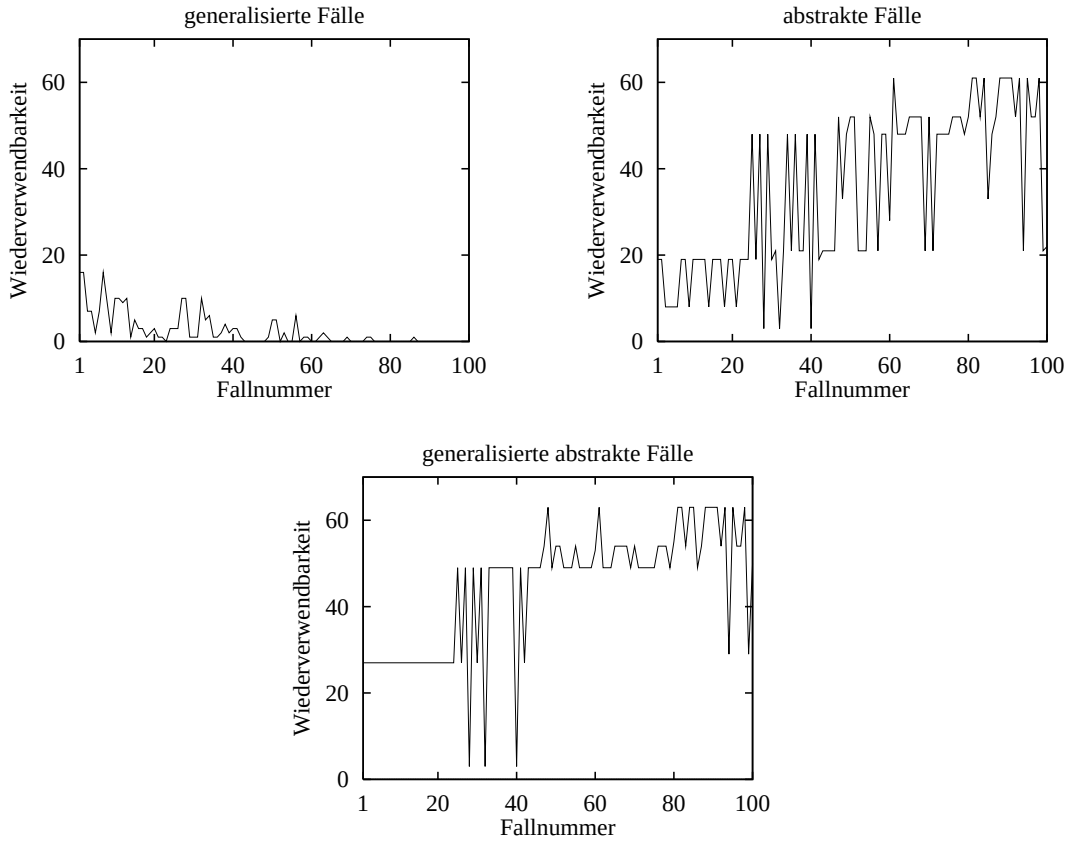

Abbildung 8.14: Flexibilität der Wiederverwendung für rein generalisierte Fälle, rein abstrakte Fälle und generalisierte abstrakte Fälle. Die Wiederverwendbarkeit (Ordinatenachse) wird durch die Anzahl der Probleme, die durch Wiederverwendung effizienter lösbar sind, angegeben.

hierend und generalisierend) ist hierzu eine eigenes Diagramm abgebildet. Auf der Abszisse dieser Diagramme sind diese Fälle gemäß ihrer Komplexität angeordnet; Fall 1 ist ein Fall mit einer Lösungslänge von 4 Operatoren, und Fall 100 ist ein Fall mit einer Lösungslänge von 18 Operatoren.

Vergleicht man die Wiederverwendbarkeit generalisierter abstrakter Fälle mit der Wiederverwendbarkeit rein generalisierter Fälle, so erkennt man einen deutlichen Vorteil für das kombinierte Verfahren. Die Flexibilität war bei der Wiederverwendung generalisierter abstrakter Fälle bei der überwiegenden Anzahl von Fällen größer, als bei der Wiederverwendung rein generalisierter Fälle. Lediglich bei 3 Fällen ergab sich eine gleiche bzw. geringere Flexibilität. Die Analyse mit dem konservativen Vorzeichentest bestätigt dieses Ergebnis mit einer Irrtumswahrscheinlichkeit von $\alpha < 0,1$ %.

Vergleicht man die Wiederverwendbarkeit generalisierter abstrakter Fälle mit der Wiederverwendbarkeit rein abstrakter Fälle, so erkennt man auch hier einen Vorteil für das

kombinierte Verfahren. Wenn auch der Unterschied in der Flexibilität deutlich geringer war als im vorherigen Vergleich, so ergab sich hier in lediglich 3 Fällen eine gleiche Flexibilität. Die Analyse mit dem konservativen Vorzeichentest bestätigt auch dieses Ergebnis mit einer Irrtumswahrscheinlichkeit von $\alpha < 0,1$ %. Somit bestätigt sich die Hypothese 8.2.

Des weiteren fällt bei der Betrachtung der Flexibilität der Wiederverwendung generalisierter Fälle auf, daß diese mit der Komplexität der Fälle abnimmt. Im Gegensatz dazu nimmt die Flexibilität bei der Wiederverwendung abstrakter bzw. generalisierter abstrakter Fälle mit der Komplexität der Fälle eher zu. Eine genauere Untersuchung dieses Phänomens ergab, daß dies vor allem daran liegt, daß die Generalisierung komplexerer Fälle zu einer großen Anzahl von essentiellen Sätzen und Constraints im generalisierten Fall führt. Der Test der Wiederverwendbarkeit eines generalisierten Falles kann dabei bereits das gesetzte Rechenzeitlimit überschreiten und in vielen Fälle sogar länger dauern als das Problemlösen durch vollständige Suche. Benutzt man hingegen ein Abstraktionsverfahren (ggf. zusätzlich zur Generalisierung), so führt dies zu stark vereinfachten (generalisierten) Problembeschreibungen, deren Wiederverwendbarkeit erheblich leichter zu überprüfen ist.

8.4.6 Experiment 3: Vergleich von Kompetenz und Effizienz

Dieses Experiment untersucht die Steigerung von Kompetenz und Effizienz durch die Wiederverwendung. Hierdurch soll insbesondere Hypothese 8.1 bestätigt werden. Außerdem soll untersucht werden, inwieweit die Kombination von Abstraktion und Generalisierung zu besseren Ergebnissen führt, als die Verwendung eines einzelnen Verfahrens. Hierzu wird das PARIS-System zunächst mit einer aus 10 konkreten Fällen bestehenden Trainingsmenge trainiert. Danach wird die Problemlösezeit für das Lösen von Problemen der Testfallmenge (die verbleibenden 90 Fälle) untersucht.

Ergebnisse

Abbildung 8.15 zeigt den Prozentsatz gelöster Probleme in Abhängigkeit von der Problemkomplexität. Hierbei können wir einen deutlichen Kompetenzvorteil der Wiederverwendung abstrakter bzw. generalisierter abstrakter Fälle gegenüber der vollständigen Suche und der Wiederverwendung generalisierter Fälle erkennen. Insgesamt liefert die Kombination beider Verfahren die besten Ergebnisse, insbesondere bei den Problemen mit größerer Komplexität.

Ein ähnliches Ergebnis zeigt sich auch bei der Betrachtung der Problemlösezeiten. Abbildung 8.16 zeigt die durchschnittliche Problemlösezeit gelöster Probleme in Abhängigkeit der Problemkomplexität. Hierbei ist ein deutlicher Effizienzvorteil durch Wiederverwendung generalisierter abstrakter Fälle und auch durch die Wiederverwendung rein abstrakter Fälle erzielt worden. Der Faktor, um die die erforderliche Rechenzeit gegenüber dem Problemlösen ohne Wiederverwendung (gemessene bzw. extrapolierte Werte) reduziert werden konnte, liegt zwischen 2,4 bei einfachen Problemen und bis zu 2400 bei komplexeren Pro-

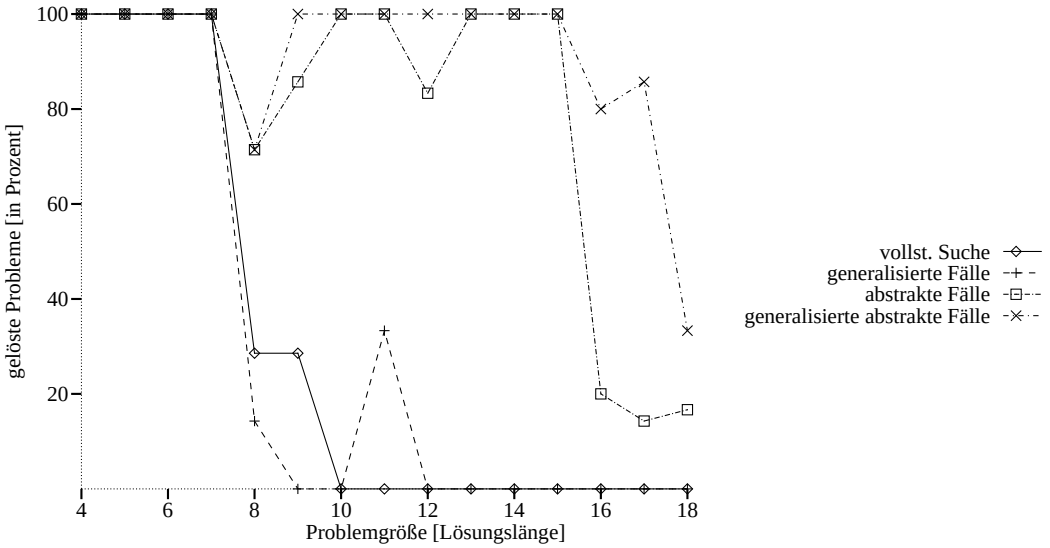

Abbildung 8.15: Prozentsatz gelöster Probleme in Abhängigkeit der Problemkomplexität für a) vollständige Suche, d. h. ohne Wiederverwendung, b) Wiederverwendung generalisierter Fälle, c) Wiederverwendung abstrakter Fälle und d) Wiederverwendung generalisierter abstrakter Fälle.

blemen. Bei komplexeren Problemen wurde hierbei die mittlere Problemlösezeit von mehr als zwei Tagen (extrapolierter Wert) auf etwa 77 Sekunden reduziert.

Des weiteren ist auffällig, daß die Wiederverwendung rein generalisierter Fälle hier nicht zu einer Reduktion der Problemlösezeit führte. Bei den einfacheren Problemen – bei diesen hat die Wiederverwendung generalisierter Fälle überhaupt zu einer Lösung geführt – ist die Effizienz oft sogar geringer als die der vollständigen Suche.

Die in diesem Experiment gewonnenen Ergebnisse sprechen eindeutig für die Gültigkeit der beiden Hypothesen 8.1 und 8.3. Jedoch beruhen diese Ergebnisse auf der (zufälligen) Auswahl einer einzigen Trainingsfallmenge.

8.4.7 Experiment 4: Detaillierte Untersuchung von Kompetenz- und Effizienzsteigerungen

Im folgenden sollen die Ergebnisse des vorherigen Experiments auf eine breitere Basis gestellt werden. Hierzu wird das vorherige Experiment in *20 Testläufen* für verschiedene Trainings- und Testmengen – je 10 bestehend aus 5 Fällen und 10 bestehend aus 10 Fällen (siehe Abschnitt 8.4.2) – durchgeführt. Die hieraus bestimmten Problemlösezeiten werden dann für jeden Testlauf separat analysiert.

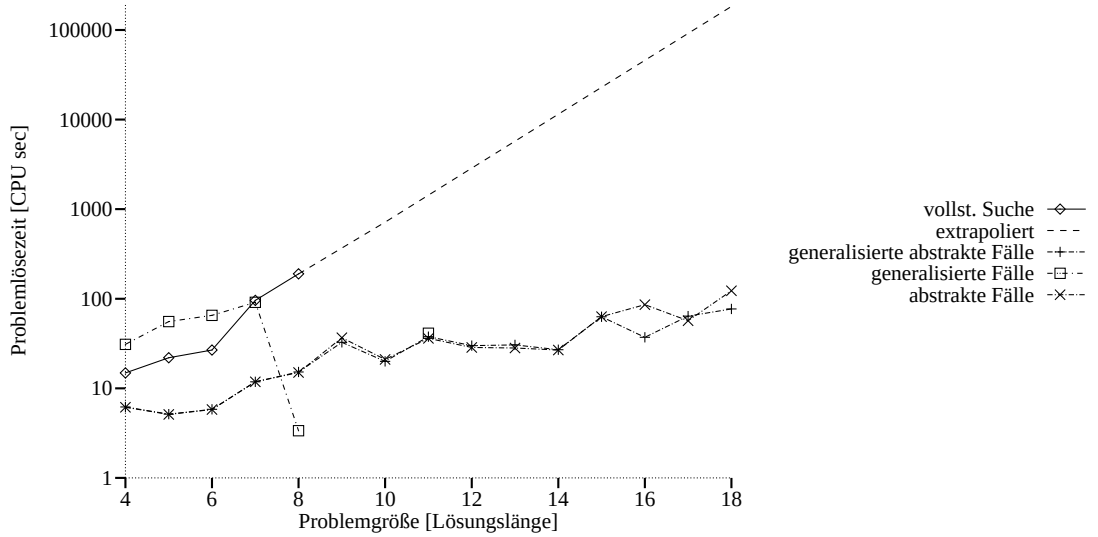

Abbildung 8.16: Problemlösezeit für gelöste Probleme in Abhängigkeit der Problemkomplexität für a) vollständige Suche d. h. ohne Wiederverwendung (gemessen und extrapoliert), b) Wiederverwendung generalisierter Fälle, c) Wiederverwendung abstrakter Fälle und d) Wiederverwendung generalisierter abstrakter Fälle.

Ergebnisse

Tabelle 8.1 zeigt den Prozentsatz der gelösten Testprobleme in Abhängigkeit von der Größe der Trainingsfallmenge und der Art der Wiederverwendung. Diese Prozentangaben sind jeweils Mittelwerte über die 10 verschiedenen Testläufe. Zusätzlich sind in Klammern die Spannweiten notiert. Anhand dieser Tabelle kann man wiederum den deutlichen Vorteil der Wiederverwendung generalisierter abstrakter Fälle im Vergleich zur Wiederverwendung rein abstrakter und rein generalisierter Fälle erkennen. Abstrakte sowie generalisierte abstrakte Fälle führen zu einer deutlichen Kompetenzsteigerung im Vergleich zur vollständigen Suche. Im Gegensatz hierzu ist ersichtlich, daß rein generalisierte Fälle nicht zu einer erkennbar verbesserten Kompetenz im Vergleich zur vollständigen Suche führen.

Tabelle 8.2 zeigt hierzu die durchschnittlichen Problemlösezeiten. Für ungelöste Probleme ist jeweils das Rechenzeitlimit in die Durchschnittsberechnung eingegangen. Die in dieser Tabelle aufgeführten Problemlösezeiten sind jeweils Mittelwerte über die 10 verschiedenen Testläufe. Hier sind zusätzlich in Klammern die Spannweiten notiert. Auch anhand dieser Tabelle erkennt man einen klaren Vorteil der Wiederverwendung generalisierter abstrakter Fälle im Vergleich zur Wiederverwendung rein abstrakter und rein generalisierter Fälle. Während die Wiederverwendung rein generalisierter Fälle keinen Effizienzvorteil im Vergleich zur vollständigen Suche bewirkt, führt die Wiederverwendung abstrakter und generalisieter abstrakter Fälle zu einer deutlichen Leistungssteigerung.

Anzahl der Trainingsfälle	Prozentsatz gelöster Probleme			
	vollst. Suche	GF	AF	GAF
5	29	29	70	84
		(24...40)	(55...91)	(75...91)
10	29	32	66	87
		(26...39)	(61...79)	(72...92)

Tabelle 8.1: Prozentsatz gelöster Probleme aus der Testmenge für vollständige Suche und verschiedene Arten der Wiederverwendung; GF: generalisierte Fälle, AF: abstrakte Fälle, GAF: generalisierte abstrakte Fälle.

Anzahl der Trainingsfälle	Durchschnittliche Problemlösezeit (in CPU-Sekunden)			
	vollst. Suche	GF	AF	GAF
5	157	157	85	56
		(139...167)	(46...110)	(44...79)
10	157	154	88	49
		(143...164)	(62...100)	(42...69)

Tabelle 8.2: Problemlösezeiten für vollständige Suche und verschiedene Arten der Wiederverwendung; GF: generalisierte Fälle, AF: abstrakte Fälle, GAF: generalisierte abstrakte Fälle.

Diese Tendenz wird ebenfalls durch die Analyse mit Hilfe des konservativen Vorzeichentests bestätigt. Tabelle 8.3 zeigt die Anzahl der Testläufe, die zu einer signifikanten Performanzberbesserung im Vergleich zur vollständigen Suche geführt haben. Hierbei wurde die Signifikanz auf dem 5 % Signifikanzniveau (d. h. Irtumswahrscheinlichkeit < 5 %) ermittelt. Erwartungsgemäß führt die Wiederverwendung generalisierter Fälle nicht zu signifikanten Performanzverbesserungen. Bei der Wiederverwendung abstrakter Fälle ergibt sich in etwa der Hälfte der Testläufe eine signifikante Verbesserung, während die Wiederverwendung generalisierter abstrakter Fälle bis auf eine Testlauf immer signifikant besser war. Hierdurch wird die Hypothese 8.1 klar bestätigt.

Abschließend vergleicht Tabelle 8.4 die Leistungsverbesserung, die durch die Kombination von Abstraktion und Generalisierung entsteht, mit der Leistungsverbesserung durch nur eines der Verfahren. Hierzu wird der konservative Vorzeichentest eingesetzt. Die Tabelle zeigt die Anzahl der signifikanten Leistungsverbesserungen durch die Wiederverwendung generalisierter abstrakter Fälle im Vergleich zu rein generalisierten und rein abstrakten Fällen. Hierbei erkennt man, daß der Vorteil der Kombination im Vergleich zur reinen Generalisierung (GAF vs. GF) für alle Testläufe offensichtlich wird; der Vorteil der Kombination im Vergleich zur reinen Abstraktion (GAF vs. AF) jedoch nur in Ausnahmefällen signifikant erscheint, obwohl die Betrachtung der Mittelwerte (siehe Tabelle 8.2) eigent-

Anzahl der Trainingsfälle	Anzahl signifikanter Leistungsverbesserungen GF	AF	GAF
5	0	4	9
10	0	6	10

Tabelle 8.3: Anzahl der Testläufe mit signfikanter (5 % Signifikanzniveau) Leistungsverbesserung durch Wiederverwendung gegenüber vollständiger Suche; GF: generalisierte Fällen, AF: abstrakte Fälle, GAF: generalisierte abstrakte Fälle.

Anzahl der Trainingsfälle	Anzahl signifikanter Leistungsverbesserungen GAF vs. AF	GAF vs. GF
5	1	10
10	2	10

Tabelle 8.4: Anzahl der Testläufe mit signifikanter (5 % Signifikanzniveau) Leistungsverbesserung durch Wiederverwendung generalisierter abstrakter Fälle gegenüber der Wiederverwendung rein abstrakter Fälle (GAF vs. AF) sowie Leistungsverbesserung durch Wiederverwendung generalisierter abstrakter Fälle gegenüber der Wiederverwendung rein generalisierter Fälle (GAF vs. GF).

lich auch hier einen deutlichen Vorteil aufzeigt. Hieraus ergibt sich, daß die Hypothese 8.3 ebenfalls bedingt bestätigt wurde. Die einzige Unsicherheit besteht in der fehlenden Signifikanz des Vorteils der Kombiniation von Abstraktion und Generalisierung im Vergleich zur reinen Abstraktion.

8.4.8 Experiment 5: Abstraktion ohne Wechsel der Repräsentationssprache

In diesem Experiment soll überprüft werden, inwieweit der Wechsel der Repräsentationssprache für den Performanzvorteil, der sich in den vorherigen Experimenten gezeigt hat, verantwortlich ist. Hierzu führen wir zum Vergleich ein Experiment mit dem System ALPINE [Knoblock, 1993] durch, das Bestandteil der PRODIGY-Architektur [Carbonell *et al.*, 1991] ist. ALPINE enthält einen hierarchischen Problemlöser, der Abstraktion durch Elimination von essentiellen Sätzen realisiert. Außerdem enthält ALPINE eine Komponente, mit der Abstraktionsebenen automatisch aus einer Problemlösedomäne generiert werden können. Hierbei ergeben sich sogenannte *geordnet monotone Abstraktionshierarchien* [Knoblock *et al.*, 1991]. Für diese ist sichergestellt, daß kein Backtracking zwischen den einzelnen Abstraktionsebenen erforderlich ist. Jedoch kann Backtracking zwischen der Verfeinerung abstrakter Operatoren nicht ausgeschlossen werden. Bei der Verwendung von Abstraktions-

hierarchien, die nicht geordnet monoton sind, ist ein besonders hoher Backtrackingaufwand zu erwarten, so daß nicht von einer komplexitätsreduzierenden Wirkung ausgegangen werden kann.

Zunächst wurde die konkrete Problemlösedomäne in die Repräsentationssprache von PRODIGY transformiert. Hierbei konnten die Operatoren und die universellen Implikationen sowie die verwendeten Prädikate unverändert bleiben. Mit Hilfe von ALPINE wurde sodann eine Abstraktionshierarchie generiert.

Ergebnis

Es hat sich gezeigt, daß ALPINE nicht in der Lage war eine geordnet monotone Abstraktionshierarchie zu erstellen, die mehrere Abstraktionsebenen enthält. Die berechnete Abstraktionshierarchie besteht aus einer einzigen Ebene, so daß hierarchisches Problemlösen unmöglich ist. Der Grund hierfür besteht darin, daß hierarchische Problemlöser wie ALPINE verschiedene essentielle Sätze nur aufgrund des Prädikatnames in unterschiedliche Abstraktionsebenen einteilen können. Die in diesem Experiment verwendete Repräsentation der Problemlösedomäne kommt jedoch mit einer geringen Anzahl von Prädikaten aus, die nicht so auf Abstraktionsebenen aufgeteilt werden können, daß die enstehende Abstraktionshierarchie geordnet monoton ist.

Es ist natürlich nicht auszuschließen, daß es eine andere, semantisch äquivalente Repräsentation dieser Domäne gibt, für die Abstraktion durch Elimination von essentiellen Sätzen einen Effizienzvorteil bringt. Jedoch ist es fraglich, ob ein Systementwickler eine solche Repräsentation erstellen kann und ob diese zusätzlich adäquat aus der Knowledge-Engineering Sicht ist. Wir können daher, zumindest für die vorliegende Repräsentation, die Hypothese 8.4 als bestätigt ansehen.

8.4.9 Experiment 6: Vergleich der Retrievalverfahren

Dieses Experiment dient dazu, die Effektivität des gewählten Retrievalverfahrens zu beurteilen. Insbesondere sollen die Hypothesen 8.5 und 8.6 überprüft werden. Um den Effekt, der durch unterschiedlich gute Retrievalverfahren entsteht, möglichst deutlich werden zu lassen, wählen wir zum Training eine möglichst große Fallbasis. Das PARIS-System wird daher mit allen 100 verfügbaren Fällen trainiert. Danach löst das System genau die in diesen Trainingsfällen vorkommenden Probleme durch Wiederverwendung. Dieses Experiment wird für verschiedene Retrievalverfahren durchgeführt und die resultierenden Problemlösezeiten werden verglichen. Daß die Trainings- und Testfallmenge in diesem Experiment identisch ist, spielte hierbei keine Rolle, da es nicht um die Beurteilung der Effizientsteigerung gegenüber vollständiger Suche geht, sondern lediglich um den Vergleich verschiedener Retrievalverfahren. Der Vergleich berücksichtigt hierbei die folgenden Verfahren:

Naives Retrieval: Dieses Verfahren organisiert die verfügbaren Fälle in einer linearen Liste und durchsucht diese sequenziell. Der erste wiederverwendbare Fall wird sodann zur Problemlösung eingesetzt.

