Diplomarbeit

MoCAS/2 - Fallbasierte Diagnose in technischen Domänen:

Die Verwendung von Abstraktion zur Ähnlichkeits-
bestimmung und Fallübertragung durch Integration
von Wissen aus technischen Modellen

Gerhard Pews

Juni 1994

Universität Kaiserslautern
Fachbereich Informatik
Arbeitsgruppe Expertensysteme
Leiter: **Prof. Dr. M. M. Richter**
Betreuer: **Dipl. Inform. Ralph Bergmann**
Dr. Frank Maurer
Dipl. Inform. Stefan Weß

Erklärung

Hiermit versichere ich, daß ich diese Arbeit selbständig angefertigt habe und keine anderen Hilfsmittel als die im Anhang aufgeführten verwendet habe.

Kaiserslautern, den 24.06.94

Gerhard Pews

Inhaltsverzeichnis

Überblick

Thema der Arbeit

In dieser Arbeit wird ein Ansatz zur fallbasierten Diagnostik vorgestellt, der durch Integration von Wissen aus simulationsfähigen Modellen der Domäne den fallbasierten Problemlöseprozeß unterstützt. Dabei liegt der Schwerpunkt auf den zwei Schritten:

1. **Ähnlichkeitsbestimmung**. Sie erfolgt auf einer semantischen Ebene. Dazu werden Erklärungsstrukturen konstruiert, die beschreiben, welche Auswirkungen ein Defekt (also die Diagnose) auf die restlichen Komponenten der Domäne (die beobachteten Symptome) hat.
2. **Fallübertragung**. Ähnlichkeit zwischen Komponenten wird durch eine Abstraktionshierarchie repräsentiert. Alles Wissen, daß zur Ähnlichkeitsbestimmung notwendig ist, muß ebenfalls hierarchisch organisiert werden.

Durch dieses Vorgehen kann die Größe der Fallbasis in der Regel erheblich verkleinert werden, ohne die Mächtigkeit des Systems zu beeinträchtigen. Weiterhin sind die generierten Lösungsvorschläge (hypothetische Defekte) durch Modellwissen abgesichert. Dadurch sind sie für einen Benutzer also insoweit plausibel, daß seine Beobachtungen sich durch den vorgeschlagener Defekt erklären lassen.

Der Ansatz wurde implementiert und mit einer Beispieldomäne getestet.

Aufbau der Arbeit

Im ersten Kapitel werden die Grundlagen für diese Arbeit beschrieben. Da diese Arbeit keine Kenntnisse im Bereich des fallbasierten Schließens voraussetzt, wird dieser Ansatz zunächst allgemein vorgestellt. Danach folgt eine kurze Einführung in die Problemstellung der Diagnose in technischen Domänen; dabei werden auch die grundlegende Begriffe und Bezeichnungen für diese Arbeit erläutert. Es folgt dann eine Beschreibung eines fallbasierten Systems, das mit der Technik des Instance-Based-Learning arbeitet. Die Probleme, die sich dabei ergeben, werden ausführlich diskutiert. Ebenfalls wird noch kurz auf modellbasierte Schlußtechniken eingegangen. Diese Betrachtungen leiten über zum Ansatz dieser Arbeit, der im zweiten Kapitel vorgestellt wird.

Im zweiten Kapitel wird zunächst eine informelle Beschreibung des Ansatzes gegeben. Dabei werden die Mechanismen zur Ähnlichkeitsbestimmung und Fallübertragung erläutert. Dazu wird auch ausführlich auf die verwendete Wissensrepräsentation eingegangen.

Im dritten Kapitel wird der Ansatz noch einmal in algorithmischer Form dargestellt, um die Vorgehensweise genau nachvollziehbar zu machen.

Das vierte Kapitel beschäftigt sich mit der Implementierung des Ansatzes. Dabei wird auf das Expertensystem-Werkzeug CoMo-Kit kurz eingegangen, mit dessen Hilfe große Teile der Implementierung durchgeführt wurden und das an verschiedenen Stellen erweitert worden ist.

Im fünften und letzten Kapitel findet sich eine Bewertung des dargestellten Ansatzes. Dazu wird auch kurz auf verwandte Arbeiten wie Derivational Analogy oder PROTOS eingegangen, um MoCAS/2 gegen diese abzugrenzen. Der Diskussion der Vor- und Nachteile folgt ein Ausblick, der sich mit Erweiterungen des Systems, besonders in Hinblick auf Planung, befaßt.

Im Anhang findet sich ein Beispiellauf einer Diagnosesitzung mit MoCAS/2, um einen Einblick in die Arbeit mit dem System zu bieten und eine Anleitung zur Benutzung der Werkzeuge, mit denen Wissen in MoCAS/2 dargestellt und verändert werden kann.

Um einen Einblick in die besonderen Anforderungen in die Problemstellung der Diagnose zu geben und mit den Grundzügen fallbasierten Schließens vertraut zu machen, ist das erste Kapitel sicherlich gut geeignet; weil dort aber auch die Domänen vorgestellt und die Schwierigkeiten erläutert werden, mit denen sich MoCAS/2 besonders befaßt, ist dieses Kapitel auch Lesern empfohlen, die mit der Materie vertrauter sind. Nicht zuletzt hilft dieses Kapitel auch, Mißverständnissen vorzubeugen, die sonst durch die Verwendung von Begriffen wie *Erklärung*, *Modell* oder *Abstraktion* entstehen können.

Wer einen schnellen Einstieg sucht, kann sich zunächst durch das Sitzungsprotokoll im Anhang einen Eindruck von der Funktionsweise des Systems verschaffen, um sich dann mit den beiden ersten Kapiteln gründlicher über MoCAS/2 zu informieren.