Hierarchisches Retrieval: Dieses Verfahren verwendet die gelernte Wiederverwendbarkeitshierarchie (siehe Abschnitt 7.2) zum Zugriff auf die Fälle. Hierbei wird der speziellste Fall aus dieser Hierarchie zum Problemlösen ausgewählt. Diese Hierarchie ist nicht mit Hilfe des Kostenmodells optimiert.

Optimiertes hierarchisches Retrieval: Bei diesem Verfahren wird im Vergleich zum zuvor beschriebenen hierarchischen Retrieval zusätzlich eine Optimierung der Hierarchie durch das Kostenmodell durchgeführt (siehe Abschnitt 7.5). Hierdurch werden Zweige in der Hierarchie umsortiert und Knoten können ausgeblendet werden.

Dieses Experiment wird für die Wiederverwendung generalisierter abstrakter Fälle sowie für die Wiederverwendung rein abstrakter Fälle durchgeführt. Da sich in den vorherigen Experimenten die Wiederverwendung rein generalisierter Fälle als relativ erfolglos herausgestellt hat, verzichten wir deshalb auf die Untersuchung dieser Variante.

Ergebnisse

Tabelle 8.5 zeigt den Prozentsatz der gelösten Probleme in Abhängigkeit des gewählten Retrievalverfahrens und der Art der Wiederverwendung. Hierbei erkennt man in allen Fällen einen Vorteil der beiden hierarchischen Verfahren im Vergleich zu einem naiven Ansatz. Die Optimierung bewirkte einen weiteren Kompetenzvorteil jedoch nur bei der Wiederverwendung generalisierter abstrakter Fälle. Erwartungsgemäß ergibt sich dasselbe Ergebnis bei der Betrachtung der durchschnittlichen Problemlösezeit, die in Tabelle 8.6 dargestellt ist. Der Überprüftung dieser Ergebnisse mit dem konservativen Vorzeichentest hat für beide Arten der Wiederverwendung einen signifikanten Vorteil der beiden hierarchischen Verfahren gegenüber dem naiven Verfahren auf dem 5 %-Signifikanzniveau ergeben. Jedoch führte die zusätzliche Optimierung selbst bei der Wiederverwendung generalisierter abstrakter Fälle nicht zu einem signifikanten Vorteil ($\alpha = 54$ %). Hierbei haben sich für

Retrievalverfahren	Prozentsatz gelöster Probleme	
	AF	GAF
naiv	67	54
hierarchisch	72	72
hierarchisch optimiert	72	94

Tabelle 8.5: Prozentsatz gelöster Probleme nach dem Training mit 100 konkreten Fällen für verschiedene Retrievalverfahren und verschiedene Arten der Wiederverwendung; AF: abstrakte Fälle, GAF: generalisierte abstrakte Fälle.

Retrievalverfahren	Durchschnittliche Problemlösezeit (sec.)	
	AF	GAF
naiv	86	101
hierarchisch	74	69
hierarchisch optimiert	74	36

Tabelle 8.6: Durchschnittliche Problemlösezeit nach dem Training mit 100 konkreten Fällen für verschiedene Retrievalverfahren und zwei verschiedene Arten der Wiederverwendung; AF: abstrakte Fälle, GAF: generalisierte abstrakte Fälle.

50 der 100 Testfälle eine kürzere Problemlösezeit ergeben und bei 50 eine gleich große oder größere Problemlösezeit. Da jedoch die Mittelwerte einen deutlichen Vorteil der Optimierung erkennen lassen, ist zu erwarten, daß die Performanzverschlechterungen aufgrund der Optimierung *quantitativ* erheblich geringer ausfallen als die Performanzverbesserungen. Der Vorzeichentest berücksichtigt dies jedoch nicht. Daher wurde zusätzlich ein Rangtest durchgeführt. Dieser Test führt auf dem 5 % Signifikanzniveau zu einer Bestätigung der Hypothese, daß das optimierte hierarchische Retrieval zu einer Verbesserung gegenüber dem hierarchischen Retrieval führt. Dies gilt jedoch nur für die Wiederverwendung generalisierter abstrakter Fälle.

Zusammenfassend hat dieses Experiment eine klare Bestätigung der Hypothese 8.5 ergeben. Hypothese 8.6 konnte jedoch nur für die Wiederverwendung generalisierter abstrakter Fälle bestätigt werden. Außerdem wird auch bei diesem Experiment der Vorteil der Kombination von Abstraktion und Generalisierung im Vergleich zur reinen Abstraktion deutlich. Dies gilt unabhängig vom gewählten Retrievalverfahren. Hierdurch ergibt sich eine weitere Bestätigung von Hypothese 8.3.

8.4.10 Experiment 7: Einfluß der Problemlösedomäne auf die Effizienz der Wiederverwendung

In diesem abschließenden Experiment soll der Einfluß der Problemlösedomäne auf die Effizienz der Wiederverwendung untersucht werden. Gemäß Hypothese 8.7 erwarten wir einen deutlichen Einfluß auf das Problemlöseverhalten des PARIS-Systemes. Darauf deuten bereits die vorherigen Experimente hin; hier wurden deutliche Unterschiede zwischen der Wiederverwendung abstrakter generalisierte Fäll und der Wiederverwendung rein abstrakten bzw. rein generalisierter Fällen aufgedeckt. Die Art der Wiederverwendung wird jedoch vollkommen durch die verwendete Problemlösedomäne bestimmt, so daß diese Experimente ebenso den Einfluß der Problemlösedomäne aufgezeigt haben. In diesem Experiment wird nun zusätzlich die vereinfachte Problemlösedomäne herangezogen. Diese erlaubt die Wiederverwendung generalisierter abstrakter Fälle, wobei jedoch die Operatoren der abstrakten Problemlösedomäne nicht mehr weitgehend unabhängig verfeinerbar sind. Unter

Verwendung dieser Problemlösedomäne wurde das Experiment 4 wiederholt, d. h. das PARIS-System wurde in 20 Testläufen mit den 20 Trainingsfallmengen trainiert und jeweils danach mit der entsprechenden Testfallmenge getestet.

Ergebnisse

Tabelle 8.7 faßt die Ergebnisse dieser Untersuchung zusammen. Man erkennt hier nur eine geringe Leistungssteigerung im Vergleich zur vollständigen Suche, die nicht im Verhältnis zur Leistungssteigerung bei Verwendung der ursprünglichen Problemlösedomäne steht (vergleiche Tabellen 8.1 und 8.2). Auch hat der Vorzeichentest gezeigt, daß in keinem der 20 Testläufe eine signifikante Leistungssteigerung zu erkennen war (bestes Ergebnis: $\alpha = 30$ %). Dieses Beispiel hat die Hypothese 8.7 somit zusätzlich zu den vorherigen Untersuchungen klar bestätigt.

Durchschnittliche Prolemlösezeit	123 s (119 s ... 125 s)
Durchschnittlicher Prozentsatz gelöster Probleme	51 (49 ... 53)
Anzahl signifikanter Testläufe	0

Tabelle 8.7: Durschschnittliche Problemlösezeit, durchschnittlicher Prozentsatz gelöster Probleme und Anzahl der Testläufe mit signifikanter (5 % Signifikanzniveau) Leistungssteigerung für 20 Testläufe mit der vereinfachten Problemlösedomäne.

Kapitel 9

Diskussion und Literaturvergleich

Dieses Kapitel diskutiert die in dieser Arbeit entwickelten Methoden und stellt sie anderen, in der Literatur beschriebenen, Ansätzen aus den Bereichen fallbasiertes Schließen (Abschnitte 9.1 und 9.2), erklärungsbasiertes Lernen (Abschnitt 9.3) und Abstraktion (Abschnitt 9.4) gegenüber. Jedes dieser Gebiete wird nach einem speziellen Schema diskutiert, das wir jeweils zunächst kurz einführen.

9.1 Einordnung ins fallbasierte Schließen

Im folgenden wollen wir das PARIS-System mit aus der Literatur bekannten Techniken, die in fallbasierten Systemen zum Einsatz kommen, vergleichen. Hierbei verwenden wir das bereits im Abschnitt 3.1 eingeführte Prozeßmodell des fallbasierten Schließens [Aamodt & Plaza, 1994] (siehe Abbildung 9.1) zur Strukturierung dieser Diskussion. Wir erörtern daher zunächst die Bedeutung von Fällen und allgemeinem Wissen in verschiedenen fallbasierten Systemen (Abschnitt 9.1.1). Danach stellen wir die in den einzelnen Phasen des Prozeßmodells (Retrieve, Reuse, Revise und Retain) eingesetzten Techniken aus der Literatur den in dieser Arbeit entwickelten Ansätzen gegenüber (siehe Abschnitte 9.1.2 bis 9.1.5). Nach der Diskussion der allgemeineren Techniken werden wir in Abschnitt 9.2 einige fallbasierte Planungssysteme mit dem PARIS-System detailliert vergleichen.

9.1.1 Wissen in fallbasierten Systemen

Wir betrachten zunächst das in fallbasierten Systemen verwendbare Wissen.

Fälle und allgemeines Wissen

Im fallbasierten Schließen werden i. A. zwei verschiedene Arten von Wissen eingesetzt:

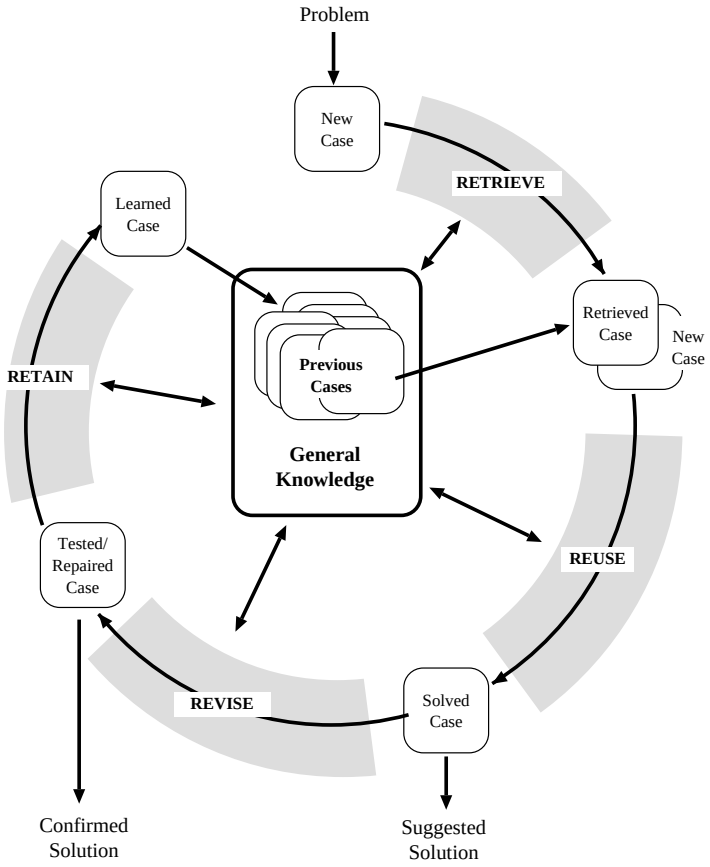

Abbildung 9.1: Der Zyklus des fallbasierten Schließens nach Aamodt und Plaza (1994)

- Fallwissen, auch episodisches Wissen [Strube & Janetzko, 1990] genannt und
- allgemeines Wissen, auch Hintergrundwissen genannt.

Das allgemeine Wissen kann auf verschiedene Weisen repräsentiert werden und in allen Phasen des Prozeßmodells zum Einsatz kommen (vgl. Abbildung 9.1). Der besondere Vorteil fallbasierter Systeme besteht darin, daß sie es erlauben, daß insgesamt erforderliche Wissen zur Bearbeitung einer Aufgabe flexibel auf diese beiden Wissensformen verteilen zu können. In [Richter, 1995] wird dieser Sachverhalt weiter präzisiert. Hiernach kann Wissen in fallbasierten Systemen in 4 verschiedenen „Containern“ enthalten sein:

- in der Repräsentationssprache (verwendete Attribute, Prädikate oder Operatoren),
- in dem bei der Fallauswahl verwendeten Ähnlichkeitsmaß,
- im Wissen zur Lösungsanpassung und
- in den Fällen.

Das Wissen in den ersten drei Containtern ist hierbei allgemeines Wissen. Es wird auch *kompiliertes Wissen* genannt, da es von einem fallbasierten System bereits vor der eigentlichen Problemlösung verwendet wird, z. B. um geeignete Indexstruktueren aufzubauen.

Nützlichkeitsbetrachtungen

Bei der Erstellung eines fallbasierten Systems spielen die Kosten für die Akquisition der verschiedenen Wissensformen eine wichtige Rolle. Sie müssen gegen den Nutzen, der sich durch den Einsatz des Systems ergibt, abgewogen werden. Diesen Sachverhalt wollen wir etwas näher betrachten und führen daher folgende Bezeichnungen ein:

- $\mathcal{F}$ ist eine Fallbasis und $C_{KA}(\mathcal{F})$[1] sind die Kosten zur Erstellung der Fallbasis (Erfassung der Fälle, Transformation in eine Zielrepräsentation, usw.),
- $\mathcal{G}$ bezeichnet das allgemeine Wissen und $C_{KA}(\mathcal{G})$ die Kosten zur Akquisition und Repräsentation dieses Wissens,
- $FBS(\mathcal{G}, \mathcal{F})$ bezeichnt ein fallbasiertes System FBS, dessen Verhalten durch $\mathcal{G}$ und $\mathcal{F}$ bestimmt wird und $N(FBS(\mathcal{G}, \mathcal{F}))$ steht für den mittleren Nutzen durch den Einsatz des fallbasierten Systems bezogen auf eine Anfrage.

Die Entwicklung eines fallbasierten Systems FBS unter Verwendung der Fallbasis $\mathcal{F}$ und des allgemeinen Wissens $\mathcal{G}$ ist genau dann rentabel, wenn folgendes gilt:

$$m \cdot N(FBS(\mathcal{G}, \mathcal{F})) > C_{KA}(\mathcal{F}) + C_{KA}(\mathcal{G}),$$

wobei m die Anzahl der Anfragen an das System ist.

Verschiedene fallbasierte Systeme FBS unterscheiden sich nun gerade darin, wie sie Fälle und allgemeines Wissen verarbeiten und welches Systemverhalten sich bei einer speziellen Aufteilung zwischen diesen beiden Wissensformen ergibt. Eine Vergrößerung der Menge des verwendeten allgemeinen Wissens kann sich wie folgt auswirken:

Auswirkung A: Verbesserung der Korrektheit oder der Qualität von Lösungen,[2]

Auswirkung B: Verlängerung der Problemlösezeiten (zeitaufwendiges Anwenden des allgemeinen Wissens)

Auswirkung C: Verkürzung der Problemlösezeiten (stärker fokussierte Suche).

Nachfolgend wollen wir diskutieren, auf welche Art Wissen in verschiedenen Ansätzen aus der Literatur repräsentiert wird und welche Auswirkungen die Verwendung dieses Wissen auf das Verhalten des entsprechenden fallbasierten Systems hat.

[1] KA in C_{KA} steht für *Knowledge Acquisition*.

[2] Hierunter kann man auch die Vergrößerung der Kompetenz betrachten, wenn ungelöste Probleme als Lösungen von ungenügender Qualität und Korrektheit angesehen werden.

Allgemeines Wissen in der Retrieve-Phase

Eine Reihe von Ansätzen, die in der Literatur diskutiert werden, betrachten nur das Retrieval von Fällen und verzichten vollständig auf die Lösungsanpassung. Hierbei kann allgemeines Wissen, das in der Retrieval-Phase verwendet wird, auf verschiedene Arten repräsentiert sein, wie z. B. in Form von

- Regeln (z. B. in [Branting, 1991; Cain *et al.*, 1991; Janetzko *et al.*, 1992; Bento & Costa, 1994; Sebang & Schoenauer, 1994]),
- semantischen Netzwerken (z. B. in [Bareiss, 1989; Porter *et al.*, 1990; Aamodt, 1991]),
- kausalen Modellen (z. B. in [Barletta & Mark, 1988]) oder
- qualitativen Modellen (z. B. in [Goel *et al.*, 1991; Navinchandra *et al.*, 1991; Cunningham & Smyth, 1994]).

Bei allen diesen Ansätzen wird das Wissen zur Laufzeit verwendet, um die Ähnlichkeitsbestimmung zu verbessern. Dies wirkt sich positiv auf die Korrektheit der entstehenden Lösungen aus (Auswirkung A), ist jedoch in der Regel auch mit einem erhöhten Berechnungsaufwand verbunden (Auswirkung B). Ist man daran interessiert ein System zu erstellen, daß bestimmten Korrektheits- oder Qualitätsanforderungen genügt, so hat man die Wahl, entweder eine Fallbasis mit einer hinreichenden Größe zu erstellen und auf allgemeines Wissen ganz zu verzichten, oder eine kleinere Fallbasis zu erstellen und allgemeines Wissen zu akquirieren (vgl. [Globig & Wess, 1993]). Welchen Weg man wählt, hängt sehr stark von der vorliegenden Anwendungssituation ab. Man muß jedoch versuchen, die Gesamtkosten $C_{KA}(\mathcal{F}) + C_{KA}(\mathcal{G})$ zu minimieren.

Allgemeines Wissen in der Retrieval und Reuse-Phase

Fallbasierte Ansätze, die sowohl das Retrieval als auch die Wiederverwendung von Fällen berücksichtigen, benutzen allgemeines Wissen dazu, den Prozeß der Lösungsanpassung zu modellieren. Auch hierzu wird Wissen in Form von

- Regeln (z. B. in [Hammond, 1986; Smyth & Keane, 1994; Bergmann *et al.*, 1994b]),
- kausalen Modellen (z. B. [Koton, 1988; Branting & Hastings, 1994; Karamouzis & Feyock, 1994]),
- qualitativen Modellen (z. B. in [Goel & Chandrasekaran, 1989; Cunningham & Smyth, 1994; Bergmann *et al.*, 1994a]) oder
- Constraints (z. B. in [Hinrichs & Kolodner, 1991; Faltings *et al.*, 1991; Hua *et al.*, 1994])

eingesetzt. Durch die Lösungsanpassung kann ein fallbasiertes System Lösungen erstellen, die in den Fällen selbst noch nicht vorgekommen sind. Solche Ansätze sind besonders dann von Nutzen (im Sinne der Nützlichkeitsbetrachtungen von Seite 217), wenn der Raum der möglichen Lösungen sehr groß ist und nicht für jede mögliche Lösung ein Fall vorhanden

ist. In dieser Situation kann die Lösungsanpassung eine deutliche Verbesserung der Korrektheit und Qualität der Lösungen (Auswirkung A) bewirken. Hierbei ist es oft so, daß die Menge der verfügbaren Fälle $\mathcal{F}$ mehr oder weniger feststeht. Für die Adaption sollte man versuchen, soviel allgemeines Wissen $\mathcal{G}$ zu repräsentieren, daß das resultierende System den Anforderungen genügt. Ein Seiteneffekt, der sich aufgrund des Einsatzes von Adaptionswissen bei diesen Ansätzen ergibt, ist die Verlängerung des Problemlösevorgangs durch den Adaptionsprozeß (Auswirkung B).

Allgemeines Wissen in der fallbasierten Planung und Konfigurierung

Eine Reihe von Ansätzen im Bereich der Planung und Konfiguration basieren auf der Integration von fallbasierten und generischen Techniken (z. B. [Kambhampati & Hendler, 1992; Veloso, 1994a; Paulokat & Wess, 1994; Muñoz *et al.*, 1994; Köhler, 1994; Pfitzner, 1993; Günter, 1994]). Sie setzen daher allgemeines Wissen voraus, das ausreichend ist um, vorgelegte Probleme im Prinzip auch ohne den Einsatz von Fällen zu lösen.[3] Dieses allgemeine Wissen ist durch die Problemlösedomäne, also z. B. durch die verfügbaren Operatoren vorgegeben. Es wird hierbei gleichermaßen für die Fallauswahl als auch für die Lösungsanpassung verwendet. Solche Ansätze können nutzbringend in Domänen eingesetzt werden, in denen hohe Anforderungen an die Korrektheit der Lösung bestehen. Das allgemeine Wissen ist hierzu unverzichtbar (Auswirkung A). Jedoch hat man es hier üblicherweise auch mit dem Problem zu tun, daß die Verarbeitung dieses Wissens hohe Kosten verursacht, so daß die Problemlösezeit mit steigender Komplexität der Problemlösedomäne eher zunimmt (Auswirkung B).

Allgemeines Wissen im PARIS-System

Auch der in dieser Arbeit vorgestellte Ansatz basiert auf der Integration von fallbasierten und generischen Techniken (vgl. [Bergmann *et al.*, 1996]). Der Unterschied zu den zuvor beschriebenen Ansätzen besteht jedoch darin, daß das Hintergrundwissen in PARIS umfangreicher ist, da es aus der konkreten und der abstrakten Problemlösedomäne sowie der generischen Abstraktionstheorie besteht. Somit erlaubt es PARIS, zusätzliches Domänenwissen über abstrakte Zusammenhänge zu repräsentieren und dieses für Retrieval und Wiederverwendung auszunutzen. Dieses (Hintergrund-) Wissen wird gleichermaßen in allen Komponenten des Systems verwendet. Auch hierbei dient das allgemeine Wissen dazu, die Korrektheit der entstandenen Lösung zu garantieren (Auswirkung A). Darüber hinaus konnte aber auch gezeigt werden, daß durch eine Vergrößerung der Menge von allgemeinem Wissen in Form von geeigneten Abstraktionsebenen eine deutliche Reduktion der Problemlösezeit (Auswirkung C) erreicht werden kann. Es zeigte sich, daß die Wiederverwendung abstrakter Fälle zu deutlichen Leistungsverbesserungen gegenüber der

[3] Bei der Verwendung von reinen generischen Techniken treten die in dieser Arbeit behandelten Komplexitätsprobleme auf.

Wiederverwendung generalisierter Fälle (keine Abstraktionsebenen) führt (vgl. Abschnitte 8.4.6 und 8.4.7). Ebenso führt die Verwendung einer vereinfachten abstrakten Problemlösedomäne (weniger Operatoren) zu einer geringeren Leistungsfähigkeit als die Verwendung einer umfangreicheren abstrakten Problemlösedomäne (vgl. Abschnitt 8.4.10).

Nachfolgend wollen wir uns auf Methoden konzentrieren, die zur Realisierung der einzelnen Phasen des Prozeßmodells des fallbasierten Schließens (siehe Abbildung 9.1) eingesetzt werden.

9.1.2 Retrieval: Auswahl ähnlicher Fälle

Im fallbasierten Schließen erfolgt die Fallauswahl in der Regel aufgrund der *Ähnlichkeit* zwischen einem Fall der Fallbasis und der Problemstellung [Wess, 1995], wobei verschiedene Ähnlichkeitsmaße mit verschiedenen Semantiken zum Einsatz kommen können [Richter, 1992a]. In fallbasierten Planungssystemen besteht die Aufgabe des Retrievals im Finden eines Falles, dessen Lösung sich zur Lösung des aktuellen Problems anpassen läßt. Der Grad der Ähnlichkeit sollte sich hierbei am erforderlichen Aufwand zur Lösungsadaption orientieren. Je geringer dieser Aufwand ist, desto höher wird die Ähnlichkeit bewertet [Kambhampati, 1990b; Humm, 1991]. Der Adaptionsaufwand läßt sich jedoch nicht a priori bestimmen. Daher kann das Ähnlichkeitsmaß nur eine Abschätzung hierfür liefern. Solche Abschätzungen gehen bei den meisten Systemen (siehe z. B. [Kambhampati & Hendler, 1992; Paulokat & Wess, 1994; Muñoz *et al.*, 1994]) auf die Idee des sogenannten „footprinting“ [Veloso, 1992a; Veloso, 1994a] zurück. Hierbei wird für die im Fall vorliegende Lösung die Menge der relevanten essentiellen Sätze des Initial- und Zielzustands bestimmt. Darauf aufbauend wird die Ähnlichkeit durch den Grad der Übereinstimmung mit dem Initial- und Zielzustand der aktuellen Problemstellung ermittelt.

Ähnlichkeit im PARIS-System

Die Retrievalkomponente des PARIS-Systems wählt einen generalisierten abstrakten Fall aus der Fallbasis aus. Hierbei wird die Wiederverwendbarkeitshierarchie zur Indexierung der Fallbasis verwendet. Der Wiederverwendbarkeitsbegriff definiert dabei eine spezielle Form der Ähnlichkeitsberechnung, die durch eine Präferenzrelation $F_1 \succeq_p F_2$ [Richter, 1992a] formalisiert werden kann. Diese Präferenzrelation besagt, daß F_1 ähnlicher ist zu p als F_2. Die folgende Präferenzrelation spiegelt das im PARIS System verwendete Ähnlichkeitsmaß wieder:

$$sim(p, GAF_1) \leq sim(p, GAF_2) \iff GAF_2 \succeq_p GAF_1 \iff GAF_2 \gtrsim p \text{ und } GAF_2 \sqsubseteq GAF_1.$$

Diese Präferenzrelation besagt, daß GAF_2 ähnlicher ist zu p als GAF_1, genau dann wenn GAF_2 schwach wiederverwendbar ist für p und GAF_1 flexibler wiederverwendbar ist als GAF_2. Diese Definition gibt genau die Retrievalstrategie wieder, mit der ein generalisierter abstrakter Fall ausgewählt wird.

Die Ähnlichkeit hängt hierbei auch indirekt vom Grad der Abstraktionsebene ab, auf der der generalisierte abstrakte Fall liegt. Je spezieller ein Fall ist, desto tiefer ist er in der Wiederverwendbarkeitshierarchie angeordnet. Ein solcher Fall wird bei der Fallauswahl gegenüber einem abstrakteren Fall bevorzugt. Die Präferenzrelation gibt folglich Fällen auf einer niedrigen Abstraktionsebene den Vorzug gegenüber Fällen auf einer höheren Abstraktionsebene. Diese Überlegung führt zu einer alternativen Definitionsmöglichkeit für den Ähnlichkeitsbegriff, auf der Basis der Abstraktionsebene, auf der ein Fall angesiedelt ist: je niedriger die Abstraktionsebene ist, auf der die Problemstellung des aktuellen Problems mit dem im Fall übereinstimmt, desto höher ist die Ähnlichkeit zwischen Problem und Fall [Bergmann & Dannenmann, 1992; Bergmann *et al.*, 1994a]. In Experimenten konnte gezeigt werden, daß dieses Ähnlichkeitsmaß einen wichtigen Beitrag zur Leistungsfähigkeit des Gesamtsystemes leistet (vgl. Abschnitt 8.4.9), da es eine gute Abschätzung des Wiederverwendungsaufwandes eines generalisierten abstrakten Falles liefert.

9.1.3 Reuse: Anpassung der Lösung

Die im fallbasierten Schließen entwickelten Ansätze zur Lösungsanpassung lassen sich grob in drei Kategorien unterteilen [Cunningham *et al.*, 1994]:

- Adaption durch Ersetzung,
- Adaption durch Transformation und
- Adaption durch (partielle) Neugenerierung.

Die Adaption durch Ersetzung ist die einfachste Art der Anpassung und erlaubt das Anpassen einzelner Lösungsparameter. Hierbei wird jedoch die Voraussetzung gemacht, daß die anzupassenden Parameter unabhängig voneinander sind. Die Struktur der Lösung kann bei diesem Ansatz nicht verändert werden. Dies ist für analytische Problemstellungen oder einfache syntetische Aufgaben ausreichend. Beispiele hierfür werden von Hennessy & Hinkle [1991] und Cunningham *et al.* [1994] gegeben.

Adaption durch Transformation erlaubt hingegen strukturelle Änderungen in der Lösung. Hierbei wird domänenspezifisches Adaptionswissen z. B. in Form von Regeln eingesetzt. Dieses beschreibt verschiedene Arten der Änderung der Lösung in Abhängigkeit der Änderung der Problemstellung. Durch Kombination verschiedener einzelner Anpassungsschritte wird die im Fall enthaltene Lösung schließlich zu einer Lösung des aktuellen Problems transformiert (siehe Abbildung 9.2). Beispiele für diese Vorgehensweise finden sich sowohl in der Diagnose (z. B. [Koton, 1988; Macchion & Vo, 1994; Bergmann *et al.*, 1994b]), im Design (z. B. [Hinrichs & Kolodner, 1991; Börner, 1994] als auch in der Planung (z. B. [Hammond, 1989; Smyth & Cunningham, 1992; Yang & Lu, 1994]).

Adaption durch partielle Neugenerierung ist die mächtigste Art der Adaption. Ein typischer Vertreter ist die Derivational-Analogy Methode [Carbonell, 1986] die bereits in Abschnitt

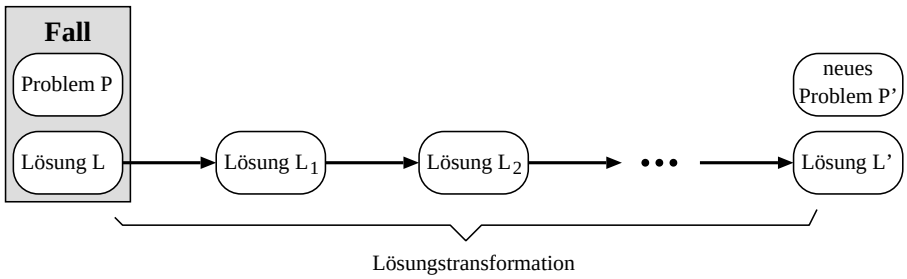

Abbildung 9.2: Lösungsanpassung durch Transformation

3.1 eingeführt wurde. Adaption durch partielle Neugenerierung setzt einen konventionellen Problemlöser voraus. Dieser wird bei der Adaption eingesetzt, um die Teile der Lösung, die der Anpassung bedürfen, neu zu generieren. Im Extremfall kann die Lösung sogar vollständig neu generiert werden. Eine scharfe Abgrenzung gegen rein generative Ansätze ist daher nicht möglich. Beispiele für Adaption durch partielle Neugenerierung finden sich im Bereich fallbasierter Ansätze zur Diagnose (z. B. [Slattery *et al.*, 1993; Pews, 1994; Bergmann *et al.*, 1994a; Portinale *et al.*, 1994], zum Design (z. B. [Mostow, 1989; Hua *et al.*, 1994]) und zur Planung (z. B. [Kambhampati & Hendler, 1992; Veloso, 1992a; Bhansali & Harandi, 1993; Ihrig & Kambhampati, 1994; Paulokat & Wess, 1994; Muñoz *et al.*, 1994; Hanks & Weld, 1995]).

Wiederverwendung in PARIS

Auch der Adaptionsansatz, der im PARIS-System verfolgt wird, gehört zur Kategorie Adaption durch partielle Neugenerierung. Diese Neugenerierung wird durch die Lösungsspezialisierung und -verfeinerung realisiert. Hierbei wird die im Fall enthaltene abstrakte Lösung zwar nicht verändert, jedoch werden in der Lösungsspezialisierung neue Detaillösungen durch den generativen Problemlöser erzeugt. Je nach Fallähnlichkeit kann der Umfang dieser Neugenerierung marginal (Wiederverwendung eines sehr speziellen abstrakten Falles) oder auch signifikant (Wiederverwendung eines abstrakten Falles auf einer sehr hohen Abstraktionsebene) sein. Im Extremfall kann die Lösung sogar vollständig durch den Problemlöser generiert werden. Gerade dies macht im Vergleich zu anderen Systemen die Flexibilität der Wiederverwendung aus. Wie bereits in den Abschnitten 8.3 und 8.4.5 gezeigt wurde, ermöglicht dieser Ansatz die Wiederverwendung von Fällen, die sich auf der vorliegenden konkreten Problemlösedomäne vollständig unterscheiden. Eine solche flexible Wiederverwendung wird zur Zeit durch keinen anderen Ansatz ermöglicht.

9.1.4 Revise: Überprüfung der Lösung

In der Revise-Phase wird die durch die Lösungsadaption entstandene Lösung überprüft. Dies kann sowohl durch Simulation der Lösung oder noch besser durch Test in der realen Welt geschehen. Bislang ist diese Phase in keinem fallbasierten System thematisiert worden. Bei der fallbasierten Planung mit Adaption durch partielle Neugenerierung ist jedoch zumindest eine Simulation der entstandenen Lösung nicht erforderlich, da die Korrektheit der Lösung in Bezug auf die Repräsentation der Problemlösedomäne bei den meisten Ansätzen gewährleistet ist. Dies gilt insbesondere auch für den im PARIS-System verfolgten Wiederverwendungsansatz. Wenn eine Lösung gefunden werden kann, so ist sichergestellt, daß diese auch korrekt ist.

9.1.5 Retain: Die Lernphase

Durch die Retain-Phase werden aufgrund neuer Fälle bzw. vorheriger Problemlöseerfahrungen „Anpassungen“ im fallbasierten System vorgenommen. Diese Phase behandelt also den Lernaspekt im fallbasierten Schließen. Im einfachsten Fall besteht Lernen in der Hinzunahme neuer Fälle in die Fallbasis. Dies wird durch nahezu alle fallbasierten Systeme unterstützt; hierin liegt eine wichtige grundsätzliche Stärke fallbasierter Systeme. Hierbei ist es ggf. erforderlich, die Indexstrukturen anzupassen bzw. für die neuen Fälle zu erweitern. Darüber hinaus sind verschiedene *Fallaufnahmestrategien* untersucht worden, die garantieren sollen, daß die Fallbasis nicht durch neue Fälle überschwämmt wird (z. B. [Aha *et al.*, 1991; Francis & Ram, 1993]). Zu demselben Zweck wurden unlängst erste Kriterien für das Löschen von Fällen in fallbasierten Systemen für analytische Problemstellungen aufgestellt [Smyth & Keane, 1995].

Eine weitere Form des Lernes in fallbasierten Systemen besteht darin, das Ähnlichkeitsmaß anzupassen. Dieser Ansatz wird überwiegend in fallbasierten Systemen für analytische Problemstellungen verfolgt. Ansätze hierzu finden sich z. B. in [Wess, 1991; Salzberg, 1991; Kira & Rendell, 1992; Wess, 1993; Wess & Globig, 1994; Kononenko, 1994; Mohri & Tanaka, 1994]. Ein erster Ansatz für die Gewichtung von Merkmalen in der Planung findet sich in [Muñoz & Huellen, 1994].

Lernen in PARIS

Im PARIS Ansatz wird Lernen auf drei verschiedene Weisen durchgeführt:

1. Zunächst werden durch Fallabstraktion und Fallgeneralisierung generalisierte abstrakte Fälle aus konkreten Fällen gelernt und in der Fallbasis gespeichert. Dies entspricht der klassischen Form des Lernens in fallbasierten Systemen, wobei jedoch zusätzlich eine Vorverarbeitung der Fälle stattfindet.

2. Die Wiederverwendbarkeitshierarchie wird aufgrund der in den neuen Fällen enthaltenen Erfahrung inkrementell angepaßt und um die neuen Fälle erweitert. Interpretiert man die Wiederverwendbarkeitshierarchie als Ähnlichkeitsmaß, wie dies in Abschnitt 9.1.2 diskutiert wurde, so kann man auch hier davon sprechen, daß dieses Ähnlichkeitsmaß durch den in Abschnitt 7.2 dargestellten Algorithmus gelernt wird. Dieser Algorithmus bestimmt nämlich die Anordnung der Fälle in dieser Wiederverwendbarkeitshierarchie und lernt so die Wiederverwendbarkeitsrelation (siehe Definition 7.2).

3. Schließlich führt die Optimierung dieser Wiederverwendbarkeitshierarchie aufgrund der aktuellen Verteilung der Fälle zum Ausblenden von Fällen aus der Fallbasis. Durch diesen Optimierungsansatz wird das „Überschwämmen" der Fallbasis durch neue Fälle verhindert. Dies kann man auch als eine Form des „Vergessens" betrachten. Im Gegensatz zu dem Ansatz von Smyth und Keane [1995] ermöglicht der in PARIS verfolgte Ansatz jedoch auch ein Reaktivieren oder „Wiedererinnern" bereits ausgeblendeter bzw. vergessener Fälle.

9.2 Fallbasierte Planungssysteme

Die vorherigen Betrachtungen werden nun durch einen tiefergehenden Vergleich mit abgeschlossenen und aktuell in der Erforschung befindlichen Ansätzen zur fallbasierten Planung intensiviert. Hierbei konzentrieren wir uns auf Systeme, die auf einer Kombination von generischen und fallbasierten Techniken beruhen. In jedem der nachfolgenden Abschnitte behandeln wir ein fallbasiertes System im Detail und vergleichen es mit dem PARIS-System. Hierbei stellen wir folgende Punkte in den Vordergrund:

- den zugrundeliegenden generischen Problemlöser,
- die Form des verwendeten allgemeinen Wissens,
- die Art der Wiederverwendung,
- das verwendete Ähnlichkeitsmaß und
- die Organisation der Fallbasis.

9.2.1 PRODIGY/Analogy

PRODIGY [Carbonell *et al.*, 1991] ist die an der Carnegie Mellon University entwickelte Architektur, mit der die Integration von Lernen und Problemlösen untersucht wird. PRODIGY enthält eine Vielzahl von Lernkomponenten, die mit einem domänenunabhängigen Problemlöser zusammenarbeiten und hierdurch dessen Performanz erhöhen. PRODIGY/ANALOGY [Veloso, 1992a; Veloso, 1994a] bezeichnet die fallbasierte Variante von PRODIGY.

Generischer Problemlöser: Ebenso wie PARIS basiert der Problemlöser von PRODIGY/ANALOGY auf der Suche im Zustandsraum. Jedoch ist dieser nicht hierarchisch, d. h. PRODIGY/ANALOGY kann nicht mit verschiedenen Abstraktionsebenen arbeiten.

Repräsentation des allgemeinen Wissens: PRODIGY/ANALOGY ist ein domänenunabhängiger fallbasierter Problemlöser, der eine Repräsentation der Problemlösedomäne (siehe [Blythe *et al.*, 1992]) voraussetzt, ähnlich der des PARIS-Systems. Auch hier wird eine Problemlösedomäne durch eine Menge von Strips-Operatoren und Regeln beschrieben. Die Mächtigkeit dieser Domänenrepräsentation ist dabei äquivalent zu der von PARIS.

Ansatz zur Wiederverwendung: PRODIGY/ANALOGY behandelt die Wiederverwendung von Problemlösefällen auf einer Repräsentationsebene und verfolgt dabei einen Derivational-Analogy-Ansatz [Carbonell, 1986] (vgl. auch Abschnitt 3.1.7). Um einen Problemlösefall wiederzuverwenden, macht PRODIGY/ANALOGY die Voraussetzung, daß die in den Fällen enthaltenen Probleme bereits zuvor durch den Problemlöser gelöst worden sind. Probleme, die der Problemlöser aufgrund von Suchraumkomplexität nicht lösen kann, können nicht als Problemlösefälle wiederverwendet werden. Eine solche Einschränkung gibt es in PARIS nicht. PARIS erlaubt die Wiederverwendung von Fällen mit der darin vorgegebenen Lösung unabhängig davon, ob der Problemlöser das vorliegende Problem zuvor lösen konnte. Den Abstraktionsprozeß von PARIS kann man dabei als eine Art Rekonstruktion möglicher hierarchischer Problemlöseprozesse (im wesentlichen Problemzerlegungen) ansehen. Der wesentliche Unterschied zwischen PRODIGY/ANALOGY und PARIS besteht jedoch in der Art der Wiederverwendung. PRODIGY/ANALOGY beschränkt sich auf die Wiederverwendung auf einer Abstraktionsebene, während in PARIS die Wiederverwendung auf verschiedenen Abstraktionsebenen thematisiert wird. Beide Ansätze verwenden jedoch nicht den Fall als Ganzes, sondern selektieren die wiederverwendbaren Bestandteile des Falls. In PRODIGY/ANALOGY sind die wiederverwendbaren Bestandteile die einzelnen Entscheidungen des Problemlösers (Auswahl von Zielen, Operatoren, Inferenzregeln, Anordnungen und der Auswahl Variablenbindungen) auf einer einzigen Abstraktionsebene. In PARIS besteht ein wiederverwendbares Bestandteil aus der Summe der Entscheidungen zur Zielauswahl, Operatorauswahl und deren Anordnung, jedoch auf einer der möglichen Abstraktionsebenen. Somit verwendet PARIS immer alle wesentlichen Entscheidungen im Fall wieder, jedoch in unterschiedlichen Detailiertheitsgraden. Durch die Fallgeneralisiergung können in PARIS jedoch die Entscheidungen über Variablenbindung (z. B. von Operationsparametern) während der Wiederverwendung – genauer gesagt während der Lösungsspezialisierung – auf eine, im Vergleich zum ursprünglichen Fall, unterschiedliche Weise getroffen werden. Hierdurch ist eine gewisse Flexibilität der Wiederverwendung auch innerhalb einer Abstraktionsebene gegeben, so wie sie auch durch PRODIGY/ANALOGY möglich ist.

Ähnlichkeitsbestimmung: Diese unterschiedlichen Arten der Wiederverwendung spiegeln sich auch in unterschiedlichen Ansätzen zur Ähnlichkeitsbestimmung wieder. In PA-

RIS wird die Ähnlichkeit aufgrund der Abstraktionsebene, auf der eine Wiederverwendung möglich ist, bestimmt. PRODIGY/ANALOGY verwendet ein Maß, das bestimmt, wie viele der notwendigen Vorbedingungen für die im Problemlösefall enthaltenen Entscheidungen in der aktuellen Problemstellung ebenfalls gegeben sind. Dieses Maß berechnet sich aufgrund des sogenannten „footprints" der Problemlöseentscheidungen. Der Algorithmus, mit dem dieser bestimmt wird, ist mit der Konstruktion der Erklärungen durch den EBL-Ansatz in PARIS stark verwandt. Während in der Fallgeneralisierung zuerst eine Fallerklärung aufgebaut und danach explizit generalisiert wird, wird in PRODIGY/ANALOGY zwar auch eine Fallerklärung aufgebaut, jedoch wird diese nicht generalisiert. Anstatt einer Generalisierung werden die essentiellen Sätze bestimmt, die in der Fallerklärung vorkommen und somit für die vorliegende Lösung relevant sind. Diese werden dann bei der Ähnlichkeitsbestimmung mit denen der aktuellen Problemstellung verglichen und dabei ggf. parametrisiert. Im Gegensatz zu PARIS ist diese Ähnlichkeitsbestimmung statisch, d. h. es werden durch die Lernkomonente keine Änderungen am Maß vorgenommen.

Organisation der Fallbasis: Ein weiterer Unterschied zwischen PRODIGY/ANALOGY und PARIS besteht im Ansatz zur Organisation der Fallbasis. Während PRODIGY/ANALOGY Hashtabellen und Diskriminationsnetzwerke zum effizienten Zugriff auf die Fälle der Fallbasis einsetzt, benutzt PARIS die Wiederverwendbarkeitshierarchie. Im Gegensatz zu PARIS enthält PRODIGY/ANALOGY keinen Ansatz zur Behandlung des Problems der „Überschwämmung" der Fallbasis, sondern speichert immer alle neuen Fälle. Probleme hierdurch haben die Autoren bei den von ihnen verwendeten Domänen und Fallbasen bislang noch nicht beobachtet [Doorenbos & Veloso, 1993].

Vergleichskriterium	PARIS	PRODIGY/ANALOGY
Basisproblemlöser	vorwärtsgerichtete Suche im Zustandsraum	rückwärtsgerichtete Suche im Zustandsraum
generelles Wissen	hierarchische Modellierung: verschiedene Repräsentationen	„flache" Modellierung: eine Repräsentation
Wiederverwendung	komplette Lösung auf unterschiedlichen Abstraktionsebenen	einzelne Entscheidungen des Problemlöseprozesses
Ähnlichkeitsmaß	Höhe der Abstraktionsebene der Wiederverwendung (dynamisch, gelernt)	Voraussetzungen für die Gültigkeit von Entscheidungen (statisch)
Fallbasisverwaltung	Wiederverwendbarkeitshierarchie, Hinzunahme von Fällen, Ein/Ausblenden von Fällen	Diskriminationsnetze, Hinzunahme von Fällen

Tabelle 9.1: Vergleich von PARIS und PRODIGY/ANALOGY

Tabelle 9.1 stellt die wesentlichen Systemmerkmale von PARIS und PRODIGY/ANALOGY nochmals einander gegenüber.

9.2.2 CAPlan/CbC

CAPLAN/CBC [Paulokat & Wess, 1994; Muñoz *et al.*, 1994] ist ein an der Universität Kaiserslautern entwickeltes fallbasiertes Planungssystem, das einige Ähnlichkeiten zu PRODIGY/ANALOGY aufweist. Wie bei PARIS sind auch die wesentlichen Wiederverwendungsmethoden von CAPLAN/CBC domänenunabhängig.

Generischer Problemlöser: CAPLAN/CBC basiert auf dem generativen Problemlöser CAPLAN [Weberskirch, 1994], der eine systematische Suche im Lösungsraum durchführt und ein abhängigkeitsgesteuertes Backtracking verwendet.

Repräsentation des allgemeinen Wissens: CAPLAN/CBC setzt eine Problemlösedomäne voraus, die zwar eine objektorientierte Modellierung von Zuständen ermöglicht, jedoch auf nur einer einzelnen Planungsebene arbeitet. Somit benutzt CAPLAN/CBC, genau wie PRODIGY/ANALOGY, eine nicht-hierarchische Domänenmodellierung.

Ansatz zur Wiederverwendung: Ebenso wie bei PRODIGY/ANALOGY, basiert der Wiederverwendungsansatz von CAPLAN/CBC auf dem Derivational-Analogy Prinzip [Carbonell, 1986] (vgl. auch Abschnitt 3.1.7), wobei sich die Detailrealisierungen jedoch aufgrund der unterschiedlichen Suchverfahren erheblich unterscheiden. CAPLAN/CBC verwendet die im Fall enthaltenen wiederverwendbaren Entscheidungen direkt als Ausgangspunkt für die Suche im generischen Problemlöser. Der in CAPLAN/CBC verwendete SNLP-Planungansatz [McAllester & Rosenblitt, 1991] erlaubt dies auf elegante Weise. Hierbei nimmt der Planer die im Fall enthaltenen Entscheidungen im Rahmen des normalen Suchprozesses ggf. wieder zurück, bzw. ergänzt diese zu einem vollständigen Plan.

Ähnlichkeitsbestimmung: Die Ähnlichkeitsbestimmung in CAPLAN/CBC basiert ebenso wie die Ähnlichkeitsbestimmung von PRODIGY/ANALOGY auf dem „footprinting" Ansatz. Jedoch verwendet CAPLAN/CBC zusätzliches Wissen über die Ordnung der Ziele. Hierbei wird davon ausgegangen, daß es in den meisten Domänen möglich ist, a priori eine partielle Anordnung der essentiellen Sätze des Zielzustandes vorzunehmen, die jeder korrekte Plan notwendigerweise erfüllen muß [Muñoz & Huellen, 1995].

Organisation der Fallbasis: Die Fallauswahl wird dann auf die Fälle beschränkt, deren Pläne diese Anordnungsanforderung vollständig erfüllen. Die Indexierung der Fallbasis geschieht durch eine 4-Ebenen-Zugriffsstruktur, die eine Erweiterung der Indexierungsstruktur von PRODIGY/ANALOGY darstellt.

Unterschiede zwischen CAPlan/CbC und PARIS: Die im vorherigen Abschnitt beschriebenen Unterschiede zwischen PARIS und PRODIGY/ANALOGY in Bezug auf der Art der Wiederverwendung, Ähnlichkeitsbestimmung und Indexierung differenzieren in gleicher Weise auch PARIS und CAPLAN/CBC. Eine Gemeinsamkeit zwischen PARIS und CAPLAN/CBC ist die Domäne (Arbeitsplanung von rotationssymmetrischen Werkstücken), die die Entwicklung beider Systeme inspiriert hat. Jedoch unterscheidet sich die Repräsentation dieser Domäne in einigen Punkten. CAPLAN/CBC verwendet eine abstraktere Repräsentation als PARIS. Z. B. berücksichtigt die konkrete Problemlösedomäne in PARIS die exakten Abmessungen der Werkstückfeatures während diese bei Repräsentation in CAPLAN/CBC nicht dargestellt werden. Jedoch ist die in CAPLAN/CBC verwendete Repräsentation umfassender in dem Sinne, daß sie es erlaubt, ein größeres Spektrum von Werkstücken (z. B. solche, die Gewinde enthalten) zu repräsentieren.