Das Kapitel vier, das sich mit der Implementierung befaßt, ist sicherlich auch für den Leser interessant, der hauptsächlich am Aspekt des Knowledge Engineering und an der Modellierung von Systemen durch CoMo-Kit interessiert ist.

KAPITEL 1 Einführung

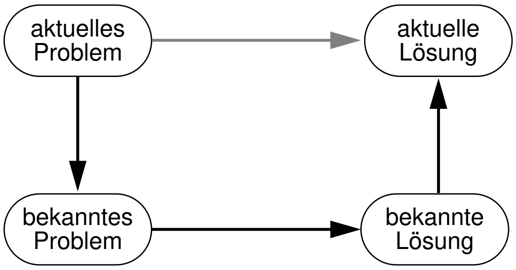
In diesem einleitenden Kapitel sollen die Grundlagen für das Verständnis dieser Arbeit dargelegt werden. Zunächst wird näher auf die allgemeine Vorgehensweise beim fallbasierten Schließen eingegangen, danach werden wichtige Begriffe eingeführt, die an der Begriffswelt der fallbasierten Diagnose in technischen Domänen orientiert sind. Im Anschluß daran wird das System PATDEX/2 vorgestellt. An PATDEX/2 sollen exemplarisch Defizite vieler bestehender Ansätze aufgezeigt werden. Schließlich wird noch kurz das Gebiet der modellbasierten Diagnose angerissen, da der Ansatz, der in dieser Arbeit vorgestellt werden soll, modellbasierte Schlußweisen integriert.

1.1 Fallbasiertes Schließen

Fallbasiertes Schließen ist eine relativ junge Technik für die Entwicklung von Expertensystemen [Kolodner93]. Es entstand zu Beginn der achtziger Jahre durch die Beobachtung, daß regelbasierte Expertensysteme sich ihre Schlußfolgerungen immer wieder aufs neue herleiten mußten, selbst wenn das Problem vorher schon erfolgreich gelöst worden war.

Menschen können sich an erfolgreiche (oder auch erfolglose) Lösungsversuche einer Aufgabe erinnern und dieses Wissen für zukünftige Aufgabenstellungen verwenden. Dies wäre auch eine wünschenswerte Eigenschaft von Expertensystemen. Zur Bearbeitung einer neuen Aufgabe soll sich das System an eine bereits gelöste, *ähnliche* Aufgabe aus seinem Fallgedächtnis erinnern und deren Lösung zum Lösen des aktuellen Problems verwenden. Dabei kann die gefundene Lösung entweder unverändert übernommen oder der Situation entsprechend angepaßt werden (vgl. Abbildung 1). Um zu einem aktuellen Problem eine Lösung zu finden (grauer

ABBILDUNG 1 Die Grundidee fallbasierten Schließens



Pfeil), sucht man ein bereits gelöstes, ähnliches Problem. Dessen Lösung nutzt man dann, um zu einer Lösung für das aktuelle Problem zu kommen. Dabei genügt oft eine einfache Zuordnung zwischen bekannten Problem und bekannter Lösung ohne Kenntnis, warum die Lösung eine Antwort auf das Problem ist. Bei wissensintensiveren Vorgehensweisen (z. B. Derivational Analogy, Seite 69) ist aber auch der Lösungsweg - oder ein Teil davon - gefordert.

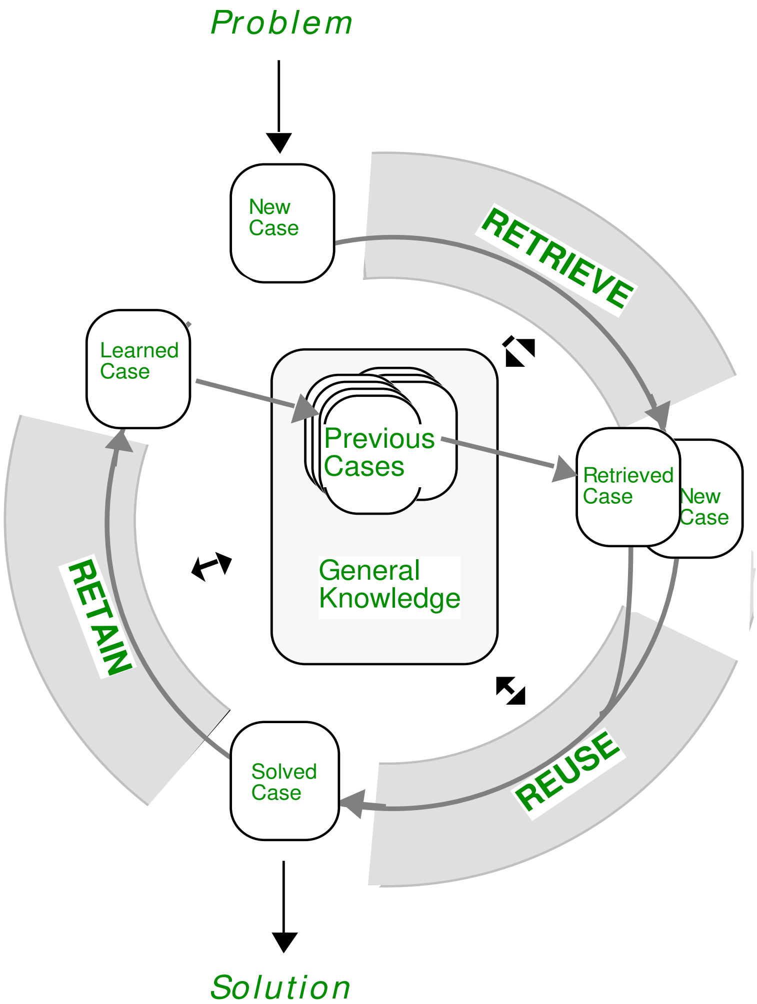
Die fallbasierte Vorgehensweise kann man durch drei typische Schritte kennzeichnen, die in [Aamodt91] zu finden sind: Die Schritte Retrieve, Reuse und Retain, also die Fähigkeit, ähnliche Fälle aufzufinden, wiederzuverwenden und im Gedächtnis zu behalten. Sie sind im Prozeßmodell in Abbildung 2 dargestellt.