Tabelle 9.2 stellt die wesentlichen Systemmerkmale von PARIS und CAPLAN/CBC nochmal einander gegenüber.

Vergleichskriterium	PARIS	CAPLAN/CBC
Basisproblemlöser	vorwärtsgerichtete Suche im Zustandsraum	rückwärtsgerichtete Suche im Lösungsraum
generelles Wissen	hierarchische Modellierung: verschiedene Repräsentationen	„flache“ Modellierung: eine Repräsentation
Wiederverwendung	komplette Lösung auf unterschiedlichen Abstraktionsebenen	Fall als Startpunkt der Suche im Problemlöser; alle Entscheidungen des Problemlöseprozesses können zurückgenommen werden
Ähnlichkeitsmaß	Höhe der Abstraktionsebene der Wiederverwendung (dynamisch, gelernt)	Voraussetzungen für Gültigkeit von Entscheidungen und Zielordnung (statisch)
Fallbasisverwaltung	Wiederverwendbarkeitshierarchie, Hinzunahme von Fällen, Ein/Ausblenden von Fällen	4-Ebenen Diskriminationsnetze, Hinzunahme von Fällen

Tabelle 9.2: Vergleich von PARIS und CAPLAN/CBC

9.2.3 PRIAR

PRIAR [Kambhampati, 1990b; Kambhampati, 1990a; Kambhampati & Hendler, 1992] ist ein weiterer domänenunabhängiger Ansatz zur Wiederverwendung von Planungsfällen.

Repräsentation des allgemeinen Wissens: In PRIAR wird die Problemlösedomäne durch eine Menge von sogenannten *Tasks* beschrieben, die im wesentlichen den Operatoren anderer Planungsansätze entsprechen. Eine solche Task besitzt eine Vorbedingung und eine Menge von Effekten, die den Vorbedingungen sowie den Add- und Deletelisten von Strips-Operatoren direkt entsprechen. Bei Tasks wird zwischen atomaren und komplexen Tasks unterschieden. Atomare Tasks entsprechen dabei direkt ausführbaren Aktionen, während komplexe Task durch sogennante Taskreduktionsschemata weiter in Untertasks zerlegt werden müssen. Diese Taskreduktionsschemata sind dabei Bestandteil der Problemlösedomäne.

Generischer Problemlöser: PRIAR verwendet den hierarchischen, nicht-linearen Planer NONLIN [Tate, 1977], der im Lösungsraum sucht. Der Planungsprozeß besteht aus einer schrittweisen Verfeinerung von Tasks durch Untertasks mit Hilfe der vorgegebenen Taskreduktionsschemata. Die Verfeinerung erfolgt hierbei derart, daß gewährleistet ist, daß die Vorbedingungen der einzelnen Tasks durch entsprechend angeordnete Vorgängertasks erfüllt werden. Ziel ist ein vollständig reduziertes Tasknetzwerk, bei dem alle Tasks vollständig auf atomare Tasks reduziert worden sind. Dieser hierarchische Problemlöseprozeß sowie die vorausgesetzte Modellierung der Problemlösedomäne haben einige Gemeinsamkeiten mit PARIS. So können komplexe Tasks als abstrakte Operatoren und atomare Tasks als konkrete Operatoren betrachtet werden. Ähnlich wie die generische Abstraktionstheorie in PARIS, dienen die Taskreduktionsschemata dem Übergang zwischen den Abstraktionsebenen. Während die generische Abstraktionstheorie jedoch eine Zustandsabstraktion beschreibt, geben die Taskreduktionsschemata eine Operatorabstraktion vor. Ein weiterer wesentlicher Unterschied zwischen beiden Systemen besteht darin, daß PARIS den Wechsel der Repräsentationssprache zwischen den Abstraktionsebenen ermöglicht, der Problemlöser von PRIAR jedoch nicht. Hierbei können zwar neue Bezeichner für komplexe Tasks eingeführt werden, jedoch müssen die essentiellen Sätze, auf denen diese Arbeiten mit denen der atomaten Tasks konsistent sind. Es können also keine neuen abstrakten essentiellen Sätze eingeführt werden.

Ansatz zur Wiederverwendung: Der Wiederverwendungsansatz in PRIAR basiert auf der Analyse von sogenannten „validation structures“. Diese sind sehr ähnlich zu den Fallerklärungen, die in der Generalisierungsphase von PARIS aufgebaut werden und weisen ebenfalls die Gültigkeit jeder Operatorvorbedingung im vorliegenden Plan nach. Eine „validation structure“ ist dabei eine Menge von Kanten zwischen den einzelnen atomaren Tasks im Plan, wobei jede Kante die Vorbedingung einer Task mit dem entsprechenden Effekt einer Vorgängertask verbindet. Im Gegensatz zu PARIS wird diese „validation structure“ in PRIAR jedoch nicht a priori generalisiert. Wenn ein neues Problem durch Wiederverwendung eines Falls gelöst werden soll, so sucht PRIAR zunächst nach einer Abbildung zwischen den Objekten des Problems und des Falls. Diese Abbildung wird dann auf den Plan und die „validation structure“ angewendet, so daß ein neuer Plan entsteht, dessen „validation structure“ jedoch nur teilweise korrekt und vollständig ist. Mit einer Reihe von

domänanunabhängigen „Reparaturstrategien“ wird dieser Plan angepaßt, so daß dieser zwar eine korrekte „validation structure“ besitzt, aber nicht vollständig in atomare Tasks zerlegt ist. Die daher noch erforderliche Vervollständigung des Plans wird dann direkt mit dem generativen Problemlöser vorgenommen.

Dieser Vorgang kann mit der Wiederverwendung abstrakter Fälle, so wie sie in PARIS im Vordergrund steht, verglichen werden. Ebenso ist das in PRIAR durchgeführte Abbilden der „validation structure“ auf ein neues Problem mit der Fallgeneralisierung und Lösungsspezialisierung vergleichbar. Während in PARIS für bestimmte Objekte durch die Fallgeneralisierung Variablen eingesetzt werden, die anschließend beim Problemlösen durch neue Objekte instantiiert werden können, sucht PRIAR beim Problemlösen direkt nach einer Ersetzung von Objekten im Fall durch neue Objekte des Problems.

Ähnlichkeitsbestimmung und Organisation der Fallbasis: Was die Ähnlichkeitsbetrachtung und die Verwaltung der Fallbasis angeht, bietet PRIAR keine Ansätze. Die Fallbasisverwaltung wird in PRIAR nicht thematisiert. Jedoch wird ein Ähnlichkeitsbegriff verwendet [Kambhampati, 1990b], der im Grundprinzip dem „footprinting“ entspricht.

Tabelle 9.3 stellt abschließend nochmals die wesentlichen Systemmerkmale von PARIS und PRIAR einander gegenüber.

Vergleichskriterium	PARIS	PRIAR
Basisproblemlöser	vorwärtsgerichtete Suche im Zustandsraum	hierarchische, rückwärtsgerichtete Suche im Lösungsraum
generelles Wissen	hierarchische Modellierung: verschiedene Repräsentationen, zustandsabstraktionsbasiert	hierarchische Modellierung: kein Repräsentationswechsel, operatorabstraktionsbasiert
Wiederverwendung	komplette Lösung auf unterschiedlichen Abstraktionsebenen	Domänenunabhängige Transformation von hierarchischen Plänen, Vervollständigung mit generativem Problemlöser
Ähnlichkeitsmaß	Höhe der Abstraktionsebene der Wiederverwendung (dynamisch, gelernt)	Voraussetzungen für Gültigkeit von Entscheidungen (statisch)
Fallbasisverwaltung	Wiederverwendbarkeitshierarchie, Hinzunahme von Fällen, Ein/Ausblenden von Fällen	nicht vorhanden

Tabelle 9.3: Vergleich von PARIS und PRIAR

9.3 Einordnung ins erklärungsbasierte Lernen

Nachfolgend wollen wir das PARIS-System mit aus der Literatur bekannten Techniken aus dem Bereich des erklärungsbasierten Lernens vergleichen. Wie im Abschnitt 3.2 erörtert wurde, versucht erklärungsbasiertes Lernen einen Suchprozeß dadurch effizienter zu gestalten, daß eine aufwendige Inferenzkette durch eine neue Regel ersetzt wird, die die gesamte Inferenzkette zusammenfaßt. In der Literatur werden eine Reihe von Ansätzen beschrieben, die erklärungsbasiertes Lernen auf vielfältige Weise im Bereich des Problemlösens einsetzen. Diese Ansätze unterscheiden sich darin, auf welche Weise Suchaufwand einspart werden kann. Diese Frage wollen wir in der nachfolgenden Diskussion in dem Mittelpunkt stellen.

Hierzu ist es erforderlich eine alternative Betrachtung der Kosten beim Problemlösen und Lernen zu entwickeln, die sich von der in Abschnitt 3.2.5 eingeführten detaillierten Nützlichkeitsbetrachtung von erklärungsbasiertem Lernen unterscheidet. Diese Betrachtungsweise dient als Diskussionsrahmen für die in den Abschnitten 9.3.2 bis 9.3.6 besprochenen Ansätze.

9.3.1 Aufwandsanalyse

Der Problemlöser arbeitet in einem einzigen Suchraum, dem *Basisraum*, und versucht eine Lösung aufgrund von Inferenzen unter Verwendung der Domänentheorie zu finden. Wir nehmen an, daß die Suche im Basisraum sehr aufwendig ist, so daß komplexere Probleme nicht lösbar sind. Vereinfacht gehen wir davon aus, daß dieser Suchraum eine Größe von $O(b^n)$ hat, wobei b der Verzweigungsgrad ist und n die Lösungslänge. Die Anwendung des erklärungsbasierten Lernens oder auch anderer Lernansätze führt nun dazu, daß der Aufwand zur Problemlösung auf verschiedene Anteile aufgeteilt wird. Hierbei sind die folgenden drei Kosten zu unterscheiden:

- **Suche im Basisraum:** Dies ist der ursprüngliche Suchraum, der, ggf. zumindest teilweise, weiterhin durchsucht wird. Diese Kosten wollen wir mit $O(b'^{n'})$ abschätzen, wobei der Verzweigungsgrad b', die Lösungslänge n' oder beide geringer sind als im ursprünglichen Suchraum.

- **Suche im Raum der Lernresultate:** Dies sind die Kosten, die erforderlich sind, um die Menge des durch den Lernprozeß erworbenen Wissens, also z. B. die durch erklärungsbasiertes Lernen erzeugten Regeln, nach nützlichem Wissen zu durchsuchen. Diese Kosten wollen wir mit C_L bezeichnen.

- **Anwendbarkeitstest:** Einige der durch den Lernprozeß erworbenen Regeln müssen auf ihre Anwendbarkeit für eine gegebene Situation hin überprüft werden. Dies erfordert bei komplexen Vorbedingungen einen signifikanten Matchaufwand. Diesen Matchaufwand wollen wir mit C_M bezeichnen.

Ein Lernansatz führt dabei zu einer Performanzsteigerung, wenn die Gesamtheit der Teilkosten geriner ist, als die ursprünglichen Kosten durch reine Suche im Basisraum. Verschiedene Lernansätze können nun dadurch unterschieden werden, wie sie den Aufwand in diese drei Teile aufteilen.

9.3.2 Prodigy/EBL

PRODIGY/EBL [Minton, 1988; Minton *et al.*, 1989; Etzioni, 1990] ist eines der ersten Systeme, das maschinelles Lernen zur Verbesserung der Performanz von Problemlösern einsetzt. PRODIGY/EBL verwendet einen erklärungsbasierten Ansatz, der Einzelentscheidungen dieses Problemlösers analysiert und Erklärungen für den Erfolg oder das Fehlschlagen von Entscheidungen aufbaut. Hierzu verwendet die EBL-Komponente eine Metatheorie als Domänentheorie, die die Funktionsweise dieses Problemlöseres axiomatisiert. Erfolg und Fehlschläge von Entscheidungen werden mit Hilfe dieser Metatheorie erklärt, und die gefundenen Erklärungen werden zu Kontrollregeln generalisiert. Diese Kontrollregeln geben Hinweise über günstige bzw. ungünstige Problemlöseentscheidungen (z. B. die Operatorauswahl) in Abhängigkeit des aktuell vorliegenden Zustands des Problemlösers.

Beim Einsatz dieses Wissens beim Problemlösen, wird bei jeder zu treffenden Problemlöseentscheidung überprüft, ob eine Kontrollregel vorhanden ist, die einer bestimmten Entscheidung Vorrang einräumt oder bestimmte Entscheidungen direkt verwirft. Hierdurch wird die Anzahl der vom Problemlöser durch Suche zu bearbeitenden Entscheidungen reduziert, d. h. der Verzweigungsgrad bei der Suche im Basisraum wird geringer. Dies ist der angestrebte Effekt, der zu einer Performanzsteigerung führen soll. PRODIGY/EBL führt jedoch zusätzliche Aufwände durch die Kosten für den Anwendbarkeitstest der Kontrollregeln und die Kosten für das Durchsuchen aller gelernten Kontrollregeln ein. Zusätzlich fällt erschwerend ins Gewicht, daß diese Kosten für jede zu treffende Problemlösentscheidung anfallen. Hieraus ergibt sich die folgende Aufwandsverschiebung:

$$O(b^n) \longrightarrow O(b'^{\,n} \cdot C_M \cdot C_L) \text{ mit } b' < b.$$

Spezielles Nützlichkeitsproblem

Experimentelle Untersuchungen haben gezeigt, daß dieser Ansatz ein Nützlichkeitsproblem [Minton, 1990b] (siehe auch Abschnitt 3.2.6) aufweist, da der Aufwand C_M zum Matchen der Kontrollregeln sehr groß werden kann. Außerdem werden sehr viele verschiedene Kontrollregeln generiert, so daß der Faktor C_L sehr stark anwächst. Dies ist insbesondere deshalb problematisch, weil PRODIGY/EBL keinen Ansatz zur effizienten Speicherung der Kontrollregeln enthält, sondern einen rein sequenziellen Zugriff verwendet.

9.3.3 Lernen von Makrooperatoren

Das Prinzip der Makrooperatoren wurde in einer Reihe von Ansätzen und Systemen [Fikes *et al.*, 1972; Korf, 1985b; Tadepalli, 1991] untersucht. Während der EBL-Ansatz von PRODIGY/EBL die Generalisierung einer einzelnen Entscheidung vornimmt, werden bei Makrooperatoren Entscheidungsketten bestimmter Größe zusammengefaßt. Hierbei ist die Domänentheorie direkt die vorliegende Problemlösedomäne; eine Axiomatisierung des Problemlösers ist nicht erforderlich.

Beim Problemlösen werden dann anstelle der Operatoren bevorzugt die Makrooperatoren verwendet. Makrooperatoren überbrücken größere Abstände im Suchraum als normale als Operatoren, wodurch die Suchraumtiefe reduziert werden kann. Um jedoch die Vollständigkeit des Problemlösers zu erhalten, müssen die ursprünglichen Operatoren ebenfalls betrachtet werden, so daß sich insgesamt ein größerer Verzweigungsgrad ergibt. Dieser Prozeß läßt sich durch die folgende Aufwandsverschiebung beschreiben:

$$O(b^n) \longrightarrow O(C_M \cdot (b + C_L)^{n'}) \text{ mit } n' < n.$$

Spezielles Nützlichkeitsproblem

Dieser Ansatz führt ebenfalls zu einer Art Nützlichkeitsproblem, wenn die Anzahl der gelernten Makrooperatoren zu stark anwächst oder die Makrooperatoren aus vielen Operatoren bestehen [Korf, 1985b; Mooney, 1989; Tadepalli & Isukapalli, 1992].

9.3.4 Lernen von Plangeneralisierungen

Die Generalisierung eines vollständigen Problemlöseprozesses wird bei Plangeneralisierungsansätzen [Shavlik *et al.*, 1987; Chien, 1989; Bergmann, 1990; Kambhampati & Kedar, 1994] durchgeführt. Dies entspricht der in PARIS durchgeführten Fallgeneralisierung. Beim Problemlösen wird ein einziger generalisierter Plan gesucht, dessen Vorbedingung direkt die aktuelle Problemstellung matcht. Ist dies der Fall, ist eine weitere Suche im Basissuchraum nicht erforderlich. Ist jedoch kein generalisierter Fall anwendbar, so muß zum Erhalt der Vollständigkeit der Basissuchraum durchsucht werden. Dies führt zur folgenden Aufwandsverschiebung:

$$O(b^n) \longrightarrow O(C_M \cdot C_L\,[+b^n])$$

Spezielles Nützlichkeitsproblem

Dieser Ansatz ist insbesondere dann problematisch, wenn die generalisierten Pläne eine aus vielen Operatoren bestehende Lösung besitzen, so daß auch der Aufwand zum Test der Anwendbarkeit des generalisierten Plans stark anwächst (siehe auch Abschnitt 6.7.3).

Dies führt auch bei diesem Ansatz zu einem Nützlichkeitsproblem [Minton, 1990a; Kedar & McKusick, 1992].

9.3.5 Fallbasierte Ansätze im Vergleich

Nicht nur die bislang diskutierten erklärungsbasierten Ansätze können im Hinblick auf eine Aufwandsverschiebung beim Problemlösen charakterisiert werden, sondern auch die zuvor besprochenen fallbasierten Ansäzte PRODIGY/ANALOGY, CAPLAN/CBC, PRIAR. Während erklärungsbasierte Ansätze die zuvor gelernte Regelmenge nach anwendbaren Regeln durchsuchen, suchen die genannten fallbasierten Systeme in der Fallbasis nach einem Fall mit einer gewissen Ähnlichkeit zur Problemstellung. Hierbei ist auch die Berechnung der Ähnlichkeit mit einem gewissen Aufwand verbunden, der durchaus mit dem des Matchen einer Regel vergleichbar ist. Folglich fällt auch bei fallbasierten Systemen zunächst ein Retrievalaufwand von $C_M \cdot C_L$ an. Danach wird die im Fall enthaltene Lösung angepaßt, wobei bestimmte Problemlöseentscheidungen direkt übernommen werden können, während andere Entscheidungen durch den Problemlöser neu getroffen werden. Dies entspricht dem Aufwand zur Lösung eines kleineren Problems der Länge n', wobei n' die Anzahl der durch den Problemlöser generativ zu treffenden Entscheidungen darstellt. Insgesamt ergibt sich die folgende Aufwandsverschiebung:

$$O(b^n) \longrightarrow O(C_M \cdot C_L + b^{n'}) \text{ mit } n' < n$$

Spezielles Nützlichkeitsproblem

Auch bei den fallbasierten Systemen ist ein Nützlichkeitsproblem zu erwarten, wenn die Fallbasis stark anwächst. Jedoch haben fallbasierte Systeme den Vorteil, daß sie auch mit Fällen umgehen können, die nicht vollständig auf die Problemstellung passen: während EBL-Ansätze immer einen perfekten Match anstreben, genügen für fallbasierte Ansätze auch partielle Matches. Daher kann das Retrieval nach bestimmten Kriterien abgebrochen werden (siehe auch Abschnitt 4.3.4), und der gefundene Fall kann trotzdem nutzbringend wiederverwendet werden. Außerdem besitzen fast alle fallbasierten Systeme Indexierungsansätze, so daß die Fallbasis nicht sequenziell durchsucht werden muß. In EBL Ansätzen werden hingegen die gelernten Regeln fast ausschließlich sequenziell abgearbeitet; Indexierungsansätze werden überwiegend ignoriert (siehe z. B. [Shavlik & O'Rorke, 1993]).

9.3.6 PARIS

Auch für PARIS gelten die Aussagen, die zuvor für fallbasierte Systeme allgemein gemacht wurden. Ein Unterschied besteht jedoch aufgrund der unterschiedlichen Art der Wiederverwendung. Durch die Wiederverwendung abstrakter Fälle ist zusätzlich zum Retrievalauf-

wand der Aufwand zur Lösung der im abstrakten Fall enthaltenen Teilprobleme zu leisten. Hieraus ergibt sich die folgende Aufwandsverschiebung:

$$O(b^n) \longrightarrow O(C_M \cdot C_L + b^{n_1} + \cdots + b^{n_k}) \text{ mit } n_1 + \cdots + n_k = n$$

Spezielles Nützlichkeitsproblem

Auch hierbei wäre ein Nützlichkeitsproblem zu erwarten, wenn die Fallbasis stark anwächst. Jedoch wird dies durch die Optimierung der Wiederverwendbarkeitshierarchie verhindert. Das hierbei zugrunde liegende Kostenmodell ermöglich eine Abschätzung der Aufwände C_M, C_L und $b^{n_1} + \cdots + b^{n_k}$ und ermöglicht so eine Minimierung des Erwartungswertes der Aufwandssumme. Experimentelle Untersuchungen (vgl. Abschnitt 8.4.9) haben gezeigt, daß hierdurch eine Verlängerung der Problemlösezeit bei wachsender Fallbasis verhindert werden kann.

9.4 Einordnung in Abstraktionsansätze

Abschließend soll nun noch eine Einordnung des in PARIS verfolgten Ansatzes in andere Abstraktionsansätze gegeben werden. Wir betrachten hierzu hierarchische Problemlöser aus der Literatur. Diese Arbeiten verwenden jedoch kein Erfahrungswissen aus Fällen. Daher konzentrieren wir uns bei dieser Diskussion nur auf die Abstraktionsaspekte. Zur Strukturierung benutzen wir die in Abschnitt 3.3.2 eingeführten Begriffsbestimmungen zum hierarchischen Problemlösen. Wir erörtern wie in verschiedenen Ansätzen

- Abstraktionsebenen aufgebaut sind,
- Zustandsabstraktionen gebildet werden und
- Planverfeinerung durchgeführt wird.

9.4.1 ABSTRIPS und ABTWEAK

ABSTRIPS [Sacerdoti, 1974] ist eines der ersten Systeme, das Abstraktionsansätze beim Problemlösen verwendet. ABSTRIPS basiert wesentlich auf STRIPS [Fikes & Nilsson, 1971] und verwendet ebenfalls Strips-Operatoren sowie die Mittel-Zweck-Analyse als Suchstrategie. ABTWEAK [Yang & Tenenberg, 1990] ist eine hierarchische Variante von TWEAK [Chapman, 1987], einem Planer, der im Lösungsraum sucht. Beide Planer sind in Bezug auf die verwendeten Abstraktionsansätze sehr ähnlich und werden daher zusammen betrachtet.

Abstraktionsebenen: In beiden Systemen wird die konkrete Problemlösedomäne durch essentielle Sätze und Operatoren vorgegeben. Eine Hierarchie von Abstraktionsebenen wird

durch die Zuordnung einer natürlichen Zahl („Criticality“ genannt) zu jedem essentiellen Satz gebildet. Eine Abstraktionsebene $\mathcal{D}_i$ enhält genau die essentiellen Sätze, deren Criticality größer oder gleich i ist. Abstrakte Operatoren ergeben sich ebenfalls aus der Zuweisung der Criticalities. Jeder Operator der konkreten Problemlösedomäne kommt in jeder abstrakten Problemlösedomäne in abgewandelter Form als abstrakter Operator vor. Hierbei werden in den Operatorbeschreibungen dieser abstrakten Operatoren nur die essentiellen Sätze betrachtet, die auch auf der entsprechenden Abstraktionsebene definiert sind. Umgekehrt besitzt hierdurch jeder abstrakte Operator eine (oder mehrere) zugehörige Konkretisierung(en) in Form von konkreten Operatoren.

Zustandsabstraktion: Zustandsabstraktion geschieht dadurch, daß die essentiellen Sätze, die auf der abstrakten Ebene nicht vorkommen, weggelassen werden. Hieraus ergibt sich die Zustandsabstraktionsabbildung $\alpha : \mathcal{D}_{i-1} \rightarrow \mathcal{D}_i$ durch $\alpha(s) = s \cap \mathcal{E}_i^*$.

Planverfeinerung: Bei der Planverfeinerung wird jeder abstrakte Operator der Abstraktionsebene $\mathcal{D}_i$ durch eine Folge von Operatoren der Abstraktionsebene $\mathcal{D}_{i-1}$ verfeinert. Der Verfeinerungsprozeß wählt hierzu zunächst eine Konkretisierung des abstrakten Operators aus. Diese enthält folglich in ihrer Vorbedingung sowohl die essentiellen Sätze des abstrakten Operators, als auch ggf. zusätzliche essentielle Sätze, die auf dieser Abstraktionsebene hinzugekommen sind. Der Problemlöser versucht nun durch Einfügen eines Teilplans vor diesen Operator dessen (zusätzliche) Vorbedingungen zu erfüllen.