Der Grundgedanke fallbasierten Schließens ist also das Verwenden von Erfahrungen. Durch einen wachsenden Erfahrungsschatz kann das System hinzulernen, ohne daß - wie beispielsweise bei induktiven Techniken - eine erneute Lernphase durchgeführt werden muß, in der der gesamte Wissensbestand neu bewertet wird, bzw. neue Schlüsse daraus gezogen werden.

Eine weitere Hoffnung, die mit dem fallbasierten Schließen (aber auch mit anderen Techniken des Maschinellen Lernens) verbunden ist, ist die, den „Flaschenhals der Wissensakquisition“ [Feigenbaum80] umgehen zu können. Wissensbasierte Systeme erfordern oft große Mengen von Wissen, das zudem noch in auf das System zugeschnittener Form dargestellt werden muß, beispielsweise in Form von Regeln. Menschen tun sich allerdings schwer, ihr Wissen in solchen Repräsentationsformalismen anzugeben, da Wissen oft unbewußt vorhanden oder nur schlecht strukturiert ist. Durch das Aufzählen von Fallbeispielen hofft man nun, die Wissensakquisition einfacher vornehmen zu können.

Damit verbunden ist ein anderer Vorteil dieses Ansatzes: Man kann auch mit Domänen arbeiten, die in ihren inneren Zusammenhängen und Wirkungsweisen nur unvollständig bekannt sind, wie etwa Anwendungen im Bereich der Wirtschaft. Hier besteht die Hoffnung, durch den Gebrauch von Fallbeispielen diese schwer formalisierbaren Domänen handhabbar zu machen.

ABBILDUNG 2 Prozeßmodell des fallbasierten Schließens aus [Aamodt91]



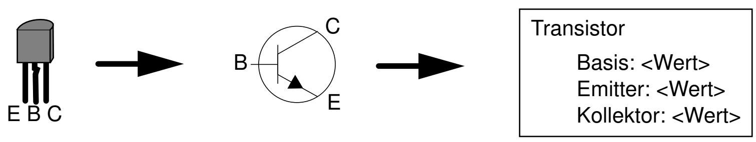
Bevor wir jetzt näher auf eine konkrete Anwendung des fallbasierten Schließens eingehen, sollen zunächst einmal grundlegende Begriffe geklärt werden, so wie sie in dieser Arbeit Verwendung finden. Als Beispiel für die Modellierung wollen wir hier technische Domänen betrachten und uns als Problemstellung mit der Diagnose von Geräten beschäftigen. Dieses Beispielgebiet hat den Vorteil, einerseits klar strukturiert zu sein, andererseits ist es aber allgemein genug, um Konzepte auch auf andere Bereiche übertragen zu können.

1.2 Begriffsbestimmungen und Modellierung der Domäne

Zunächst wird zunächst gezeigt, wie man eine Maschine, ihre Komponenten (Bauteile) und ihr Verhalten modellieren kann. Das liefert dann die Grundlage für Fallbeschreibungen, für die danach die wichtigsten Begriffe informell eingeführt werden.

Die im folgenden vorgestellte Modellierungsweise von Bauteilen basiert im wesentlichen auf der Modellierung im Expertensystem MOLTKE, sie ist ausführlich in [PfeiferRichter93], [Rehbold91] und [Schuch92] beschrieben. Ein technisches Bauteil, beispielsweise ein Transistor, hat verschiedene Ein- und Ausgänge, über die er mit seiner Umwelt verbunden ist (Basis, Kollektor und Emitter). Das führt zu einer frame-artigen Darstellung, wie sie in Abbildung 3 dargestellt ist. Die

ABBILDUNG 3 Modellierung eines Transistors



Ein-/Ausgänge (Ports) werden als Attribute des Transistors betrachtet.

Zusätzlich zu den Ein- und Ausgängen kann man auch noch interne Zustände (States) modellieren. Das macht die Modellierung übersichtlicher, kann aber grundsätzlich auch durch einen rückgekoppelten Ausgang repräsentiert werden.

Wenn man an bestimmte Leitungen Spannung anlegt, bewirkt das, daß daraufhin an anderen Leitungen Spannung anliegt, der Transistor also einen bestimmten Gesamtzustand einnimmt. Diesen Zustand kann man beschreiben, indem man die Werte der Ein- und Ausgänge als Attribut-Wert-Paare notiert, beispielsweise: (Basis: hohes_Potential), (Kollektor: hohes_Potential), (Emitter: niedriges_Potential). Wir wollen uns hier der Einfachheit halber auf eine statische, qualitative Beschreibung beschränken.

Die Funktionsweise eines solchen Bauteils kann man dann durch Regeln, bzw. Constraints beschreiben, beispielsweise: (Basis: hohes_Potential & Emitter: niedriges_Potential -> Kollektor: niedriges_Potential). Wie man an diesem Beispiel vielleicht schon sehen kann, hängt die Funktionsweise oft nicht nur vom Bauteil selbst ab, sondern auch noch von seiner Umgebung, in unserem Fall also vom Schaltkreis, in den der Transistor eingebaut ist. Darauf werden wir später noch näher eingehen.

Wenn man Bauelemente zu Baugruppen zusammenfügt, erzeugt man damit neue, komplexe Bauelemente, die wiederum Ein- und Ausgänge besitzen und die analog zu den atomaren Bauelementen beschrieben werden können. Auch Baugruppen lassen sich wiederum zu weiteren Baugruppen zusammenfassen, wobei eine baumartige Zerlegungsstruktur entsteht, deren Wurzel dann die ganze Maschine bildet. Den Zustand einer Maschine kann man jetzt durch die Angabe von Ein- und Ausgangswerten auf beliebigen Detaillierungsgraden beschreiben, wobei diese Grade in einer einzelnen Zustandsbeschreibung auch gemischt sein können.