Vergleich mit PARIS: Der in ABSTRIPS und ABTWEAK verwendete Abstraktionsansatz ist ein Spezialfall des in PARIS realisierten Abstraktionsverfahrens. Abstraktionen, wie sie in ABSTRIPS und ABTWEAK vorkommen, können durch geeignete Wahl der abstrakten Problemlösedomäne und Abstraktionstheorie nachgebildet werden [Bergmann, 1992b]. Die Einschränkung beider Systeme besteht genau darin, daß Abstraktion nur durch Elimination von essentiellen Sätzen möglich ist. Es ist unmöglich, neue Sätze oder gar neue Operatoren auf verschiedenen Abstraktionsebenen einzuführen. Die Repräsentationssprache kann hierbei durch den Abstraktionsprozeß nicht gewechselt werden. Aufgrund dieses Defizites ist dieser Abstraktionsansatz sehr repräsentationsabhängig [Holte *et al.*, 1994; Bergmann & Wilke, 1995a]. Insbesondere in der Domäne der Arbeitsplanung und der dieser Arbeit zugrundeliegenden Repräsentation (siehe Abschnitt 8.2) war es nicht möglich, durch einen solchen Abstraktionsansatz eine Leistungssteigerung zu erzielen (siehe [Bergmann & Wilke, 1995a] und Abschnitt 8.4.8). Hierbei ist die Möglichkeit, eine neue Sprache auf verschiedenen Abstraktionsebenen einzuführen, als essentiell zu bewerten.

9.4.2 ALPINE und HIGHPOINT

ALPINE [Knoblock, 1990; Knoblock, 1994] und HIGHPOINT [Bacchus & Yang, 1994] sind zwei Ansätze, die ebenfalls in Bezug auf die verwendeten Abstraktionsverfahren sehr ähn-

lich sind zu ABSTRIPS und ABTWEAK.

Abstraktionsebenen: In ALPINE und HIGHPOINT werden Abstraktionsebenen genau wie in ABSTRIPS und ABTWEAK gebildet. Jedoch muß der Systementwickler hier die Criticalities für essentielle Sätze nicht manuell bestimmen. Diese werden durch ein spezielles Analyseverfahren direkt aus der konkreten Problemlösedomäne abgeleitet. Der Erfolg dieser Analyse ist ebenfalls stark von der Repräsentation der Problemlösedomäne abhängig. Diese hat einen starken Einfluß darauf, ob sich eine aus vielen Ebenen bestehende Hierarchie ergibt oder ob nur eine einzige Ebene (keine Abstraktion) gebildet werden kann [Knoblock, 1994].

Zustandsabstraktion und Planverfeinerung: Zustandsabstraktion und Planverfeinerung entsprechen der von ABSTRIPS und ABTWEAK.

Vergleich mit PARIS: Die für ABSTRIPS und ABTWEAK beschriebenen Einschränkungen gelten in gleicher Weise auch für ALPINE und HIGHPOINT. Was die automatische Generierung der Abstraktionshierarchien angeht, so konnten insbesondere in der Domäne der Arbeitsplanung bzw. in der in dieser Arbeit gezeigten Repräsentation davon (siehe Abschnitt 8.2), keine Abstraktionshierarchie aus mehreren Ebenen gebildet werden.

9.4.3 Graphenbasierte Verfahren

Einige in der Literatur beschriebenen Ansätze, die sich vor allem mit der Bildung von geeigneten Abstraktionsebenen beschäftigen, basieren auf der Analyse von Zustandsgraphen. Beispiele für solche Ansätze sind die Nadelöhrmengen [Mädler, 1991; Mädler, 1992] und der STAR-Algorithmus [Holte *et al.*, 1994; Holte *et al.*, 1995].

Abstraktionsebenen: Die konkrete Problemlösedomäne ist durch einen endlichen Zustandsgraphen gegeben. Hierbei sind Zustände Knoten im Graphen und Operatoren werden als Kanten vom Ausgangszustand zum Zielzustand realisiert. Diese Zustandsgraphen werden in diesen Ansätzen explizit repräsentiert und vollständig gespeichert. Verschiedene Abstraktionsebenen werden durch verschiedene Graphen repräsentiert. Die Knoten des Graphen für die Abstraktionsebene $\mathcal{D}_i$ korrespondieren dabei mit einem Teilgraphen der Abstraktionsebene $\mathcal{D}_{i-1}$.

Zustandsabstraktion: Die Zustandsabstraktion wird durch die Zusammenfassung eines Teilgraphen der Abstraktionsebene $\mathcal{D}_i$ zu einem Knoten der Abstraktionsebene $\mathcal{D}_{i-1}$ bestimmt.

Planverfeinerung: Bei der Planverfeinerung definiert jeder abstrakte Operator der Abstraktionsebene $\mathcal{D}_i$ ein neues Wegesuchproblem im Graphen der Abstraktionsebene $\mathcal{D}_{i-1}$. Hierbei wird ein Weg von dem Ausgangsknoten zu einem Knoten aus dem Teilgraphen gesucht, der mit dem abstrakten Zielknoten korrespondiert.

Vergleich mit PARIS: Der wesentliche Unterschied zwischen diesen Ansätzen und dem in dieser Arbeit verfolgten Ansatz besteht in der expliziten Graphrepräsentation. In PARIS wird der Zustandsgraph weder explizit repräsentiert, noch wird seine Endlichkeit vorausgesetzt. Die in Abschnitt 8.2 vorgestellte Arbeitsplanungsdomäne repräsentiert einen unendlichen Zustandsgraphen, so daß die graphenbasierten Verfahren hierfür nicht eingesetzt werden können. Die Einschränkung auf endliche Zustandsgraphen bietet jedoch auch einen wichtigen Vorteil. So ist es möglich, Abstraktionsebenen automatisch zu generieren, ohne dabei das Problem der Repräsentationsabhängigkeit (wie z. B. bei ALPINE und HIGHPOINT) zu bekommen. In PARIS müssen die Abstraktionsebenen manuell vorgegeben werden. Das Erlernen von geeigneten Abstraktionsebenen erscheint, aufgrund der hohen Vielfalt der darstellbaren Abstraktionen, nicht möglich. Denkbar sind jedoch Verfahren, die eine bestehende abstrakte Problemlösedomäne bei Bedarf um neue Operatoren erweitert.

Kapitel 10

Zusammenfassung, Bewertung und Ausblick

In dieser Arbeit wurden Methoden entwickelt und analysiert, die die Wiederverwendung von Erfahrungswissen aus Fällen für Problemlöseaufgaben ermöglichen. Hierdurch sollte das Problem der durch große Suchräume bedingten langen Rechenzeiten überwunden werden, welches bei generischen Ansätzen oft zu einem unüberwindbaren Hindernis geführt hat. Die Hauptziele dieser Arbeit bestanden darin, einen *domänenunabhängigen* Ansatz zu entwickeln, der Fälle auf *flexible* Weise anpaßt, die *Korrektheit* der entstandenen Lösung garantiert und insgesamt *leistungsfähiges* Problemlösen realisiert. Darüber hinaus sollten vorliegende Fälle *automatisch* verarbeitet werden, und ein *inkrementelles Erweitern* der Fallbasis sollte möglich sein.

10.1 Zusammenfassung

In dieser Arbeit wurde ein neuer Ansatz vorgestellt, der die Wiederverwendung abstrakter Fälle (siehe Kapitel 5) und die Wiederverwendung generalisierter Fälle (siehe Kapitel 6) in den Vordergrund stellt und diese beiden Verfahren integriert. Hierdurch wird das fallbasierte Schließen um Techniken des erklärungsbasierten Lernens und der Abstraktion erweitert. Dieser Ansatz ermöglichte es erstmals, Fälle auf verschiedenen Abstraktionsebenen auf flexible Weise wiederzuverwenden und dabei vorliegendes abstraktes Wissen zu berücksichtigen. Seine Tragfähigkeit wurde durch Beispiele aus einem realen Anwendungsbereich (Arbeitsplanung), sowie durch umfangreiche Experimente im strengen Sinne der Statistik belegt (siehe Kapitel 8).

Wiederverwendung abstrakter Fälle

Zur Wiederverwendung abstrakter Fälle werden die vorliegenden konkreten Fälle durch die Fallabstraktion in eine Menge abstrakter Fälle auf verschiedenen Abstraktionsebenen transformiert. Dieser Transformationsprozeß benutzt Domänenwissen über abstrakte Zusammenhänge in einer Domäne (abstrakte Problemlösedomäne). Zusätzlich beschreibt eine generische Abstraktionstheorie den Übergang zwischen verschiedenen Abstraktionsebenen. Eine wichtige Eigenschaft des entwickelten Verfahrens zur Fallabstraktion ist, daß es den Wechsel der Repräsentationssprache zwischen verschiedenen Abstraktionsebenen erlaubt. Dies hat sich als wesentlich für die Leistungfähigkeit der Wiederverwendung abstrakter Lösungen herausgestellt (siehe Abschnitt 8.4.8 sowie [Bergmann & Wilke, 1995a]).

Falls ein abstrakter Fall zur Lösung eines neuen Problems wiederverwendbar ist, so wird die in diesem Fall enthaltene abstrakte Lösung mit Hilfe eines generischen Problemlösers zu einer vollständigen konkreten Lösung verfeinert. Hierbei dient die abstrakte Problemlösung als Dekomposition des aktuellen Problemes in mehrere kleinere Teilprobleme. Diese Teilprobleme können von einem Problemlöser wesentlich schneller bearbeitet werden als das gesamte Problem. Entscheidend für die effiziente Lösungsverfeinerung ist die Formulierung der abstrakten Problemlösedomäne. Hierbei hat sich gezeigt, daß es wichtig ist, daß die abstrakten Operatoren unabhängig voneinander sind und eine geringe Teilzieldistanz aufweisen (siehe Abschnitte 8.1 und 8.4.10). Für diesen Fall konnte in Experimenten im Mittel eine signifikante Effizienz und Kompetenzverbesserungen des Problemlösens durch Wiederverwendung abstrakter Fälle verglichen mit einem generischen Problemlöser, nachgewiesen werden (siehe Abschnitte 8.4.6 und 8.4.7).

Wiederverwendung generalisierter Fälle

Für die Wiederverwendung generalisierter Fälle wird die Technik des erklärungsbasierten Generalisierens eingesetzt, um einen generalisierten Fall aus einem speziellen Fall zu erzeugen. Dabei wird eine *Erklärung* aufgebaut – dies ist ein Beweis auf der Grundlage der Operatoren und Regeln der Problemlösedomäne –, die nachweist, daß die im Fall vorliegende Lösung tatsächlich eine Lösung für das im Fall vorkommende Problem ist. Auf der Basis dieser Erklärung wird dann in einem *Generalisierungschritt* eine Problemklasse und eine Lösungsklasse bestimmt. Für diesen Generalisierungsschritt konnte nachgewiesen werden, daß er nur korrekte Fallgeneralisierungen bestimmt.

Um festzustellen, ob ein neues Problem durch Wiederverwendung eines generalisierten Falles gelöst werden kann, muß überprüft werden, ob das neue Problem in die Problemklasse des generalisierten Falls fällt. Die im Fall enthaltene Lösungsklasse muß dann zu einer Lösung spezialisiert werden. Dies erfordert jedoch im Worst-Case eine exponentielle Zeitkomplexität. In Experimenten hat sich herausgestellt, daß dieser Zeitaufwand größer sein kann als der Zeitaufwand, den ein rein generischer Problemlöser erfordert. Daher konnte durch diese Methode keine signifikante Effizienz und Kompetenzverbesserungen, vergli-

chen mit einem generischen Problemlöser, nachgewiesen werden (siehe Abschnitte 8.4.6 und 8.4.7).

Wiederverwendung generalisierter abstrakter Fälle

Die Kombination der beiden zuvor erörterten Methoden ist auf einfache Weise möglich. Hierzu werden vorliegende konkrete Fälle zunächst zu abstrakten Fällen abstrahiert; diese werden danach mit Hilfe der Fallgeneralisierung generalisiert. Als Ergebnis entstehen generalisierte abstrakte Fälle. Bei der Problemlösung wird die Lösungsklasse des ausgewählten generalisierten abstrakten Falls zunächst spezialisiert, und die sich daraus ergebende abstrakte Lösung wird anschließend zu einer konkreten Lösung verfeinert. In Experimenten hat sich herausgestellt, daß dieser kombinierte Wiederverwendungsansatz, ebenfalls im Mittel, eine signifikante Effizienz und Kompetenzverbesserungen des Problemlösens, verglichen mit einem generischen Problemlöser, bewirkt (siehe Abschnitte 8.4.6 und 8.4.7). Auch im Vergleich mit der Wiederverwendung abstrakter Fälle konnten für den kombinierten Ansatz zusätzliche Vorteile im Hinblick auf höhere Effizienz und Kompetenz nachgewiesen werden (siehe ebenfalls Abschnitte 8.4.6 und 8.4.7). Auch läßt sich durch die Wiederverwendung generalisierter abstrakter Fälle eine höhere Flexibilität erreichen, als durch die Wiederverwendung rein generalisierter oder rein abstrakter Fälle (siehe Abschnitt 8.4.5). Auffallend bei diesen Ergebnissen war, daß die Fallgeneralisierung alleine nicht zu einem verbesserten Problemlöseverhalten geführt hat. In Verbindung mit der Fallabstraktion kann sie jedoch eine weitere Verbesserung bewirken. Dies erklärt sich dadurch, daß der Aufwand zur Lösungsspezialisierung stark von der Größe der Problembeschreibung abhängt. Bei generalisierten abstrakten Fällen ist diese Problembeschreibung jedoch auf einer höheren Abstraktionsebene angesiedelt und enthält folglich weniger Details. Daher ist der Aufwand zur Lösungsspezialisierung drastisch reduziert, so daß sich die Wiederverwendung auszahlt.

Organisation der Fallbasis

Zur Organisation der Fallbasis wurde ein Ansatz entwickelt, der es erlaubt generalisierte abstrakte Fälle in einer Wiederverwendbarkeitshierarchie (das ist ein n-ärer Baum an dessen Knoten generalisierte abstrakte Fälle stehen) anzuordnen. In dieser Wiederverwendbarkeitshierarchie ist ein generalisierter abstrakter Fall an einem Knoten für eine größere Menge konkreter Probleme wiederverwendbar als ein generalisierter abstrakter Fall an den Nachfolgerknoten. Der wesentliche Vorteil, den die Wiederverwendbarkeitshierarchie bringt, liegt darin, daß sie es erlaubt, die Menge der zu durchsuchenden Fälle stark einzuschränken. Ist ein generalisierter abstrakter Fall an einem Knoten der Hierarchie als nicht wiederverwendbar nachgewiesen, so folgt automatisch, daß alle generalisierten abstrakten Fälle am darunter befindlichen Teilbaum ebenfalls nicht wiederverwendbar sind. In Experimenten (siehe Abschnitt 8.4.9) hat sich ein deutlicher Vorteil dieses Retrievalverfahrens im Vergleich zu einem naiven sequenziellen Retrievalansatz gezeigt. Die Effizienz und die

Kompetenz hat sich im Mittel durch die Wiederverwendbarkeitshierarchie signifikant gesteigert.

Eine weitere Verbesserung des Retrievals konnte durch die Einführung eines Kostenmodells erreicht werden. Dieses Kostenmodell bestimmt den Erwartungswert der Problemlösezeit bei einer gegebenen Fallbasis von generalisierten abstrakten Fällen und einer Wiederverwendbarkeitshierarchie. Hierbei wird sowohl der Retrievalaufwand als auch der Aufwand zur Lösungsspezialisierung und -verfeinerung berücksichtigt. Unter Verwendung dieses Kostenmodells konnte ein Optimierungsansatz entwickelt werden, der es erlaubt, das Problemlöseverhalten durch Umsortieren von Geschwisterknoten der Wiederverwendbarkeitshierarchie, sowie durch Ausblenden von weniger nützlichen generalisierten abstrakten Fällen weiter zu verbessern. In Experimenten (siehe ebenfalls Abschnitt 8.4.9) konnte für die Wiederverwendung generalisierter abstrakter Fälle im Mittel eine signifikante Steigerung der Effizienz und Kompetenz im Vergleich zur einfachen Wiederverwendbarkeitshierarchie gezeigt werden.

10.2 Gesamtbewertung

10.2.1 Einsatzvoraussetzungen

Damit die beschriebenen Methoden eingesetzt werden können, müssen eine Reihe von Voraussetzungen erfüllt sein. Diese wurden bereits in Abschnitt 1.1.4 genannt; sie sollen aber an dieser Stelle nochmals rekapituliert und in Bezug auf ihre Erfüllbarkeit bewertet werden. Zunächst betrachten wir die allgemeinen Voraussetzungen.

1. Voraussetzung: Vorhandensein von Fällen

Als erste Voraussetzung ist das Vorhandensein von Fällen zu nennen. Dies ist natürlich die Grundvoraussetzung für den Einsatz eines fallbasierten Verfahrens. Fälle entstehen oft als Nebenprodukt einer vom Menschen durchgeführten Problemlösung und werden meist aus Dokumentationsgründen aufbewahrt. Daher ist diese Voraussetzung in vielen Bereichen leicht zu erfüllen. Liegen Fälle, die frühere Problemlösungen beschreiben, jedoch nicht oder nicht vollständig vor, so kann man versuchen, neue Fälle zu akquirieren. Hierzu muß eine Möglichkeit zur Fallaufzeichnung geschaffen werden und in den im Problembereich etablierten Problemlösevorgang integriert werden. Eine weitere Alternative besteht darin, daß Experten typische Fälle, die nicht unbedingt genau so vorgekommen sein müssen, konstruieren. Beide Alternativen führen jedoch zu einem erhöhten Entwicklungsaufwand.

2. Voraussetzung: Ähnlichkeit der zukünftigen Probleme

Eine weitere allgemeine Voraussetzung betrifft die Kenntnis des Spektrums der erwarteten zukünftigen Probleme, sowie deren Ähnlichkeit zu den vorhandenen Fällen. Wir müssen voraussetzen, daß zukünftige Probleme ähnlich sind zu den verfügbaren Fällen. Ein wesentlicher Vorteil der vorgestellten Methoden ist ihre Flexibilität, also ihre Fähigkeit Fälle selbst dann wiederzuverwenden, wenn Problemstellungen stärker abweichen. Zwar muß bei einer stärkeren Abweichung von Problemstellung und Fall eine längere Problemlösezeit in Kauf genommen werden, was jedoch in gewissen Grenzen noch akzeptabel sein kann. Da sich in vielen Problembereichen die zu lösenden Probleme nicht schlagartig ändern, ist die Annahme begründet, daß die in der Zukunft zu lösenden Probleme nicht stark von den bisher gelösten Problemen abweichen. Daher kann diese Voraussetzung in der Regel erfüllt werden.

3. Voraussetzung: Änderungen im Problemspektrum

Die dritte allgemeine Voraussetzung besagt, daß bei großen Änderungen im Spektrum der zu lösenden Probleme ein Bereichsexperte in der Lage sein sollte, neuartige Probleme ohne Systemunterstützung lösen zu können. Diese Voraussetzung dürfte im allgemeinen keine Probleme aufwerfen. Jedoch folgt hieraus auch, daß man bei fallverarbeitenden Ansätzen nicht vollständig auf menschliche Experten verzichten kann, wenn die Systeme in Bereichen eingesetzt werden sollen, in denen stärker variierende Problemstellungen vorkommen können.

Neben diesen allgemeinen Voraussetzungen ergeben sich auch weitere technische Voraussetzungen, die durch die speziellen Wiederverwendungsmethoden bedingt sind.

4. Voraussetzung: Repräsentierbarkeit von Fällen

Eine weitere technische Voraussetzung besteht darin, daß die im Problembereich vorliegenden Fälle als Fälle im Sinne von Definition 3.1 repräsentiert sein müssen. Um dies zu erreichen, ist ggf. eine Transformation der vorliegenden Fallrepräsentation (z. B. CAD-Modell für die Arbeitsplanung) erforderlich, die je nach Anwendungsbereich mit mehr oder weniger großem Aufwand verbunden sein kann. Generell kann man diese Voraussetzung jedoch als erfüllbar betrachten.

5. Voraussetzung: Modellierbarkeit als Aktionsplanungsaufgabe

Hier ist die zentrale Voraussetzung, daß der vorliegende Problembereich als Aktionsplanungsaufgabe modelliert und in der in Kapitel 2 beschriebenen Weise als Problemlöse-

domäne formalisiert werden kann. Hierbei wird vorausgesetzt (siehe Abschnitt 2.4), daß die im Gegenstandsbereich zu betrachtenden Welten und Aktionen vollständig bekannt sind. Der Gegenstandsbereich darf sich selbst nicht verändern, d. h. die Definition der Aktionen muß konstant bleiben. Außerdem müssen die Aktionen diskrete deterministische Weltveränderungen beschreiben, die nur sequentiell ausgeführt werden können. Diese Voraussetzungen stellen natürlich eine Einschränkung in der Anwendbarkeit der Methoden dar. Diese entsprechen jedoch den üblichen Einschränkungen, die in der KI-Planung generell gemacht werden. Trotz dieser Einschränkungen gibt es genügend relevante Problembereiche, wie z. B. die Arbeitsplanung, für die diese Voraussetzungen erfüllt werden können.

6. Voraussetzung: Formulierbarkeit von Abstraktionsebenen

Diese Voraussetzung besagt, daß man den vorliegenden Problembereich auf verschiedenen Abstraktionsebenen beschreiben muß. Hierzu ist es erforderlich, neben der konkreten Problemlösedomäne eine abstrakte Problemlösedomäne sowie eine generische Abstraktionstheorie zu entwickeln. Wie in Abschnitt 8.1 herausgearbeitet wurde, müssen weitere Anforderungen (siehe Kriterien 8.1 bis 8.5) an deren Formulierung gestellt werden. In der entwickelten Beispielanwendung konnten diese Anforderungen erfüllt werden (siehe Abschnitt 8.2.3). Im allgemeinen muß durch geeignete Wissensakquisitionsverfahren solches Domänenwissen akquiriert und repräsentiert werden.

Zusammenfassung

Zusammenfassend läßt sich sagen, daß die gemachten Voraussetzungen in vielen Problembereichen erfüllt werden können. Dies ist jedoch im Einzelfall mit einem gewissen Aufwand verbunden, der bei den Entwicklungskosten für eine Anwendung berücksicht werden muß. Dieser Aufwand ist jedoch geringer als der Aufwand zur Entwicklung eines generischen wissensbasierten Problemlösers. Auch bei einem generischen wissensbasierten Problemlöser ist es erforderlich, den Problembereich z. B. als Aktionsplanungsaufgabe zu modellieren. Bei einem hierarchischen Problemlöser ist ebenfalls eine hierarchische Modellierung erforderlich. In Bezug auf die Voraussetzungen 5 und 6 sind generische und fallbasierte Ansätze also vergleichbar. Der deutliche Aufwandsvorteil für fallbasierte Ansätze – und so auch für PARIS – besteht darin, daß kein Kontrollwissen akquiriert werden muß sondern lediglich Fälle zur Verfügung gestellt werden müssen (Voraussetzungen 1 bis 4).

10.2.2 Diskussion der Anforderungen

Zu Beginn dieser Arbeit wurden ein Reihe von Anforderungen formuliert (siehe Abschnitt 1.1.2), die für den praktischen Einsatz von fallbasierten Problemlösern von zentraler Bedeutung sind. Wichtig ist, daß alle Anforderungen gleichzeitig erfüllt werden können. Der

Verzicht auf eine dieser Anforderungen macht in der Regel den Einsatz eines fallbasierten Problemlösers unmöglich. Jedoch wird dies bislang durch keinen Ansatz geleistet. In dieser Arbeit konnten hingegen erstmals alle diese Anforderungen wie folgt erfüllt werden:

Höhere Leistungsfähigkeit: Die zentrale Anforderung bezieht sich auf die Leistungsfähigkeit von Problemlösen durch Wiederverwendung. In einer Reihe von empirischen Untersuchungen konnte ein deutlicher Effizienz- und Kompetenzvorteil der Wiederverwendung im Vergleich zu generischen Ansätzen gezeigt werden (siehe Abschnitte 8.4.6 und 8.4.7). Die Effizienzverbesserungen liegen hierbei in einem Bereich, der von praktischer Relevanz ist. So konnte z. B. für die in den Experimenten[1] untersuchten komplexeren Probleme im Mittel eine Verkürzung der Problemlösezeit von mehr als zwei Tagen (extrapolierter Wert für den generischen Problemlöser) auf 77 Sekunden (gemessener Wert bei Wiederverwendung von generalisierten abstrakten Fällen) erreicht werden. Außerdem konnte der Prozentsatz der gelösten Problem von 29 % (generischen Problemlöser) auf 87 % (bei Wiederverwendung von generalisierten abstrakten Fällen) gesteigert werden, wobei der generische Problemlöser lediglich die einfachsten Problem lösen konnte.

Korrektheit: Die Korrektheitsanforderung besagt, daß die Wiederverwendung von Fällen zu korrekten Lösungen führen muß. Die entwickelten Methoden besitzen diese Eigenschaft, die formal nachgewiesen werden konnte (siehe Sätze 5.6 und 6.2).

Flexibilität: Die Flexibilitätsanforderung besagt, daß ein vorliegender Fall für möglichst viele neue Probleme eine Leistungssteigerung bewirken soll. Die Flexibilität wurde durch eine empirische Untersuchung ermittelt. Hierbei konnte gezeigt werden, daß die Wiederverwendung generalisierter abstraker Fälle flexibler ist als die Wiederverwendung rein generalisierter bzw. rein abstrakter Fälle. Bei der Wiederverwendung generalisierter abstrakter Fälle konnte im Experiment im Mittel jeder der 100 unterschiedlichen, zufällig gewählten Fälle zur Lösung von 44 der 100 darin enthaltenen Probleme effizient wiederverwendet werden.

Automatische Fallverarbeitung: Die Anforderung, daß die vorliegenden Fälle vollautomatisch wiederverwendet werden, konnte ebenfalls erfüllt werden. Die entwickelten Verfahren erfordern keine Aufzeichnung des Lösungswegs als Bestandteil des Falls. Dieser muß daher auch nicht (z. B. durch einen menschlichen Experten) rekonstruiert werden. Außerdem arbeiten die Verfahren zur Fallabstraktion und -generalisierung vollautomatisch. Ebenso die Fallauswahl, sowie die Lösungsspezialisierung und -verfeinerung.

Inkrementelle Erweiterbarkeit: Die Anforderung, daß das System inkrementell um neue Fälle erweitert werden kann, konnte ebenfalls erfüllt werden. Dies ist dadurch möglich, daß die Wiederverwendbarkeitshierarchie, die als Zugriffsstruktur für die Fälle dient, inkrementell erstellt wird.

Domänenunabhängigkeit: Auch die Anforderung an die Domänenunabhängigkeit der Methoden konnte erfüllt werden. Das domänenspezifische Wissen ist vollständig in der

[1] Bei diesen Experimenten wurden zum Test nur Probleme verwendet, die in den Fällen der Fallbasis noch nicht vorhanden waren.

konkreten und der abstrakten Problemlösedomäne, sowie in der generischen Abstraktionstheorie repräsentiert. Die vorgestellten Verfahren verwenden dieses Wissen, sind aber selbst domänenunabhängig.

Zusammenfassend läßt sich sagen, daß in dieser Arbeit die Grundlagen geschaffen wurden, die für die Entwicklung von praxistauglichen fallbasierten Planungssystemen von zentraler Bedeutung sind. Dies ist nicht zuletzt auf die erzielte Effizienzsteigerung und Flexibilität zurückzuführen. Obwohl natürlich keine Reduktion der Komplexität des Problemlösens erreicht werden konnte, wurde jedoch eine für die praktische Einsetzbarkeit wichtige Schwelle überwunden.

10.2.3 Grenzen dieser Arbeit

Einige Fragen konnten im Rahmen dieser Arbeit nicht behandelt werden, bzw. haben sich aufgrund dieser Arbeit neu gestellt.

Wie bereits erwähnt, war die Qualität oder Optimalität von Lösungen nicht Thema dieser Arbeit. Zwar besteht bei fallbasierten Methoden allgemein die Hoffnung, daß gute Lösungen entstehen, wenn Fälle wiederverwendet werden, die gute Lösungen enthalten. Diese Hypothese wurde jedoch im Rahmen dieser Arbeit nicht untersucht.

In der Arbeit wurde die vollautomatische Wiederverwendung von Fällen in den Vordergrund gestellt. Es gibt jedoch auch Anwendungen, in denen eine Benutzerinteraktion bei der Problemlösung aus unterschiedlichen Gründen wünschenswert ist. Hierzu liefert diese Arbeit keinen Beitrag.

Eine Frage, die sich auch durch diese Arbeit erneut stellt, betrifft die Akquisition von Domänenwissen – also von Operatoren – zum fallbasierten Problemlösen. Allgemeine Wissensakquisitionsmethoden und Werkzeuge, wie z. B. CoMo-Kit [Maurer, 1993], MeMo-Kit [Neubert, 1993] oder Mikado-Kit [Schmidt, 1994], bieten hier zwar einen Ausgangspunkt, jedoch ist zu erwarten, daß der Akquisitionsaufwand durch Techniken, die speziell für fallbasierte Ansätze zugeschnitten sind, weiter reduziert werden kann. Diese Problematik wurde jedoch im Rahmen dieser Arbeit nicht angegangen.

Eine weitere Fragestellung, die ebenfalls in dieser Arbeit ausgeklammert werden mußte, ist die der Integration in ein industrielles Umfeld (z. B. Integration mit CAD/CAM). Dies ist jedoch für den praktischen Einsatz eines solchen Systems von hoher Bedeutung. So konnte auch in dieser Arbeit keine im Einsatz befindliche Anwendung erstellt werden; für Aktionsplanungsaufgaben existieren bis zum heutigen Zeitpunkt weltweit keine im Einsatz befindlichen Anwendungen fallbasierter Systeme.

10.3 Ausblick

Aus den zuvor beschriebenen Grenzen dieser Arbeit ergeben sich bereits eine Reihe von Ansatzpunkten für weitere Forschungsaktivitäten.

Weiterentwicklung zur Einsatzreife

An erster Stelle ist hierbei die Weiterentwicklung bis zur Einsatzreife, also bis zum ersten im tatsächlichen Einsatz befindlichen fallbasierten System für die Aktionsplanung zu sehen. Dies erfordert eine eingehende Beschäftigung mit Fragen einer umfassenden Akquisition von Domänenwissen, Fragen der Benutzerinteraktion, sowie dem Problem der Integration in ein industrielles Umfeld. Diese Fragen sind nicht nur für das in dieser Arbeit vorgestellte PARIS System von Bedeutung sondern für alle zur Zeit im Bereich der Forschung entwickelten Systeme wie z. B. CAPLAN/CBC, Déjà Vu, PRIAR oder PRODIGY/ANALOG (siehe Kapitel 9).

Erweiterung auf Konfigurations- und Designaufgaben

Weitere zukünftige Arbeiten sollten sich auch mit der Erweiterung des hier vorgestellten Ansatzes zur Wiederverwendung generalisierter und abstrakter Fälle auf andere Problemlösebereiche beschäftigen. Naheliegend sind andere synthetische Problemstellung, wie z. B. Konfiguration oder Design. Auch in diesen Bereichen sind diese Wiederverwendungstechniken erfolgsversprechend. Andere fallbasierte Techniken sind hier bereits erfolgreich angewendet worden [Pfitzner, 1993; Hua *et al.*, 1994; Napoli & Lieber, 1993; Fabel, 1993]. Ein wesentlicher Unterschied zu der in dieser Arbeit behandelten Aktionsplanung besteht in der Art der Wissensrepräsentation und den generischen Problemlösetechniken für Konfigurations- und Planungsaufgaben. Hierbei sind insbesondere Techniken zur Repräsentation und Verarbeitung von Constraints zu nennen.

Fallbasierte modellbasierte Diagnose

Die in dieser Arbeit entwickelten Techniken können vermutlich auch auf diagnostische Fragestellungen angepaßt werden. Hierbei ist insbesondere an modellbasierte Diagnose gedacht, da auch diese mit dem Problem der großen Suchräume zu tun hat. Erste Ansätze hierzu wurden bereits im System MoCAS/2 [Pews & Wess, 1993; Bergmann *et al.*, 1994a; Pews, 1994] entwickelt. Diese sind jedoch noch nicht umfassend untersucht worden.

Literaturverzeichnis

Aamodt, A. 1991. *A Knowledge-Intensive, Integrated Approach to Problem Solving and Sustained Learning.* Dissertation, University of Trondheim.

Aamodt, A. 1994. Explanation-driven case-based reasoning. *Seiten 274–288 aus:* Wess, S., Althoff, K.-D. und Richter, M. M. (Hrsg.), *Topics in Case-Based Reasoning.* Springer.

Aamodt, A. und Plaza, E. 1994. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications*, **7**(1), 39–59.

Aha, D., Kibler, D. und Albert, M.K. 1991. Instance-Based Learning Algorithms. *Machine Learning*, **6**, 37–66. March 1991.

Althoff, K.-D. und Wess, S. 1992. Case-based reasoning and expert system development. *Aus:* Schmalhofer, F., Strube, G. und Wetter, Th. (Hrsg.), *Contemporary Knowledge Engineering amd Cognition.* Heidelberg: Springer.

Althoff, K.-D., Wess, S., Bartsch-Spörl, B. und Janetzko, D. (Hrsg.). 1992. *Workshop: Ähnlichkeit von Fällen beim fallbasierten Schließen.* SEKI-Report.

Althoff, K.-D., Auriol, E., Bergmann, R., Breen, S., Dittrich, S., Johnston, R., Manago, M., Traphöner, R. und Wess, S. 1995. Case-based reasoning for decision support and diagnostic problem solving: The INRECA approach. *Seiten 63–72 aus:* Bartsch-Spörl, B., Janetzko, D. und Wess, S. (Hrsg.), *Fallbasiertes Schließen - Grundlagen und Anwendungen.* Zentrum für Lernende Systeme und Anwendungen, Universität Kaiserslautern, Report LSA-95-02.

Anderson, J. R. 1983. *The architecture of cognition.* Cambridge, MA: Harvard University Press.

Anderson, J. R. 1989. A theory of the origins of human knowledge. *Artificial Intelligence*, **40**, 313–351.

Apt, K.R. und Emden, M.H. 1982. Contributions to the theory of logic programming. *JACM*, **29**(3), 841–862.

ASHLEY, K. D. 1990. *Modeling Legal Argument: Reasoning with Case and Hypotheticals.* Cambridge: MIT-Press.

BACCHUS, F. UND YANG, Q. 1991. The downward refinement property. *Seiten 286–292 aus: Proceedings of the International Conference on AI.*

BACCHUS, F. UND YANG, Q. 1994. Downward refinement and efficiency of hierarchical problem solving. *Artificial Intelligence*, **71**, 43–100.

BAREISS, R. 1989. *Exemplar-Based Knowledge Acquisition: A unified Approach to Concept Representation, Classification and Learning.* Academic Press.

BAREISS, R. (Hrsg). 1991. *Proceedings: Case-Based Reasoning Workshop.* San Mateo, California: Morgan Kaufmann Publishers, DARPA. Washington, D.C., USA, May 8–10, 1991.

BARLETTA, R. UND MARK, W. 1988. Explanation-Based Indexing of Cases. *Seiten 50–61 aus: Proceedings of the 7th Annual Conference on Artificial Intelligence AAAI-88.* Morgan Kaufmann.

BARRETT, A. UND WELD, D. 1994. Partial order planning: Evaluating possible efficiency gains. *Artificial Intelligence*, **67**.

BENTO, C. UND COSTA, E. 1994. A similarity metric for retrieval of cases imperfectly explained. *Seiten 92–105 aus:* WESS, S., ALTHOFF, K.-D. UND RICHTER, M. M. (Hrsg.), *Topics in Case-Based Reasoning.* Springer.

BERGADANO, F. UND GIORDANA, A. 1990. Guiding instruction with domain theories. *Kap. 17, Seiten 474–492 aus:* KODRATOFF, Y. UND MICHALSKI, R. S. (Hrsg.), *Machine learning: An artificial intelligence approach*, Band 3. San Mateo, CA: Morgan Kaufmann.

BERGMANN, R. 1990. *Generierung von Skelettplänen als Problem der Wissensakquisition.* Diplomarbeit, Universität Kaiserslautern, Erwin-Schrödinger-Straße, Postfach 3049, W-6750 Kaiserslautern, Germany.

BERGMANN, R. 1992a. Knowledge acquisition by generating skeletal plans. *Seiten 125–133 aus:* SCHMALHOFER, F., STRUBE, G. UND WETTER, TH. (Hrsg.), *Contemporary Knowledge Engineering and Cognition.* Heidelberg: Springer.

BERGMANN, R. 1992b. Learning Plan abstractions. *Seiten 187–198 aus:* OHLBACH, H.J. (Hrsg), *GWAI-92 16th German Workshop on Artificial Intelligence.* Springer Lecture Notes on AI, Band 671.

BERGMANN, R. 1993. Integrating abstraction, explanation-based learning from multiple examples and hierarchical clustering with a performance component for planning. *Aus:* PLAZA, ENRIC (Hrsg), *Proceedings of the ECML-93 Workshop on Integrated Learning Architectures (ILA-93).*

BERGMANN, R. 1995. Plan abstraction with change of representation language. *Aus: Proceedings of the Symposium on Abstraction, Reformulation and Approximation, SARA-95.*

BERGMANN, R. UND DANNENMANN, A. 1992. Abstrakte Fallrepräsentation für Abruf und Adaption von Planungsfällen auf verschiedenen Abstraktionsniveaus. *Aus:* [Althoff *et al.*, 1992].

BERGMANN, R. UND WILKE, W. 1995a. Building and refining abstract planning cases by change of representation language. *Journal of Artificial Intelligence Research*, **3**, 53–118.

BERGMANN, R. UND WILKE, W. 1995b. Flexible reuse of plans by abstraction and refinement. *Aus: IJCAI95-Workshop: Formal Approaches to the Reuse of Plans, Proofs, and Programs.*

BERGMANN, R. UND WILKE, W. 1995c. Learning Abstract Planning Cases. *Seiten 55–76 aus:* LAVRAC, N. UND WROBEL, S. (Hrsg.), *Machine Learning: ECML-95.* Lecture Notes in Artificial Intelligence.

BERGMANN, R., PEWS, G. UND WILKE, W. 1994a. Explanation-based similarity: A unifying approach for integrating Domain Knowledge into case-based reasoning. *Seiten 182–196 aus:* WESS, S., ALTHOFF, K.-D. UND RICHTER, M.M. (Hrsg.), *Topics in Case-Based Reasoning.* Lecture Notes on Artificial Intelligence, Band 837. Springer.

BERGMANN, R., WESS, S., TRAPHÖNER, R. UND BREAN, S. 1994b. *Using Background Knowledge in the Integrated System.* Bericht D29. Inreca Consortium.

BERGMANN, R., MUÑOZ, H. UND VELOSO, M. 1996. Fallbasiertes Planen: Ausgewählte Methoden und Systeme. *Künstliche Intelligenz*, **1/96**.

BHANSALI, S. UND HARANDI, M.T. 1993. Synthesis of UNIX programs using derivational analogy. *Machine Learning*, **10**, 7–55.

BLYTHE, J., ETZIONI, O. UND THE PRODIGY GROUP. 1992. *Prodigy 4.0: The Manual and Tutorial.* Bericht CMU-CS-92-150. Carnegie Mellon University, Pittsburgh, PA.

BÖRNER, K. 1994. Toward Formalizations in Case-Based Reasoning for Synthesis. *Aus:* AHA, DAVID L. (Hrsg), *Proceedings AAAI-94 Case-Based Reasoning Workshop.*

BRANTING, K. 1991. Exploiting the Complemantary of Rules and Precedents with Reciprocity and Fairness. *Aus:* [Bareiss, 1991]. Washington, D.C., USA, May 8–10, 1991.

BRANTING, K. UND HASTINGS, J. D. 1994. An empirical evaluation of model-based case matching and adaptation. *Seiten 72–78 aus:* AHA, DAVID (Hrsg), *AAAI-94 Workshop on Case-Based Reasoning.* American Association for Artificial Intelligence.

Braverman, M. S. und Russel, S. J. 1988. Boundaries of operationality. *Seiten 221–234 aus:* Laird, J. (Hrsg), *Proceedings of the Fifth International Conference on Machine Learning.* San Mateo, CA: Morgan Kaufmann.

Burstein, M. H. 1989. Analogy vs. CBR: The Purpose of Mapping. *Seiten 13–136 aus: Proceedings of the Case-based Reasoning Workshop.* Morgan Kaufmann.

Bylander, T. 1991. Complexity results for planning. *Seiten 274–279 aus: Proceedings of the International Conference on Artificial Intelligence.* Morgan Kaufmann.

Bylander, T. 1992. Complexity results for extended planning. *Seiten 20–27 aus: Proceedings of the International Conference on AI Planning Systems.* Morgan Kaufmann.

Cain, T., Pazzani, M. J. und Silverstein, G. 1991. Using Domain Knowledge to Influence Similarity Judgements. *Aus:* [Bareiss, 1991]. Washington, D.C., USA, May 8–10, 1991.

Carbonell, J. G. 1986. Derivational analogy: A theory of reconstructive problem solving and expertise aquisition. *Kap. 14, Seiten 371–392 aus:* Michalski, R. S., Carbonell, J. G. und Mitchell, T. M. (Hrsg.), *Machine learning: An artificial intelligence approach*, Band 2. Los Altos, CA: Morgan Kaufmann.

Carbonell, J. G., Knoblock, C. A. und Minton, S. 1991. Prodigy: An integrated architecture for planning and learning. *Seiten 241–278 aus:* VanLehn, K. (Hrsg), *Architectures for Intelligence.* Lawrence Erlbaum Associates, Publishers.

Chapman, D. 1987. Planning for conjunctive goals. *Artificial Intelligence*, **32**, 333–377.

Chien, S. A. 1989. Using and refining simplifications: Explanation-based learning of plans in intractable domains. *Seiten 590–595 aus: Proceedings of the International Joint Conference on Artificial Intelligence-89*, Band 1. Detroit, MI: Morgan Kaufmann.

Cunningham, P. und Smyth, B. 1994. A comparison of model-based and incremental case-based approaches to electronic fault diagnosis. *Aus:* Aha, David (Hrsg), *AAAI-94 Workshop on Case-Based Reasoning.* American Association for Artificial Intelligence.

Cunningham, P., Finn, D. und Slattery, S. 1994. Knowledge Engineering Requirements in Derivational Analogy. *Aus:* [Wess *et al.*, 1994a].

DeJong, G. (Hrsg). 1993. *Investigating explanation-based learning.* Kluwer Academic Publishers.

DeJong, G. und Mooney, R. 1986. Explanation-based learning: An alternative view. *Machine Learning*, **1**(2), 145–176.

DeJong, G. F. 1981. Generalizations based on explanations. *Seiten 67–69 aus: Proceedings of the 7th International Joint Conference on Artificial Intelligence.* Los Altos, CA: Morgan Kaufmann.

Dietterich, T. G.. 1986. Learning at the knowledge level. *Machine Learning*, **1**(3), 287–315.

Dijkstra, E. W. 1959. A note on two problems in connection with graphs. *Numerische Mathematik*, **1**, 269–271.

Dijkstra, E. W. 1976. *A Discipline of Programming.* Englewood Cliffs, N. J., USA: Prentice-Hall.

Doorenbos, R. und Veloso, M. 1993. Knowledge organization and the utility problem. *Seiten 28–34 aus: Proceedings of the Third International Workshop on Knowledge Compilation and Speedup-Learning.*

Duval, B. 1991. Abduction for explanation-based learning. *Seiten 348–360 aus:* Kodratoff, Y. (Hrsg), *Lecture Notes in Artificial Intelligence: Machine Learning-EWSL-91*, Band 482. Berlin: Springer.

Ellman, T. 1989. Explanation-based learning: A survey of programs and perspectives. *ACM Computing Surveys*, **21**(2), 163–222.

Ernst, G. und Newell, A. 1969. *GPS: A case study in generality and problem solving.* Academic Press.

Etzioni, O. 1990. Why PRODIGY/EBL works. *Seiten 916–922 aus:* Press, MIT (Hrsg), *Proceedings Eighth National Conference on Artificial Intelligence*, Band 2. London , England: MIT Press.

Etzioni, O. und Etzioni, R. 1994. Statistical methods for analyzing speedup learning. *Machine Learning*, **14**, 333–347.

Etzioni, O., Hanks, S., Weld, D., Draper, D., Lesh, N. und Williamson, M. 1992. An approach to planning with incomplete information. *Seiten 115–125 aus: Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning.* Morgam Kaufmann.

Eversheim, W. 1989. *Organisation in der Produktionstechnik, Band 3: Arbeitsvorbereitung.* VDI-Verlag.

Fabel. 1993. *Survey of* Fabel. Fabel Report 2. The Fabel Consortium, GMD.

Faltings, B., Hua, K., Schmitt, G. und Shih, S.-G. 1991. Case-based Representation of Architectural Design Knowledge. *Aus:* [Bareiss, 1991]. Washington, D.C., USA, May 8–10, 1991.

FIKES, R. E. UND NILSSON, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, **2**, 189–208.

FIKES, R. E., HART, P. E. UND NILSSON, N. J. 1972. Learning and executing generalized robot plans. *Artificial Intelligence*, **3**, 251–288.

FRANCIS, A. G. UND RAM, A. 1993. The utility Problem in Case-Based Reasoning. *Aus:* LEAKE, D. (Hrsg), *Proceedings AAAI-93 Case-Based Reasoning Workshop*AAAI Press, American Association fort Artificial Intelligence.

FUCHS, M. 1995a. *Exploiting past proof experience.* Bericht LSA-95-08E. Zentrum fü Lernende Systeme und Anwendungen, Universität Kaiserslautern.

FUCHS, M. 1995b. Learning proof heuristics by adaption parameters. *Seiten 235–243 aus: Proceedings of the International Conference on Machine Learning.* Morgan Kaufmann.

GENEST, J., MATWIN, S. UND PLANTE, B. 1990. Explanation-based learning with incomplete theories: A three step approach. *Seiten 286–294 aus:* PORTER, B. W. UND MOONEY, R. J. (Hrsg.), *Machine Learning: Proceedings of the Seventh International Conference on Machine Learning.* San Mateo, CA: Morgan Kaufmann.

GEORGEFF, M. P. 1987. Planning. *Ann. Rev. Comput. Sci.*, **2**, 359–400.

GEORGEFF, M. P. UND LANSKY, A. L. 1987. Reactive reasoning and planning. *Aus: Proceedings of the International Conference on Artificial Intelligence.* Morgan Kaufmann.

GIUNCHIGLIA, F. UND WALSH, T. 1992. A theory of abstraction. *Artificial Intelligence*, **57**, 323–389.

GLOBIG, C. UND WESS, S. 1993. Case-Based and Symbolic Classification Algorithms - A case Study using Version Space. *Aus:* [Richter *et al.*, 1993].

GOEL, A. UND CHANDRASEKARAN, B. 1989. Use of Device Models in Adaption of Design Cases. *Seiten 100–109 aus:* HAMMOND, K. J. (Hrsg), *Proceedings: Case-Based Reasoning Worhshop.* San Mateo, California: Morgan Kaufmann Publishers, DARPA. Pensacola Beach, Florida, USA, May 31–June 2, 1989.

GOEL, A., KOLODNER, J., PEARCE, M., BILLINGTON, R. UND ZIMRING, C. 1991. Towards a case-based reasoning tool for aiding conceptual design problem solving. *Aus:* [Bareiss, 1991]. Washington, D.C., USA, May 8–10, 1991.

GÜNTER, A. 1994. Fallbasiertes Konfigurieren mit hybriden Expertensystemen. *KI - Künstliche Intelligenz*, 82–84.

HALL, R. 1988. Learning by failing to explain: Using partial explanations to learn in incomplete or intractable domains. *Machine Learning*, **3**(1), 45–77.