Mit den bisher gemachten Überlegungen ist man in der Lage, Zustände einer Maschine und einzelner Bauteile zu beschreiben. Das wird genutzt, um Fallbeschreibungen zu erzeugen. Die Vorstellung der Begriffe geschieht „bottom-up“, zuerst werden einzelne Meßwerte beschrieben, diese Beschreibungen werden dann wiederum zu komplexeren Beschreibungen zusammengesetzt. Da die Fallbeschreibung, die für den Ansatz dieser Arbeit notwendig ist, von der Fallbeschreibung in MOLTKE abweicht, werden die benötigten Begriffe neu eingeführt und informell beschrieben.

Begriff: „Symptom“

Ein Paar aus Ein- bzw. Ausgangsbezeichung und dem dort anliegenden Wert heißt Symptom

Dabei wird der Begriff „Symptom“ nicht wie in der Umgangssprache mit einem pathologischen Symptom gleichgesetzt. Ein „Symptom“ beschreibt lediglich einen beobachteten Wert an einer bestimmten Stelle, gleichgültig, ob dieser Wert erwünschtes oder unerwünschtes Verhalten widerspiegelt.

Da wir es bei der Beschreibung der Maschine mit einer Baumstruktur zu tun haben und es vorkommen kann, daß Teilbäume mehrfach auftauchen, ist die Ein/Ausgangsbezeichnung als Pfad durch den Srukturbaum zu verstehen, beispielsweise: „Kollektor von Transistor T5 in Schaltkreis S2 auf Steckkarte ST4 in Baugruppe B1“. Das Mehrfachvorkommen von Teilbäumen (und damit von Baugruppen) ist in solchen Beschreibungen die Regel und vom Standpunkt des Konstrukteurs auch angestrebt.

Begriff: „Situation“

Eine Menge von Einzelsymptomen heißt Situation.

Durch eine Menge (Konjunktion) von Symptomen wird der Zustand des betrachteten Systems beschrieben. Diese Beschreibung ist normalerweise nicht vollständig; von vielen überhaupt erhebbaren Symptomen sind typischerweise nur sehr wenige gegeben.

Es ist sinnvoll, diese Beschreibung noch weiter aufzuteilen, denn eine Situation, in der ein Störfall in der Maschine aufgetreten ist, beschreibt folgendes: Der Benutzer hat die Maschine in einem bestimmten Zustand vorgefunden und wollte sie dazu veranlassen, eine bestimmte Aktion durchführen. Er beobachtete aber ein anderes, ungewolltes Verhalten, das sich von dem erwarteten Verhalten unterschied. Daher wird eine Situation noch in zwei Mengen von Symptomen unterteilt: Das „Intendierte Verhalten“ und das „Observierte Verhalten“

Das observierte Verhalten hat seine Ursache im Defekt eines Bauteils (oder auch mehrerer Bauteile). Dies wird durch die Diagnose beschrieben:

Begriff: „Diagnose“

Eine Diagnose ist eine Menge von Symptomen, die das Defektverhalten eines Bauteils beschreiben.

Da wir den Zustand eines Bauteils durch Symptome (die Werte an seine Ein- und Ausgängen) gut beschreiben können, bietet sich eine solche Darstellung an. Wir haben damit den Defekt des Bauteils vollständig beschrieben; allerdings können verschiedene Defekte durch gleiche Zustände beschrieben sein: Ein verklemmtes Relais und ein verschmortes Relais können gleiches Verhalten nach außen zeigen - allerdings ist es fraglich, ob man hier eine Unterscheidung treffen sollte; man sollte nicht über-diagnostizieren.

Kombiniert man nun eine Situationsbeschreibung mit der Beschreibung eines Defekts (also der Diagnose) erhalten wir einen Fall.

Begriffe: „Fall", „Fallbasis"

Eine Situationsbeschreibung zusammen mit einer Diagnose heißt Fall; eine Menge von Fällen Fallbasis.

Um in einem Diagnosesystem sinnvoll mit dieser Fallbasis arbeiten zu können, also Fälle zu finden, deren Lösungen für die aktuelle Problemstellung nützlich sind, benötigen wir abschließend noch eine Methode, die Ähnlichkeit zwischen der Problemstellung (einer Situation) und einem Fall zu bestimmen.

Begriff: „Ähnlichkeitsmaß"

Eine Berechnungsvorschrift, die die Ähnlichkeit zwischen einem Fall der Fallbasis und einer Situation bewertet, heißt Ähnlichkeitsmaß.

Meist ist ein Ähnlichkeitsmaß eine Funktion, die zu einem Fall und einer Situation einen Wert zwischen Eins und Null liefert (wie etwa in PATDEX/2, das im nächsten Abschnitt vorgestellt wird). Die Ähnlichkeit kann aber auch durch einen komplexen Algorithmus bestimmt werden, so wie es im Ansatz dieser Arbeit realisiert wird.

Wichtig ist an dieser Stelle zunächst nur, den Zusammenhang zwischen Ähnlichkeitsmaß und Fallbasis darzustellen: Ein leistungsstarkes Ähnlichkeitsmaß benötigt eine kleine Fallbasis, ein schwaches Maß benötigt eine große Fallbasis. Ein extremer Fall für ein schwaches Ähnlichkeitsmaß wäre die Prüfung auf Gleichheit, wie etwa bei einem einfachen Datenbankzugriff. Die Fallbasis müßte dann alle nur denkbaren Fälle umfassen. Da deren Anzahl oft unendlich ist (etwa bei reellwertigen Ein/Ausgangswerten), muß man sich über ein geschickteres Maß Gedanken machen.