HALL, R. 1989. Computational approaches to analogical reasoning: A comparative analysis. *Artificial Intelligence*, **39**(1), 39–120.

HAMMOND, K. J. 1986. *Case-Based Planning: An Integrated Theory of Planning, Learning and Memory.* Dissertation, Yale University, New Haven, Connecticut.

HAMMOND, K. J. 1989. *Case-Based Planning: Viewing Planning as a Memory Task.* Boston, Massachusetts: Academic Press.

HANKS, S. UND WELD, D. S. 1995. A domain-independent algorithm for plan adaptation. *Journal of Artificial Intelligence Research*, **2**, 319–360.

HENNESSY, D. UND HINKLE, D. 1991. Initial Results from Clavier: A Case-Based Autoclave Loading Assistent. *Aus:* [Bareiss, 1991]. Washington, D.C., USA, May 8–10, 1991.

HERZBERG, J. 1989. *Planen: Einführung in die Planungsmethoden der Künstlichen Intelligenz.* BI-Wissenschaftsverlag.

HINRICHS, T. R. UND KOLODNER, J. L. 1991. The Roles of Adaptation in Case-Based Design. *Aus:* [Bareiss, 1991]. Washington, D.C., USA, May 8–10, 1991.

HIRSH, H. 1987. Explanation-based generalisation in a logic-programming environment. *Seiten 221–227 aus:* MCDERMOTT, J. (Hrsg), *Proceedindings of the 10th International Joint Conference on Artificial Intelligence*, Band 1. Los Altos , CA: Morgan Kaufmann.

HOLTE, R. C., DRUMMOND, C., PEREZ, M. B., ZIMMER, R. M. UND MACDONALD, A. J. 1994. Searching with abstractions: A Unifying framework and new high-performance algorithm. *Seiten 263–270 aus: Proceedings of the 10th Canadian Conference on Artificial Intelligence.* Morgan Kaufmann Publishers.

HOLTE, R. C., MKADMI, T., ZIMMER, R. M. UND MACDONALD, A. J. 1995. *Speeding up problem solving by abstraction: A graph-oriented approach.* Bericht. University of Ottawa, Ontario, Canada.

HORZ, A. 1992. Ähnlichkeit von Fällen beim fallbasierten Planen. *Aus:* [Althoff *et al.*, 1992].

HUA, K., SMITH, I. UND FALTINGS, B. 1994. Integrated case-based building design. *Seiten 436–445 aus:* WESS, S., ALTHOFF, K.-D. UND RICHTER, M. M. (Hrsg.), *Topics in Case-Based Reasoning.* Springer.

HUMM, B. 1991. *Entwurf und Implementierung eines wissensbasierten Ähnlichkeitsmasses für die fallbasierte Planung am Beispiel der Arbeitsplanung für die Teilefertigung.* Diplomarbeit, Universität Kaiserslautern, Fachbereich Informatik.

IHRIG, L. UND KAMBHAMPATI, S. 1994. Evaluating the effectiveness of derivational replay in partial-order vs. state-space planning. *Seiten 8–12 aus:* AHA, DAVID (Hrsg), *Proceedings of the AAAI-Workshop on Case-Based Reasoning.*

JANETZKO, D., WESS, S. UND MELIS, E. 1992. Goal-Driven Similarity Assessment. *Aus:* OHLBACH, HANS-JÜRGEN (Hrsg), *Proc. German Workshop on AI (GWAI'92).* Springer Verlag.

KAMBHAMPATI, S. 1990a. A Theory of Plan Modification. *Aus: Proc. AAAI-90.*

KAMBHAMPATI, S. 1990b. Mapping and Retrieval During Plan Reuse: A Validation Structure Based Approach. *Aus: Proc. AAAI-90.*

KAMBHAMPATI, S. UND HENDLER, J. 1992. A Validation-Structure-Based Theory of Plan Modifications. *Artificial Intelligence.*

KAMBHAMPATI, S. UND KEDAR, S. 1994. A unified framework for explanation-based generalization of partially ordered partially instantiated plans. *Artificial Intelligence*, **67**, 29–70.

KAMBHAMPATI, S. UND SRIVASTAVA, B. 1995. Universal classical planner: An algorithm for unifying state-space and plan-space planning. *Aus: Proceedings of the 3rd European Planning Workshop (EWSP-95).*

KAMPHAMPATI, S. UND CHEN, J. 1993. Relative utility of EBG based plan reuse in partial ordering vs. total ordering planning. *Seiten 514–519 aus: Proceedings of the National Conference on Artificial Intelligence.*

KAPUR, D. UND NARENDRAN, P. 1986. NP-completeness of the set unification and matching problems. *Seiten 489–495 aus:* GOOS, G. UND HARTMANIS, J. (Hrsg.), *Proceedings of the 8th International Conference on Automated Deduction.* Springer-Verlag.

KARAMOUZIS, S. T. UND FEYOCK, S. 1994. Cases, models, and rules for fault management: An evaluation. *Seiten 62–66 aus:* AHA, DAVID (Hrsg), *AAAI-94 Workshop on Case-Based Reasoning.* American Association for Artificial Intelligence.

KEDAR, S. UND MCKUSICK, K. 1992. Tradeoffs in the utility of learned knowledge. *Seiten 281–282 aus: Proceedings of the First International Conference on AI Planning Systems.*

KEDAR-CABELLI, S. 1988. Analogy from a unified perspective. *Seiten 65–103 aus: Analogical Reasoning.* Kluwer Academic.

KEDAR-CABELLI, S. T. UND MCCARTY, L. T. 1987. Explanation-based generalization as resolution theorem proving. *Seiten 383–389 aus: Proceedings of the 4th International Workshop on Machine Learning.*

KERYSZIG, E. 1988. *Statistische Methoden und ihre Anwendung.* Vandenhoeck und Ruprecht.

KI - 1/96. 1996. *KI - Künstliche Intelligenz, Sonderausgabe zum fallbasierten Schließen.*

KIRA, K. UND RENDELL, L. A. 1992. A practical approach to feature selection. *Seiten 249–256 aus: Proceedings of the Ninth Internationcal Conference on Artificial Intelligence.* Morgan Kaufmann.

KNOBLOCK, C. A. 1989. A theory of abstraction for hierachical planning. *Seiten 81–104 aus: Proceedings of the Workshop on Change of Representation and Inductive Bias.* Boston, MA: Kluwer.

KNOBLOCK, C. A. 1990. Learning abstraction hierarchies for problem solving. *Seiten 923–928 aus: Proceedings Eighth National Conference on Artificial Intelligence*, Band 2. London: MIT Press.

KNOBLOCK, C. A. 1993. *Generating abstraction hierarchies: An automated approach to reducing search in planning.* Kluwer Academic Publishers.

KNOBLOCK, C. A. 1994. Automatically generating abstractions for planning. *Artificial Intelligence*, **68**, 243–302.

KNOBLOCK, C. A., TENENBERG, J. D. UND YANG, Q. 1991. Characterizing abstraction hierarchies for planning. *Seiten 692–697 aus: Proceedings of the 9th National Conference on Artificial Intelligence*, Band 2.

KNOBLOCK, C.A. UND YANG, Q. 1994. Evaluating the tradeoffs in partial-order planning algorithms. *Aus: Canadian Conference on Artificial Intelligence.*

KÖHLER, J. 1994. *Wiederverwendung von Plänen in deduktiven Planungssystemen.* DISKI 65, infix-Verlag, Dissertation, Universität Saarbrücken.

KOLBE, T. UND WALTER, C. 1994. Reusing proofs. *Seiten 80–84 aus: Proceedings of the 11th European Conference on Artificial Intelligence.* John Wiley and Sons, Ltd.

KOLODNER, J. L. 1983a. Maintaining organization in a dynamic long-term memory. *Cognitive Science*, **7**, 243–280.

KOLODNER, J. L. 1983b. Reconstructive memory : A computer model. *Cognitive Science*, **7**, 281–328.

KOLODNER, J. L. 1993. *Case-based reasoning.* Morgan Kaufmann.

KOLODNER, J. L., SIMPSON, R. L. UND SYCARA, K. 1985. A Process Model of Case-Based Reasoning in Problem Solving. *Seiten 284–290 aus: International Joint Conference on Artificial Intelligence IJCAI-85.* Morgan Kaufmann.

KONONENKO, I. 1994. Estimating attributes: Analysis and extensions of RELIEF. *Seiten 171–182 aus: Proceedings of the European Conference on Machine Learning.* Springer Verlag.

KORF, R. E. 1980. Toward a model of representation changes. *Artifical Intelligence*, **14**, 41–78.

KORF, R. E. 1985a. Depth-first iterative-deepening: An optimal admissible tree search. *Artifical Intelligence*, **27**, 97–109.

KORF, R. E. 1985b. Macro-Operators: A weak method for learning. *Artifical Intelligence*, **26**, 35–77.

KORF, R. E. 1987. Planning as search: A quantitative approach. *Artifical Intelligence*, **33**, 65–88.

KORF, R. E. 1988. Optimal path-finding algorithms. *Seiten 223–267 aus:* L. KANAL, V. KUMAR (Hrsg), *Search in Artificial Intelligence.* New York: Springer.

KORF, R. E. 1993. Linear-space best-first search. *Artifical Intelligence*, **62**, 41–78.

KOTON, P. 1988. Reasoning about Evidence in Causal Explanation. *Seiten 260–170 aus:* KOLODNER, J. L. (Hrsg), *Proceedings Case-Based Reasoning Workshop.* San Mateo, California: Morgan Kaufmann Publishers, DARPA. Clearwater Beach, Florida, USA, May 10–13, 1988.

KRIPKE, S. 1963. Semantic analysis of modal logic I: normal modal propositional calculi. *Math. Log. Grundl. Math.*, **9**, 67–96.

KUSHMERICK, N., HANKS, S. UND WELD, D. 1994. An algorithm for probabilistic least-commitment planning. *Seiten 1073–1078 aus: Proceedings of the National Conference on Artificial Intelligence.* Melno Park, California: American Association for Artificial Intelligence.

LAIRD, J. E., NEWELL, A. UND ROSENBLOOM, P. S. 1987. SOAR: An architecture for general intelligence. *Artificial Intelligence*, **33**, 1–64.

LAIRD, P. UND GAMBLE, E. 1990. EBG and term-rewriting systems. *Seiten 425–440 aus:* S. ARIKAWA, S. GOTO, OHSUGA, S. UND YUKUMORI, T. (Hrsg.), *Proceedings of the workshop on algorithmic learning theory.* Ohmsha-Verlag.

LEBOWITZ, M. 1990. The utility of similarity-based learning in a world needing explanation. *Kap. 15, Seiten 399–422 aus:* KODRATOFF, Y. UND MICHALSKI, R. S. (Hrsg.), *Machine learning: An artificial intelligence approach*, Band 3. San Mateo, CA: Morgan Kaufmann.

LIFSCHITZ, V. 1987. On the semantics of STRIPS. *Seiten 1–9 aus: Reasoning about Actions and Plans: Proceedings of the 1986 Workshop.*

LLOYD, J. W. 1984. *Foundations of Logic Programming.* Springer.

MACCHION, D. J. UND VO, D. P. 1994. A hybrid knowledge-based system for technical diagnosis learning and assistance. *Seiten 301–311 aus:* WESS, S., ALTHOFF, K.-D. UND RICHTER, M. M. (Hrsg.), *Topics in Case-Based Reasoning.* Springer.

MÄDLER, F. 1991. Problemzerlegung als optimalitätserhaltende Operatorabstraktion. *Seiten 74–83 aus:* CHRISTALLER, T. (Hrsg), *GWAI-91: 15. Fachtagung für Künstliche Intelligenz, Informatik Fachberichte.* Berlin: Springer.

MÄDLER, F. 1992. Towards structural abstraction. *Aus: Proceedings of the International Conference on Planning for AI Systems AIPS-92.*

MANAGO, M., ALTHOFF, K.-D., AURIOL, E., TRAPHÖNER, R., WESS, S., CONRUYT, N. UND MAURER, F. 1993. Induction and Reasoning from Cases. *Aus:* [Richter *et al.*, 1993].

MANAGO, M., BERGMANN, R., WESS, S. UND TRAPHÖNER, R. 1994. CASUEL*: A Common Case Representation Language - Version 2.0.* Bericht. Esprit Project INRECA Deliverable D1.

MAURER, F. 1993. *Hypermediabasiertes Knowledge Engineering für verteilte wissensbasierte Systeme.* DISKI, Band 48. infix, Sankt Augustin.

MCALLESTER, D. UND ROSENBLITT, D. 1991. Systematic nonlinear planning. *Seiten 634–639 aus: Proceedings of the 9th National Conference on Artificial Intelligence.*

MCCARTHY, J. 1980. Circumscription - a form of nonmonotonic reasoning. *Artificial Intelligence*, **13**, 27–39.

MICHALSKI, R. S. 1994. Inferential theory of learning as a conceptual basis for multistrategy learning. *Kap. 1, Seiten 3–62 aus:* MICHALSKI, R. UND TECUCI, G. (Hrsg.), *Machine Learning: A Multistrategy Approach.* Morgan Kaufmann.

MICHALSKI, R. S. UND KODRATOFF, Y. 1990. Research in machine learning: Recent progress, classification of methods, and future directions. *Kap. 1, Seiten 3–30 aus:* KODRATOFF, Y. UND MICHALSKI, R. S. (Hrsg.), *Machine learning: An artificial intelligence approach*, Band 3. San Mateo, CA: Morgan Kaufmann.

MINSKY, M. 1963. Steps toward artificial intelligence. *Aus:* FEIGENBAUM, E. (Hrsg), *Computers and Thought.* New York, NY: McGraw-Hill.

MINTON, S. 1988. *Learning Search Control Knowledge: An Explanation-Based Approach.* Kluwer, Boston, MA.

MINTON, S. 1990a. Issues in the design of operator composition systems. *Aus: Proceedings of the 7th International Conference on Machine Learning.*

MINTON, S. 1990b. Quantitativ results concerning the utility of explanation-based learning. *Artifical Intelligence*, **42**, 363–391.

MINTON, S., CARBONELL, J. G., KNOBLOCK, C. A., KUOKKA, D. R., ETZIONI, O. UND GIL, Y. 1989. Explanation-based learning: A problem solving perspective. *Artificial Intelligence*, **40**, 63–118.

MINTON, S., BRESINA, J. UND DRUMMOND, M. 1991. Commitment strategies in planning: A comparative analysis. *Seiten 259–265 aus: Proceedings of the International Conference on Artificial Intelligence.* Morgan Kaufmann.

MINTON, S., BRESINA, J. UND DRUMMOND, M. 1994. Total-order and partial-order planning: A comparative analysis. *Journal of Artificial Intelligence Research*, **2**, 227–262.

MITCHELL, T. M. 1983. Learning and problem solving. *Seiten 1139–1151 aus: Proceedings of the 8th International Joint Conference on Artificial Intelligence.* Los Altos, CA: Morgan Kaufmann.

MITCHELL, T. M., KELLER, R. M. UND KEDAR-CABELLI, S. T. 1986. Explanation-based generalization: A unifying view. *Machine Learning*, **1**(1), 47–80.

MOHRI, T. UND TANAKA, H. 1994. An optimal weighting criterion of case indexing for both numeric and symbolic attributes. *Seiten 123–127 aus: Proceedings AAAI-Workshop on Case-Based Reasoning.*

MOONEY, R. J. 1989. The effect of rule use on the utility of explanation-based learning. *Aus: Proceedings of the International Joint Conference on Artificial Intelligence.*

MOONEY, R. J. 1993. Explanation Generalization in EGGS. *Kap. 2, Seiten 20–59 aus:* DE JONG, G. (Hrsg), *Investigating Explanation-Based Learning.* Kluwer Academic Publishers.

MOSTOW, J. 1983. A problem solver for making advice operational. *Seiten 279–283 aus: Proceedings of the National Conference on Artificial Intelligence.* Los Altos, CA: Morgan Kaufmann.

MOSTOW, J. 1989. Design by Derivational Analogy: Issues in the Automated Replay of Design Plans. *Artificial Intelligence*, **40**(1-3), 119–184. September 1989.

MOZETIC, I. 1990. Abstraction in model-based diagnosis. *Seiten 64–75 aus: AAAI Workshop on Automatic Generation of Approximations and Abstractions.*

MUGGLETON, S. 1991. Inductive logic programming. *New Generation Computing*, **8**(4), 295–317.

MUÑOZ, H. UND HUELLEN, J. 1994. *Dynamically weighting relevant features in plan-space case-based planning.* Eingereicht zur Veröffentlichung.

MUÑOZ, H. UND HUELLEN, J. 1995. Retrieving cases in structured domains by using goal dependencies. *Aus: Proceedings of the International Conference on Case-Based Reasoning (ICCBR).*

MUÑOZ, H., PAULOKAT, J. UND WESS, S. 1994. Controlling a Nonlinear Hierachical Planner using Case-Based Reasoning. *Aus: Proceedings 2nd EWCBR.*

NAPOLI, A. UND LIEBER, J. 1993. Finding Strategies in Organic Synthesis Planning with Case-Based Reasoning. *Aus:* [Richter *et al.*, 1993].

NAVINCHANDRA, D., SYCARA, KATIA P. UND NARASIMHAN, S. 1991. Influences: A Thematic Abstraction for Creative Use of Multiple Cases. *Aus:* [Bareiss, 1991]. Washington, D.C., USA, May 8–10, 1991.

NEUBERT, S. 1993. *Modellkonstruktion in MIKE: Methoden und Werkzeuge.* DISKI, Band 60. infix, Sankt Augustin.

NEWELL, A. UND SIMON, H. 1962. Computer simulation of human thinking. *Science*, **134**, 2011–2017.

NEWELL, A. UND SIMON, H. 1972. *Human Problem Solving.* Prentice-Hall Englewood Cliffs, NJ.

NILSSON, N. J. 1980. *Principles of Artificial Intelligence.* Tioga Publishers.

PAULOKAT, J. UND WESS, S. 1993. Fallauswahl und fallbasierte Steuerung bei der nichtlinearen hierarchischen Planung. *Seiten 109–120 aus:* HORZ, A. (Hrsg), *Beiträge zum 7. Workshop Planen und Konfigurieren.* Arbeitspapiere der GMD, Nr. 723.

PAULOKAT, J. UND WESS, S. 1994. Planning for Machining Workpieces with a partial-order, nonlinear planner. *Aus: AAAI-Fall Symposium on Planning and Learning: On to Real Applications.*

PENBERTHY, J. S. UND WELD, D. 1994. Temporal planning with continuous change. *Seiten 1010–1015 aus: Proceedings of the National Conference on Artificial Intelligence.* Melno Park, California: American Association for Artificial Intelligence.

PEWS, G. 1994. MOCAS/2 - *Fallbasierte Diagnostik in technischen Domänenen.* Diplomarbeit, Fachbereich Informatik, Universität Kaiserslautern.

PEWS, G. UND WESS, S. 1993. Combining Case-Based and Model-Based Approaches for Diagnostic Applications in technical Domains. *Aus:* [Richter *et al.*, 1993].

PFITZNER, K. 1993. *Fallbasiertes Konfigurieren technischer Systeme.* Dissertation, Fachbereich Informatik, Universität Hamburg.

PLAISTED, D. 1981. Theorem proving with abstraction. *Artifical Intelligence*, **16**, 47–108.

PLAISTED, D. 1986. Abstraction using generalization functions. *Seiten 365–376 aus: Proceedings of the 8th Conference on Automated Deduction*, Band 16.

POLYA, G. 1945. *How to solve it.* Princeton, N.J.: Princeton University Press.

PORTER, B. W., BAREISS, R. UND HOLTE, R. C. 1990. Concept Learning and Heuristic Classification in Weak-Theory Domains. *Artificial Intelligence*, **45**.

PORTINALE, L., TORAASO, P., ORTALDE, C. UND GIARDINO, A. 1994. Using Case-Based Reasoning to Focus Model-Based Diagnostic Problem Solving. *Seiten 325 – 337 aus: Topics in Case-based Reasoning*, Band 2. Springer-Verlag.

RICHTER, M. M. 1978. *Logikkalküle.* Teubner Studienbücher Informatik.

RICHTER, M. M. 1992a. Classification and Learning of Similarity Measures. *Aus: Proc. der 16. Jahrestagung der Gesellschaft für Klassifikation e.V.* Springer Verlag.

RICHTER, M. M. 1992b. *Prinzipien der Künstlichen Intelligenz.* Stuttgart: Teubner Verlag. 2. Auflage.

RICHTER, M. M. 1995. *The knowledge contained in similarity measures.* Eingeladener Vortrag auf dem ICCBR'95 (auch unter URL http://wwwagr.informatik.uni-kl.de/~lsa/CBR/Richtericcbr95remarks.html).

RICHTER, M. M., WESS, S., ALTHOFF, K.-D. UND MAURER, F. (Hrsg.). 1993. *First European Workshop on Case-Based Reasoning (EWCBR-93).*

RIESBECK, C. K. UND SCHANK, R. C. 1989. *Inside Case-Base Reasoning.* Hillsdale, New Jersey: Lawrence Erlbaum Associates.

ROSENBLOOM, P. S. UND LAIRD, J. E. 1986 (August). Mapping explanation-based learning onto SOAR. *Aus: Proceedings National Conference on Artificial Intelligence*, Band 2.

ROSENBLOOM, P. S. UND NEWELL, A. 1986. The chunking of goal hierarchies: A generalized model of practice. *Kap. 10, Seiten 247–288 aus:* MICHALSKI, R. S., CARBONELL, J. G. UND MITCHELL, T. M. (Hrsg.), *Machine learning: An artificial intelligence approach*, Band 2. Los Altos, California: Morgan Kaufmann.

SACERDOTI, E. D. 1975. The nonlinear nature of plans. *Seiten 206–214 aus: Proceedings of the International Joint Conference on Artificial Intelligence.* Morgan Kaufmann.

SACERDOTI, E.D. 1974. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, **5**, 115–135.

SACERDOTI, E.D. 1977. *A Structure for Plans and Behavior*. Band 5. American-Elsevier, New York,.

SACHS, L. 1988. *Statistische Methoden: Planung und Auswertung*. Springer-Verlag.

SALZBERG, S. 1991. A Nearest Hyperrectangle Learning Method. *Machine Learning*, **6**, 277–309.

SCHANK, R. C. 1982. *Dynamic Memory: A Theory of Learning in Computers and People*. New York: Cambridge University Press.

SCHMIDT, G. 1994. *Modellbasierte, interaktive Wissensakquisition und Dokumentation von Domaenenwissen*. DISKI 90, infix-Verlag, Dissertation, Universität Kaiserslautern.

SCHOPPERS, M. J. 1987. Universal plans for reactive robots in unpredictable eviron-ments. *Seite 1039 aus: Proceedings of the International Joint Conference on Artificial Intelligence*. Morgan Kaufmann.

SEBANG, M. UND SCHOENAUER, M. 1994. A rule-based similarity measure. *Seiten 119–131 aus:* WESS, S., ALTHOFF, K.-D. UND RICHTER, M. M. (Hrsg.), *Topics in Case-Based Reasoning*. Springer.

SEGRE, A. M. 1987. On the operationality/generality trade-off in explanation-based learning. *Seiten 242–248 aus:* MCDERMOTT, J. (Hrsg), *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, Band 1. Los Altos , CA: Morgan Kaufmann.

SHAVLIK, J. W. UND O'RORKE, P. 1993. Empirically evaluating EBL. *Kap. 7, Seiten 222–294 aus: Investigating Explanation-Based Learning*, Band 5. Kluwer Academic Publishers.

SHAVLIK, J. W., DEJONG, G. F. UND ROSS, B. H. 1987. Acquiring special case schemata in explanation-based learning. *Seiten 851–860 aus: Proceedindings of Ninth Anual Conference of the Cognitive Science Society*, Band 1.

SHOHAM, Y. 1986. Chronological ignorance: time, nonmonotonicity, necessity and causal theories. *Seiten 389–393 aus: Proceedings of the Fith National Conference on Artificial Intelligence*.

SIMON, H. 1975. The functional equivalence of problem solving skills. *Cognitive Psychology*, **7**, 268–288.

SLATTERY, S., CUNNINGHAM, P. UND FINN, D. 1993. Modelling of Engineering Thermal Problems - An Implementation using CBR with Derivational Analogy. *Aus:* [Richter *et al.*, 1993].

SMYTH, B. UND CUNNINGHAM, P. 1992. Deja Vu: A Hierarchical Case-Based Reasoning System for Software Design. *Seiten 587–589 aus:* NEUMANN, B. (Hrsg), *ECAI-92.*

SMYTH, B. UND KEANE, M. 1994. Retrieving adaptable cases. *Seiten 209–220 aus:* WESS, S., ALTHOFF, K.-D. UND RICHTER, M. M. (Hrsg.), *Topics in Case-Based Reasoning.* Springer.

SMYTH, B. UND KEANE, M. 1995. Remembering to forget: A competence-preserving case deletion policy for case-based reasoning systems. *Seiten 377–383 aus:* MELLISH, CHRIS S. (Hrsg), *Proceedings of the International Conference on Artificial Intelligence.* Morgan Kaufmann Publishers.

STADLER, M. UND WESS, S. 1989. PATDEX – *Konzept und Implementierung eines fallbasierten analogieorientierten Inferenzmechanimus zur Diagnose eines CNC-Bearbeitungszentrums.* Projektarbeit, Universität Kaiserslautern. Betreuer: K.-D. Althoff.

STRUBE, G. UND JANETZKO, D. 1990. Episodisches Wissen und fallbasiertes Schließen: Aufgaben für die Wissensdiagnostik und die Wissenspsychologie. *Schweizerische Zeitschrift für Psychologie*, **49**(4), 211–221.

SUBRAMANIAN, D. UND SMITH, D. 1988. Knowledge level learning: An alternative view. *Seiten 196–200 aus:* DE JONG, G. (Hrsg), *Proceedings of the 1988 AAAI Symposium on Explanation-based Learning.*

SURMANN, D. 1993. *Implementierung und vergleichende Untersuchung verschiedener Varianten abstraktionsbasierter Planungsverfahren.* Projektarbeit, Universität Kaiserslautern.

SUSSMAN, G. J. 1975. *A Computer Model of Skill Acquisition.* New York: American Elsevier.

TADEPALLI, P. 1991. A formalization of explanation-based macro-operator learning. *Seiten 616–622 aus:* KAUFMANN, MORGAN (Hrsg), *Proceedings of the International Join Conference on Artifical Intelligence-91.*

TADEPALLI, P. UND ISUKAPALLI, R. 1992. Learning plan knowledge from simulators. *Aus:* TADEPALLI, P. (Hrsg), *Proceedings of the ML'92 Workshop on Knowledge Compilation and Speedup Learning.*

TAMBE, M. UND NEWELL, A. 1988. Some chunks are expensive. *Seiten 451–458 aus: Proceedings of the 5th International Conference on Machine Learning.*

TATE, A. 1977. Generating project networks. *Seiten 888–893 aus: Proceedings of the International Joint Conference on Artificial Intelligence.* Morgan Kaufmann.

TECUCI, G. UND KODRATOFF, Y. 1990. Apprenticeship learning in imperfect domain theories. *Kap. 19, Seiten 514–552 aus:* KODRATOFF, Y. UND MICHALSKI, R. S. (Hrsg.), *Machine learning: An artificial intelligence approach*, Band 3. San Mateo, CA: Morgan Kaufmann.

TENENBERG, J. 1987. Preserving consistency across abstraction mappings. *Seiten 1011–1014 aus:* MCDERMOTT, J. (Hrsg), *Proceedings of the 10th International Conference on Artifical Intelligence.* Los Altos, CA: Morgan Kaufmann.

TENENBERG, J. 1988. *Abstraction in Planning.* Dissertation, Computer Science Department, University of Rochester, New York.

UNRUH, A. UND ROSENBLOOM, P. S. 1989. Abstraction in problem solving and learning. *Seiten 590–595 aus: Proceedings of the International Join Conference on Artifical Intelligence-89.* Detroit, MI: Morgan Kaufmann.

V. NEUMANN, J. UND MORGENSTERN, O. 1944. *Theory of games and economic behavior.* John Wiley and Sons, Inc.

VELOSO, M. M. 1992a. *Learning by Analogical Reasoning in General Problem Solving.* Dissertation CMU-CS-92-174, Carnegie Mellon University.

VELOSO, M. M. 1992b. *Learning by analogical reasoning in general problem solving.* Dissertation, Carnegie Mellon University, Pittsburgh, PA.

VELOSO, M. M. 1994a. *Planning and learning by analogical reasoning.* Springer Verlag.

VELOSO, M. M. 1994b. Prodigy/Analogy: Analogical Reasoning in General Probelm Solving. *Aus:* [Wess *et al.*, 1994a].

VELOSO, M. M. UND BLYTHE, J. 1994. Linkability: Examining causal link commitments in partial-order planning. *Aus: Proceedings of the 2nd International Conference on Planning for AI Systems AIPS-94.*

VELOSO, M. M. UND CARBONELL, J. G. 1993. Towards scaling up machine learning: A case study with derivational analogy in PRODIGY. *Kap. 8, Seiten 233–272 aus:* MINTON, STEVEN (Hrsg), *Machine Learning Methods for Planning.* Morgan Kaufmann.

WALDINGER, R. 1977. Achieving several goals simulataneously. *Machine Intelligence*, **8**, 94–136.

WARREN, D. H. 1983. *The runtime environment for a PROLOG compiler using a copy algorithm.* Bericht 83/052. SUNY and Stone Brook, N.Y.

WEBERSKIRCH, F. 1994. *Realisierung eines nichtlinearen Planungssystems zur Unterstützung der Arbeitsplanerstellung bei der computerintegrierten Fertigung (CIM).* Diplomarbeit, Universität Kaiserslautern.

WELD, D. S. 1994. An introduction to least commitment planning. *AI Magazine*, **15**, 27–61.

WESS, S. 1991. PATDEX*/2: Ein System zum adaptiven, fallfokussierenden Lernen in technischen Diagnosesituationen.* SEKI-Working Paper SWP91/01. Dept. of Computer Science, University of Kaiserslautern, Germany.

WESS, S. 1993. PATDEX - Inkrementelle und wissensbasierte Verbesserung von Ähnlichkeitsurteilen in der fallbasierten Diagnostik. *Aus: Tagungsband 2. deutsche Expertensystemtagung XPS-93.* Hamburg: Springer Verlag.

WESS, S. 1995. *Fallbasiertes Problemlösen in wissensbasierten Systemen zur Entscheidungsunterstützung und Diagnostik: Grundlagen, Systeme und Anwendungen.* Dissertation, Fachbereich Informatik, Universität Kaiserslautern.

WESS, S. UND GLOBIG, C. 1994. Case-Based and Symbolic Learning - A Case Study. *Aus:* [Wess *et al.*, 1994a].

WESS, S., ALTHOFF, K.-D. UND RICHTER, M. M. (Hrsg.). 1994a. *Topics in Case-Based Reasoning - Selected papers from the First European Workshop on Case-Based Reasoning, Kaiserslautern 1993.* LNAI, Nr. 837. Springer Verlag.

WESS, S., ALTHOFF, K.-D. UND DERWAND, G. 1994b. Using kd-Trees to improve the Retrieval Step in Case-Based Reasoning. *Seiten 83–88 aus: Topics in Case-Based Reasoning.*

WIELEMAKER, J. 1992. *SWI-Prolog 1.6: Reference Manual.* Bericht. University of Amsterdam.

WIELEMAKER, J. UND ANJEWIERDEN, A. 1992. *PCE-4: Functional Overview.* Bericht. University of Amsterdam.

WILKE, W. 1993. *Entwurf und Implementierung eines Algorithmus zum wissensintensiven Lernen von Planabstraktionen nach der PABS-Methode.* Projektarbeit, Universität Kaiserslautern.

WILKE, W. 1994. *Entwurf, Implementierung und experimentelle Bewertung von Auswahlverfahren für abstrakte Pläne im fallbasierten Planungssystem PARIS.* Diplomarbeit, Universität Kaiserslautern, Germany.

WILKINS, D. 1984. Domain-independent planning: Representation and plan generation. *Artificial Intelligence*, **22**, 269–301.

WILKINS, D. 1988. *Practical Planning: Extending the classical AI planning paradigm.* Morgan Kaufmann.

WILKINS, D. 1990. Knowledge base refinement as improving an incorrect and incomplete domain theory. *Kap. 18, Seiten 493–513 aus: Machine Learning: An Artificial Intelligence Approach, Vol 3.* Morgan Kaufmann.

WINSTON, P. H., BINFORD, T. O., KATZ, B. UND LOWRY, M. 1983. Learning physical descriptions from functional definitions, examples and precedents. *Seiten 433–439 aus: Proceedings of the National Conference on Artificial Intelligence.* Morgan Kaufmann.

YANG, H. UND LU, W. 1994. Case adaptation in a case-based process planning system. *Aus: Proceedings of the International Conference on AI Planning.*

YANG, Q. UND TENENBERG, J. D. 1990. Abtweak: Abstracting a nonlinear, least commitment planner. *Seiten 204–209 aus: Proceedings of the 8th National Conference on Artificial Intelligence.*

Abbildungsverzeichnis

2.1 Welten und Aktionen in der Arbeitsplanung 16

3.1 Der Zyklus des fallbasierten Schließens nach Aamodt und Plaza, 1994. . . . 37
3.2 Übertragung von Problemlöseentscheidungen bei Derivational Analogy. . . 43
3.3 Einsparung von Inferenzen beim Nachweis der Konzeptzugehörigkeit dreier Beispiele 46
3.4 Beispiel eines Erklärungsgraphen 49
3.5 Basisalgorithmus zur Bestimmung der Zugehörigkeit eines Beispiels zu einem Zielkonzept. 50
3.6 Bestimmung der Lage der gelernten operationalen Zielkonzeptdefinition . . 53
3.7 Abstraktion und Konkretisierung 58
3.8 Mehrere Abstraktionsebenen, die in Bezug auf ihren Abstraktionsgrad geordnet sind 59
3.9 Problemlösen mit Abstraktion und Verfeinerung 60
3.10 Hierarchisches Problemlösen 62
3.11 Suchraum mit einem Verzweigungsgrad b und einer Lösungslänge l. 63
3.12 Beispiel für die Kosten $C_S(b, l, k)$ bei $b = 3$, $k = 1 \ldots 3$ und $l = 1 \ldots 5$. . . . 64

4.1 Wiederverwendung abstrakter Fälle 72
4.2 Beispiel für eine Beschreibung eines Werkstücks und eines Rohlings auf der konkreten Ebene 75
4.3 Beschreibung einer Lösung für die Fertigungsaufgabe aus Abbildung 4.2 auf der konkreten Ebene 76
4.4 Beispiel für die Beschreibung eines Werkstückes und eines Rohlings auf der abstrakten Ebene 77
4.5 Lösung für die Fertigungsaufgabe aus Abbildung 4.4 auf der abstrakten Ebene 77

4.6 Zwei neue Probleme auf der konkreten Ebene 78
4.7 Schematische Darstellung eines generalisierten Falls 80
4.8 Wiederverwendung generalisierter Fälle 80
4.9 Wiederverwendung generalisierter abstrakter Fälle 83
4.10 Beispiel für einen generalisierten Fall 84
4.11 Aufwand bei der Wiederverwendung generalisierter abstrakter Fälle in Abhängigkeit der Abstraktionsebene der Wiederverwendung 86
4.12 Abhängigkeit von Retrievalzeit und Wiederverwendungszeit nach Veloso (1992) 87
4.13 Ausschnitt aus einer Wiederverwendbarkeitshierarchie 87
4.14 Ein Beispiel für eine Wiederverwendbarkeitshierarchie 88
4.15 Die Architektur des PARIS Systems 89

5.1 Die Architektur des PARIS Systems 91
5.2 Idee der Fallabstraktion 95
5.3 Fallabstraktion mit Zustands- und Sequenzabstraktionsabbildung 98
5.4 Einfaches Beispiel für eine Fallabstraktion 98
5.5 Vereinigung verschiedener abstrakter Sichten 99
5.6 Vereinigung verschiedener Abstraktionsebenen 100
5.7 Die vier Phasen der Fallabstraktion am Beispiel 102
5.8 Algorithmus zur Bestimmung der konkreten Zustandssequenz 103
5.9 Algorithmus zur Bestimmung der ableitbaren abstrakten Sätze 104
5.10 Algorithmus zur Suche nach anwendbaren abstrakten Operatoren 105
5.11 Beispiel für einen Beweisbaum für eine Operatorvorbedingung 106
5.12 Algorithmus zur Suche nach gerichteten Pfaden im Graph 107
5.13 Algorithmus zur Entscheidung der schwachen Wiederverwendbarkeitsbedingung 110
5.14 Suchräume bei der Lösungsverfeinerung 111
5.15 Basisalgorithmus zur Lösungsverfeinerung 112
5.16 Algorithmus zur Lösungsverfeinerung mit Backtracking 116

6.1 Die Architektur des PARIS Systems 119
6.2 Ausschnitt aus einem Erklärungsgraphen 125

6.3 Schematische Darstellung einer Operatorerklärung 128
6.4 Operatorerklärung für hinzugefügte essentielle Sätze 129
6.5 Operatorerklärung für bedrohte essentielle Sätze 131
6.6 Operatorerklärung für nicht-bedrohte essentielle Sätze 131
6.7 Detaillierte Darstellung einer Operatorerklärung 132
6.8 Schematische Darstellung einer Vorbedingungserklärung 133
6.9 Schematische Darstellung einer Fallerklärung 134
6.10 Algorithmus zur Erzeugung einer Fallerklärung 135
6.11 Beispiel für eine Fallerklärung 137
6.12 Beispiel für eine generalisierte Fallerklärung 139
6.13 Beziehung zwischen einer Erklärung und einer generalisierten Erklärung . . 140
6.14 Grundalgorithmus zur Erklärungsgeneralisierung 142
6.15 Algorithmus zur Bildung des generalisierten Falles 144
6.16 Algorithmus zum Matchen eines generalisierten Zustandes 147
6.17 Algorithmus zum Test der Wiederverwendbarkeit eines generalisierten Falles und zur Bestimmung einer Lösungsspezialisierung 148

7.1 Die Architektur des PARIS Systems 151
7.2 Gewinnung der Trainingsbeispiele 152
7.3 Hilfsabbildung zum Beweis von Lemma 7.2 154
7.4 Aufteilen eines Mehrfachknotens der Wiederverwendbarkeitshierarchie . . . 160
7.5 Das „Hochziehen“ eines (Teil)-Knotens zur Korrektur der Wiederverwendbarkeitshierarchie 160
7.6 Algorithmus zur Realisierung des Update-Schrittes 161
7.7 Algorithmus zur Realisierung des Insert-Schrittes. 162
7.8 Eine Beispielfolge von Wiederverwendbarkeitshierarchien 163
7.9 Aufwand bei der Wiederverwendung generalisierter abstrakter Fälle in Abhängigkeit der Flexibilität der Wiederverwendbarkeit 165
7.10 Algorithmus zum Retrieval von generalisierten abstrakten Fällen 167
7.11 Zeitaufwände und Wahrscheinlichkeiten für die Betrachtung eines Knotens 169
7.12 Wiederverwendbarkeitshierarchie für das Beispiel aus Tabelle 7.1 172
7.13 Interpretation der Beispielvarianten 1 bis 4 173

8.1 Teilzieldistanz eines abstrakten Operators 182
8.2 Beispiel für die Zerlegung eines Werkstückes in Bearbeitungsbereiche . . . 183
8.3 Ein Plan zur Fertigung des Werkstückes aus Abbildung 8.2 186
8.4 Ein Beispiel für die Fallabstraktion des konkreten Falls aus Abbildung 8.3 . 190
8.5 Ergebnis der Generalisierung des abstrakten Falles AF_3 191
8.6 Eine Wiederverwendbarkeitshierarchie vor der Aufnahme der neuen Fälle . 191
8.7 Wiederverwendbarkeitshierarchie nach der Aufnahme der neuen Fälle . . . 192
8.8 Wiederverwendbarkeitshierarchie nach der Aufnahme weiterer Fälle 193
8.9 Neues Problem (Zielwerkstück und Rohling) auf konkreter und abstrakter Ebene 194
8.10 Instantiierter generalisierter abstrakter Plan aus Abbildung 8.5 und Lösungsverfeinerung 195
8.11 Vergleich von Ursprungsfall und neuer Lösung 196
8.12 Ein Beispiel für eine Wiederverwendbarkeitshierarchie nach dem Lernen von 100 konkreten Fällen. 197
8.13 Problemlösen ohne Wiederverwendung von Fällen 204
8.14 Flexibilität der Wiederverwendung 205
8.15 Prozentsatz gelöster Probleme 207
8.16 Problemlösezeit für gelöste Probleme 208

9.1 Der Zyklus des fallbasierten Schließens nach Aamodt und Plaza (1994) . . 216
9.2 Lösungsanpassung durch Transformation 222

Tabellenverzeichnis

3.1 Vergleich zwischen EBL und Derivational Analogy 57

7.1 Ein Beispiel zur lösungserhaltenden Wiederverwendbarkeit generalisierter abstrakter Fälle. 163

7.2 Vergleich der Kosten für den Wiederverwendbarkeitstest und die Lösungsverfeinerung 165

8.1 Prozentsatz gelöster Probleme 209

8.2 Problemlösezeiten für vollständige Suche und verschiedene Arten der Wiederverwendung 209

8.3 Anzahl der Testläufe mit signifikanter Leistungsverbesserung (Teil 1) . . . 210

8.4 Anzahl der Testläufe mit signifikanter Leistungsverbesserung (Teil 2) . . . 210

8.5 Prozentsatz gelöster Probleme 212

8.6 Durchschnittliche Problemlösezeit 213

8.7 Ergebnisse mit vereinfachter Problemlösedomäne 214

9.1 Vergleich von PARIS und PRODIGY/ANALOGY 226

9.2 Vergleich von PARIS und CAPLAN/CBC 228

9.3 Vergleich von PARIS und PRIAR 230

Index

Fett gedruckte Seitenzahlen weisen auf formale Definitionen oder Bezeichnungen hin.

Ableitungsoperator, 18
abstrakte Ebene, 57
abstrakte Problemlosedomane, **93**, 186
abstrakter Fall, 5, 72
Abstraktion, 5, 57–67, 235
Abstraktionsebenen, 5, 58, 73
ABSTRIPS, 235
ABTWEAK, 235
Adaption durch Transformation, 40
Addliste, **22, 23**
Ahnlichkeitsmas, 39
Aktionen, 14
Aktionsplanung, 1
ALPINE, 210, 236
Arbeitsplanung, 14, 15, 21, 74, 183–197
atomare Formeln, 18
Atome, 18
Aufwandsabschatzung, 115
Aussage, 125, 126, 137

Backtracking, 114
Basisknoten, 125, 137
bedrohte essentielle Satze, **130**
Begrundungen, 125, 137
Beispiel, **45**

CAPlan/CbC, 227
Constraints, 79, 120, 143, 147

Deleteliste, **22, 23**
Derivational Analogy, 42
Domane, 14, **24**
Domanentheorie, **45**, 124
Domanenunabhangigkeit, 4

Effizienz, 206, 207
Erklarung, 48, 125
erklarungsbasiertes Lernen, 6, **47**, 44–56, 81, 231
essentielle Satze, **19**
essentielle Satzformen, **19**

Fall, 36, **42**
Fallabstraktion, 73, **97**, 92–108, 189
Fallauswahl, 84–89
fallbasiertes Schliesen, 35–44, 215, 234
Fallbasis, 36, 86, 192
Fallerklarung, 126, 134
Fallgeneralisierung, 81, 120–145, 189
Fallretrieval, 6
Flexibilitat, 2, 4, 204
Folgerungsknoten, 127
Formel, 18

generalisierte Erklarung, 137
generalisierter abstrakter Fall, 6, 82
generalisierter Fall, 5, 79, **121**, 143
generalisierter Zustand, **120**
Generalisierung, 5, **122**
generative Adaption, 40
generische Abstraktionstheorie, **94**, 182

Hierarchisches Problemlosen, 59
HIGHPOINT, 236
Hintergrundwissen, 216
Hornklauseln, 18
Hornlogik, 103, 126, 140

Hypothesen, 198

Implementierung, 179
innere Knoten, 125, 137
instantiierte Operatorbeschreibung, **24**
Interpretation, **25**

Kompetenz, 206, 207
konkrete Ebene, 57, 60
konkrete Problemlosedomane, **93**, 183
konkreter Fall, 72
Konkretisierung, 57
Konzeptspezialisierung, **46**
korrekte Losung, 15, **25**
Korrektheit, 4
Kosten, 63, 84, 231

Leistungsfahigkeit, 3
Losung, 14, **25**
Losungsklasse, 79, **121**, 143
Losungsraum, 32
Losungsspezialisierung, 82, 145–149, 194
Losungsverfeinerung, 61, **110**, 110–118, 195

Makrooperatoren, 233

Nutzlichkeit, 50
Nutzlichkeitsproblem, 53

operationale Konzeptspezialisierung, 124, 140
Operationalitatskriterium, **46**, 53, 124, **140**
Operator, **22, 23**
Operatorbeschreibung, **22**
Operatorerklarung, 127–132
Operatorknoten, 128

PABS-Algorithmus, 100
parametrisiertere Operatorbeschreibung, **23**
PARIS, 7, 89–90, 179
Plan, 15, **25**
Planabstraktion, 60
Plangeneralisierungen, 233
Plankonkretisierung, 61
Planverfeinerung, 61
Pradikatenlogik, 17
PRIAR, 228
Problem, 14, **25**
Problemabstraktion, 61
Problemklasse, 79, **120**, 143
Problemlosedomane, 14, **24**
Problemlosefall, 42
Problemlosen, 13–34
PRODIGY/Analogy, 224
PROLOG, 20, 103

Qualifikationsproblem, 28

Rangtest, 203
Retain-Phase, 41, 223
Retrieval-Phase, 220
Retrieve-Phase, 38
Reuse-Phase, 40, 221
Revise-Phase, 41, 223

Schutzbedingung, **130**
schwache Wiederverwendbarkeit, **109**
Sequenzabstraktionsabbildung, **96**
Situation, 14
SLD-Resolution, 20, 103, 132, 148
STRIPS, 16
Substitution, 18

Teilzieldistanz, 181
Term, 17
Testfalle, 200
Tiefensuche, 115
Trainingsbeispiel, 124, 126, 140
Trainingsfalle, 200

Unabhangigkeit abstrakter Operatoren, 181
universelle Implikation, **20**

Vollstandigkeit, 180
Vorbedingung, **22, 23**
Vorbedingungserklarung, 127, 132–134

Vorzeichentest, 202

Welt, 14
Wiederverwendbarkeit abstrakter Falle, 73, **108**
Wiederverwendbarkeit generalisierter Falle, 82, 145
Wiederverwendbarkeitshierarchie, 6
Wiederverwendbarkeitsrelation, 87
Wiederverwendung, 5
Wiederverwendungsaufwand, 85
Wissen, allgemeines, 216
Wissensakquisition, 180

Zielkonzept, **45**, 124
Zielkonzeptes, 126
zulassige Pradikate, **120**
Zustand, **19**
Zustandsabstraktion, 60
Zustandsabstraktionsabbildung, **95**
Zustandsabstraktionsabbildung, gerechtfertigte, **95**
Zustandsknoten, 127
Zustandskonkretisierung, 60
Zustandsmatching, 146
Zustandsraum, **19**, 31
Zustandsubergang, **22**, **24**

Lebenslauf

Persönliche Daten

Name	Ralph Bergmann
Anschrift	Marie-Juchaczstr. 16
	67663 Kaiserslautern
geboren am	29. Januar 1965 in Trier
Staatsangeh.	Deutsch
Familienstand	ledig

Ausbildung

1971–1975	Grundschule Trier-Ehrang
1975–1984	Friedrich-Spee-Gymnasium Trier
1984	Abschluß: Abitur
1984 – 1990	Studium der Informatik mit Nebenfach Elektrotechnik
	an der Universität Kaiserslautern
1990	Abschluß: Diplom-Informatiker

Berufstätigkeit

1990 – 1992	wissenschaftlicher Mitarbeiter
	an der Universität Kaiserslautern (DFKI-Projekt)
seit 1992	wissenschaftlicher Mitarbeiter
	an der Universität Kaiserslautern
	(FB-Informatik, AG Künstliche Intelligenz – Expertensysteme)