Im Idealfall kommt ein solches Maß mit einem prototypischen Fall für jeden Defekt aus und ist in der Lage, Unterschiede zur gegebenen Anfragesituation angemessen zu bewerten und trotzdem die richtige Lösung bzw. Diagnose zu finden. In der Regel ist für den Einsatz eines konkreten fallbasierten Systems von vornherein erkennbar, in welcher Anzahl Fälle zur Verfügung stehen. Daraus ergibt sich dann die Wahl des Ähnlichkeitsmaßes.

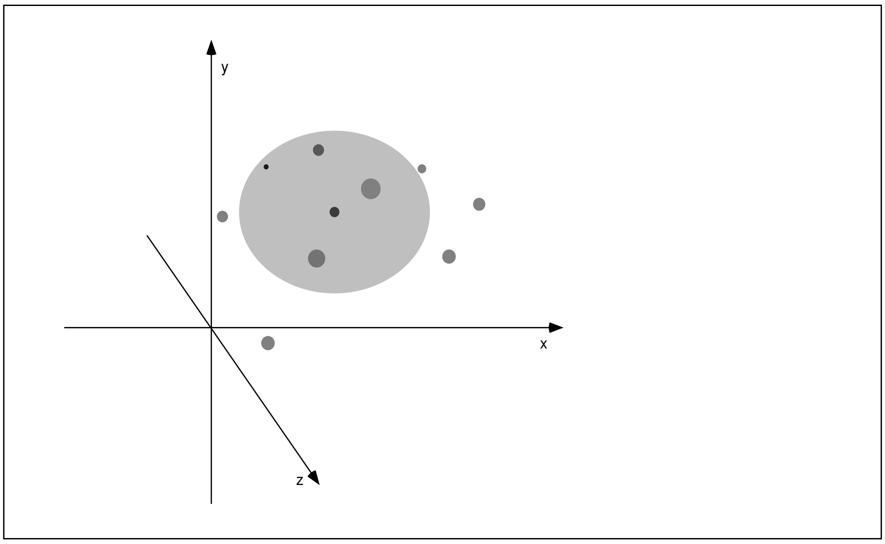
Im nächsten Abschnitt soll ein System vorgestellt werden, daß einen fallbasierten Ansatz zur Diagnose verfolgt: PATDEX/2 (Pattern Directed Expert System, Version 2). Der darin verfolgte Ansatz des Instance-Based-Learning ist ein weitverbreiteter Spezialfall fallbasierten Schließens, der sehr oft sogar mit fallbasiertem Schließen gleichgesetzt wird.

1.3 PATDEX/2

Die Vorgehensweise des Instance-Based-Learning in PATDEX/2 kann man sich sehr gut an einem Modell klarmachen. Eine Situationsbeschreibung besteht aus einer Menge von Symptomen, also Attribut/Wert-Paaren. Zur besseren Darstellbarkeit sollen nur die drei Attribute *x*, *y* und *z* betrachtet werden. Die Ausprägungen dieser Attribute sind beliebige numerische Werte. Ein Fall besteht aus einer Situationsbeschreibung und einer Diagnose, z. B.: „Fall1: (x = 2, y = 4, z = 1), Diagnose: A“.

Diesen Fall kann man dann als Punkt in einem Koordinatensystem darstellen. Um diesen Punkt herum legt man einen Raum, z. B. eine Kugel. Alle Punkte (Fälle), die innerhalb dieser Kugel liegen, nennt man nun *ähnlich*. Sucht man zu einer Situation

ABBILDUNG 4 Raum der ähnlichen Fälle um einen Fall



die passende Diagnose, bestimmt man dazu den Fall, der im Raum der Anfrage am nächsten liegt, und übernimmt dessen Diagnose.

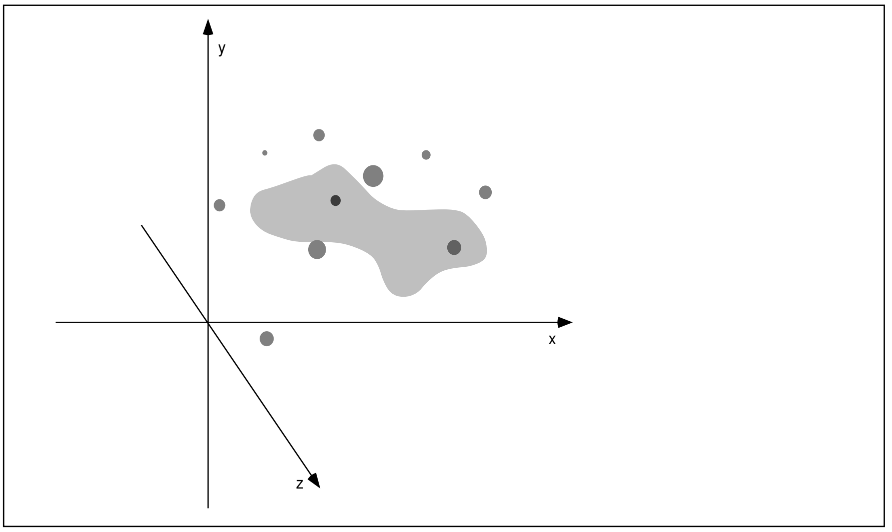
Es stellt sich also das Problem, den Abstand und damit die Ähnlichkeit zwischen zwei Punkten im Raum zu bestimmen. Dieses Ähnlichkeitsmaß ist eine Formel, die als Ergebnis einen Wert (üblicherweise zwischen Null und Eins) liefert. Durch Verändern von Koeffizienten in der Formel kann man den Raum um den Punkt (Fall) verformen (Abbildung 5, “Verändern des Ähnlichkeitsmaßes”).

Diese Formel rechet - grob vereinfacht dargestellt - die Anzahl der in Fall und Anfrage übereinstimmenden Symptomwerte gegen die Anzahl der unterschiedlichen Symptomwerte auf. PATDEX/2 verwendet ein Maß der Form:

$$sim\,(Sit, C) = \frac{\alpha|E|}{\alpha|E| + \beta|W| + \gamma|U| + \eta|N|}$$

Dabei bezeichnen:

ABBILDUNG 5 Verändern des Ähnlichkeitsmaßes



E: *Menge der erfüllten Symptome*. Symptome, die in Fall und Anfrage ähnlich sind, d. h. deren Abstand unter einem vorher festgesetzten Grenzwert liegt.

W: *Menge der widersprüchlichen Symptome*. Symptome sind widersprüchlich, wenn der Abstand über dem festgelegten Grenzwert liegt

U: *Menge der unbekannten Symptome*. Diese Symptome sind in der Fallbeschreibung gegeben, allerdings in der konkreten Situation noch nicht erhoben worden.

N: *Menge der redundanten Symptome*: Diese Symptome sind in der konkreten Situation erhoben worden, fehlen aber in der Fallbeschreibung.

α, β, γ und η sind Gewichte, durch die sich verschiedene Strategien zur Bestimmung der Ähnlichkeit verwirklichen lassen. In der Praxis hat sich für Diagnoseaufgaben die eher pessimistische Strategie mit: $\alpha = 1$, $\beta = 2$, $\gamma = -1/2$, $\eta = -1/2$ bewährt, ein Umschalten zwischen Strategien ist in PATDEX/2 möglich.

Eine allgemeine Form von Abstandsmaßen ist durch das Tverski-Kontrastmodell beschrieben, das in [Aamodt93] beschrieben ist.

PATDEX/2 erweitert diese Berechnung noch. In der oben angeführten Formel (aus dem ersten PATDEX-System) wurde jeweils nur mit der Anzahl von Symptomen gerechnet, beispielsweise wurde in $|W|$ Anzahl der Symptome betrachtet, in denen sich Fall und Situation unterscheiden. Das bedeutet aber, daß man alle Symptomabweichungen als gleich wichtig betrachtet. Außerdem wird auch nicht zwischen kleinen und großen Abweichungen unterschieden.

Das ändert sich in PATDEX/2. Statt der Anzahl von Symptomen wird hier eine gewichtete Summe betrachtet. Für jedes Paar voneinander abweichender Symptome wird die lokale Ähnlichkeit berechnet und dann mit einem für dieses Symptom individuellen Gewicht multipliziert. Das führt zu einem viel differenzierteren Ähnlichkeitsmaß.

Die individuellen Gewichte für die lokalen Ähnlichkeiten sind in einer Relevanzmatrix organisiert; hier findet sich zu jedem Symptom ein Gewicht, das von der Diagnose des Falls abhängt. Dadurch wird ausgedrückt, daß für eine bestimmte Diagnose manche Symptome wichtiger sind als die restlichen.

PATDEX kann nun die Relevanzmatrix selbst verändern, es kann also nicht nur Fälle lernen, sondern auch die Art, ihre Ähnlichkeiten zu bestimmen. Das ist nötig, da die Klassifikationsfähigkeit durch Aufnehmen von neuen Fällen nicht unbedingt besser werden muß.

Will man beispielsweise genau den Raum lernen, der in Abbildung 4 dargestellt ist, und hat man durch einen gegebenen Fall und ein geeignetes Ähnlichkeitsmaß diesen Raum wie in der Abbildung festgelegt, ist die Klassifikationsaufgabe gelöst. Würde man jetzt noch einen neuen Fall hinzunehmen, würden sich die beiden Ähnlichkeitsräume überlappen und weitere Fälle einschließen, die man eigentlich nicht als ähnlich bezeichnen möchte. Daher müßte jetzt das Ähnlichkeitsmaß verändert werden, um weiter richtig klassifizieren zu können.

Es gibt noch eine Fülle von anderen Ähnlichkeits- bzw. Distanzmaßen, die übersichtlich in [Wess91] dargestellt sind. Diese kurze Vorstellung von PATDEX/2 hat nur das „Herz“ des Systems, die Ähnlichkeitsbestimmung umrissen; auf die vielen weiteren Möglichkeiten von PATDEX/2 wie etwa Strategien zur Testauswahl kann hier nicht eingegangen werden, sie sind aber ausführlich in [Wess91] und [Wess93] erläutert.

1.4 Ansatzpunkte für Verbesserungen

1.4.1 Fallübertragung

Oft sind die Fallbasen, mit denen ein fallbasiertes System arbeiten muß, sehr klein im Vergleich zu der Menge aller denkbar auftretenden Fälle. Zum einen liegt das daran, daß die Fallbasis absichtlich klein gehalten wird, da der Suchaufwand linear mit ihrer Größe wächst (im ungünstigsten Fall müssen alle Fälle der Fallbasis betrachtet werden müssen, bevor eine Lösung gefunden wird[1]). Zum anderen sind oft einfach nicht mehr Fälle vorhanden.

Das Problem dabei wird an einem Beispiel deutlich:

1. Es gibt zwar Techniken (wie etwa KD-Bäume), die den Suchaufwand im durchschnittlichen Fall auf logarithmische Komplexität senken, aber auch sie haben linearen Suchaufwand im Worst-Case

Beispiel

Ein Diagnosesystem für das eigene Auto

Autos gehen heutzutage zum Glück relativ selten kaputt, so daß für das eigene Auto normalerweise kaum Fallbeispiele vorliegen. Die Zeitspanne, die vergeht, bis ein jedes der mehreren tausend Einzelteile einmal versagt, übersteigt die Lebensdauer eines Fahrzeuges um das Vielfache. Wollte man nur aufgrund der bisher aufgetretenen Defekte Diagnosen stellen, hätte man nicht genügend Fälle zur Verfügung.

Eine Möglichkeit zur Abhilfe ist einfach: Man erweitert die Fallbasis um Fälle von anderen Autos. Damit tritt aber ein neues Problem auf: Keine zwei Autos sind genau gleich. Sie unterscheiden sich in Baureihe, Ausstattung, verwendetem Material und vielem mehr. Für ein Diagnosesystem, das keine Fallanpassung durchführen kann, müßten alle diese Unterschiede aufgehoben werden, d. h. man würde die Autos auf einer geeignet hohen Abstraktionsebene beschreiben. Damit verliert das System dann aber wieder an Fähigkeiten, denn wenn das eigene Auto ganz spezielle „Macken“ hat, auf die man aufgrund der bisherigen Erfahrung mit seinem Auto sofort getippt hätte, gehen die Fälle, die diese „Macken“ beschreiben, unter den vielen anderen Fällen von anderen Autos unter.

Das gleiche Problem taucht anders gelagert wieder auf: Betrachten wir einen Automotor. Fall 1 beschreibt einen Defekt im ersten Zylinder, Fall 2 denselben Defekt im zweiten Zylinder. In unserem Wagen tritt nun ein Defekt im ersten Zylinder auf. Um jetzt darauf zu schließen, daß der erste Zylinder und nicht der zweite defekt ist, können wir nur Fall 1 benutzen, Fall 2 ist unähnlich. Sind wir jetzt der Lösung aber nähergekommen und wollen genau wissen, wo im ersten Zylinder der Fehler eigentlich steckt, können wir Fall 2 als ähnlichen Fall hinzuziehen. Genauso wollen wir auch die Fälle nutzen: Nicht für jeden denkbaren Defekt in jedem einzelnen der zwölf Zylinder unseres Wagens wollen wir Fälle aufnehmen; es genügt, wenn wir für jeden möglichen Zylinderdefekt einen Fall (egal in welchem Zylinder) kennen und ihn dann auf den aktuellen Zylinder übertragen können. Auch hier steht man bei Systemen ohne Fallanpassung vor dem Dilemma, hoch zu abstrahieren und alle Zylinder ununterscheidbar zu machen oder konkret und unterscheidbar zu bleiben und damit zwölfmal mehr Fälle für die Zylinder zu benötigen.

Das gleiche Problem wie für die Ähnlichkeit von Komponenten der Domäne stellt sich für die Ähnlichkeit von Meßwerten: Auch hier muß man abstrahieren, um Unterschiede auszugleichen, und auch hier können gerade diese Unterschiede in anderem Zusammenhang wichtig sein.

Fassen wir noch einmal zusammen:

- Um die Fallbasis klein zu halten oder um mit wenigen Fällen arbeiten zu können, ist Fallübertragung notwendig.
- Fallübertragung findet immer in einem gewissen Kontext statt, in dem man entscheiden kann, wann Fälle ähnlich sind und wann nicht. Dieser Kontext ändert sich während der Problemlösung.

- Das gleiche Problem stellt sich für die Ähnlichkeit von Meßwerten wie für die Ähnlichkeit von Komponenten selbst.

1.4.2 Ähnlichkeitsbestimmung

Ähnlichkeitsbestimmung ist von der Fallübertragung nicht eindeutig zu trennen, da Ähnlichkeitsmaß und Fallbasengröße ja in Beziehung zueinander stehen. Man kann immer behaupten, daß ein Ähnlichkeitsmaß nur deshalb einen unähnlichen Fall als ähnlich bewertet, weil der eigentlich ähnliche Fall nicht in der Fallbasis enthalten ist. Dem ist entgegenzuhalten, daß es einerseits sicher wünschenswert ist, wenn ein System lieber keine Antwort als eine falsche Antwort gibt und andererseits es nicht immer möglich und erwünscht ist, neue Fälle in die Fallbasis aufzunehmen. Ein System soll möglichst schnell in den Zustand gelangen, in dem es „ausgelernt“ hat und den Benutzer nicht damit vertröstet, den gerade gemachten Fehler nicht ein zweites Mal zu begehen.

Ähnlichkeitsbestimmung ist ein zentraler Punkt fallbasierten Vorgehens, denn mit dem oben angeführten Argument, daß man jeden Fall durch Aufnehmen in die Fallbasis lernen kann, wenn das Ähnlichkeitsmaß nicht das gewünschte Ergebnis liefert, kann man auch jede beliebige Datenbank zum fallbasiertes Expertensystem erheben.

Die Grundannahme des oben vorgestellten PATDEX/2-Ansatzes (und die von vielen anderen Lernverfahren, die aus einer Menge von Beispielen auf Lösungen schließen) ist eine Stetigkeitsanforderung:

Ähnliche Diagnosen (Klassen) sind ähnlich beschrieben, geringfügige Abweichungen in der Beschreibung haben nur geringfügige Auswirkungen auf die Klassifikation

Das stimmt in der Diagnose nicht immer, in anderen Bereichen wie der Planung normalerweise gar nicht. Ob diese Idee verwendbar ist, hängt davon ab, ob man eine geeignete Beschreibung finden kann, für die obige Behauptung gilt. Eine natürliche oder intuitive Beschreibung wird diese Voraussetzung nicht immer erfüllen, es ist fraglich, ob man sie überhaupt immer angeben kann, ohne das Problem bereits gelöst zu haben.

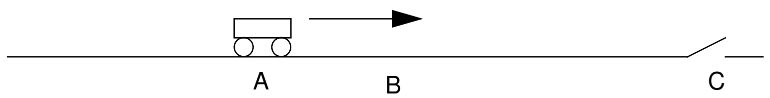
Wenn wir beispielsweise eine technische Domäne so modelliert haben, wie wir das oben getan haben, kann ein Fall eine ganz neue Bedeutung dadurch erhalten, daß ein einzelner Meßwert sich ändert, beispielsweise, daß ein Schalter geschlossen ist, wo er vorher geöffnet war - alle anderen Meßwerte können dann eine andere Bedeutung haben.

Im Bereich der Planung beschreibt man eine Aufgabe in der Regel dadurch, daß man einen gewünschten Anfangs- und Endzustand angibt. Gesucht sind dann Aktionen, die den Anfangszustand in den Zielzustand überführen. Hier kann eine kleine Änderung eines dieser Zustände einen völlig anderen Lösungsweg erforderlich machen. Deutlich wird dies am bekannten Roboter-Beispiel: Gegeben sind eine Menge von Räumen, die durch Türen miteinander verbunden sind. Die Aufgabe besteht darin, einen Roboter von einem Startraum in einen Zielraum zu bewegen.

Auch hier kann eine kleine Änderung wie das Verschließen einer Tür dafür sorgen, daß der neue Weg zum Ziel ein ganz anderer ist als der alte.

In PATDEX/2 wird versucht, diesem Problem durch die Relevanzmatrix zu begegnen. In dieser Matrix wird zu jeder Diagnose und jedem Meßpunkt ein Wert eingetragen, der die Relevanz dieses Meßpunkts für die Diagnose repräsentieren soll. Das setzt voraus, daß die Relevanz eines Meßpunkts nur von der Diagnose abhängig ist. Daß dem nicht so ist, zeigt ein Gegenbeispiel in Abbildung 6: Ein Wagen fährt auf einer Schiene. Wenn er an Punkt C angekommen ist, betätigt er einen

ABBILDUNG 6 Der Wagen auf der Schiene



Schalter. Nehmen wir an, der Wagen sei defekt. Wenn wir den Wagen von A nach B bewegen wollen, er sich aber nicht bewegt, betätigt er auch den Schalter nicht. Das ist jedoch nicht relevant, da er auch im Falle des Funktionierens den Schalter nicht betätigt hätte. Wollen wir den Wagen jedoch auf Position C bewegen, ist die Tatsache, daß der Schalter nicht geschlossen ist, ein starker Hinweis auf den Defekt des Wagens und durchaus relevant. Man sieht also, daß auch die Relevanz von Symptomwerten nicht nur von der Diagnose allein abhängt, sondern auch wieder vom Zusammenhang, in dem das Symptom auftritt. Daher kann man Symptomen, die am Schalter aufgenommen werden, keine feste Relevanz in der Relevanzmatrix zuweisen; Relevanz erhält ein solches Symptom erst in Abhängigkeit von anderen Symptomen (wie etwa der Position des Wagens).

Man könnte an dieser Stelle argumentieren, daß man diesen Konflikt dadurch beheben kann, daß man statt der Diagnose „Wagen defekt" zwei Diagnosen: „Wagen mitten auf Schiene defekt" und „Wagen am Ende der Schiene defekt" einführt. Es werden Teile der Situation, in dem der Störfall aufgetreten ist, zusätzlich in die Diagnose codiert. Dadurch entstehen mehrere Diagnosen, obwohl ein identischer Fehler aufgetreten ist. Dieses Vorgehen führt dann dazu, daß viel mehr Fälle benötigt werden, da ein Fehler üblicherweise in verschiedenen Situationen auftreten kann. Es soll aber gerade vermieden werden, die Fallbasis aufzublähen.

Eine weitere Voraussetzung für die Verwendung eines Ähnlichkeitsmaßes wie dem in PATDEX/2 ist es, daß die Symptomwerte in einer Art geordnet sind, die ihrer Bedeutung in der Domäne gerecht wird. Sind diese Werte ungeordnet oder hat die Ordnung keine inhaltliche Bedeutung, macht es keinen Sinn, einen Abstand bzw. eine Ähnlichkeit über ihre Entfernung im Raum definieren zu wollen. Solche ungeordneten Werte können aber häufig auftreten; oft ist es auch so, daß der Wertebereich nur aus zwei Werten besteht. Man kann dann zwar definieren, daß „aus" kleiner als „an" ist, die Ordnung ist aber normalerweise nicht sinnvoll im Ähnlichkeitsmaß verwendbar.

Fassen wir zusammen:

- Ähnlichkeitsmaße wie in PATDEX/2 sind nur sinnvoll, wenn ähnliche Objekte durch ähnliche Beschreibungen repräsentiert sind. Das bedeutet in dem oben vorgestellten Raummodell, daß dort nahe beieinanderliegende Punkte zur gleichen Klasse gehören.
- Diese Voraussetzung ist nicht bei jeder Beschreibung von vornherein gegeben, oft ist eine solche Beschreibung nicht einfach zu finden.

1.4.3 Erklärungsfähigkeit

Wenn ein System zu einer Lösung gekommen ist, ist es wünschenswert, wenn diese Lösung auch dem Benutzer erklärt werden kann. Fallbasierte Systeme haben es dabei relativ einfach, sie geben den ähnlichsten Fall aus der Fallbasis an. Allerdings sollten sie dabei auch begründen können, warum der gefundene Fall zu der Anfrage ähnlich ist. Gerade bei einfachen Domänen ist das zwar in der Regel nicht nötig, da Ähnlichkeiten dort offensichtlich sind, bei komplexeren Domänen ist aber eine weitergehende Begründung erforderlich. Die bloße Angabe des Ähnlichkeitswerts - also einer Zahl zwischen Null und Eins - ist dann sicher nicht mehr ausreichend.

1.5 Modellbasierte Ansätze

In den letzten Abschnitten wurde eine Vorgehensweise vorgestellt, die versucht, aus gegebenen Fallbeispielen auf Diagnosen für neue Situationen zu schließen. Dabei läßt man aber Wissen über die Domäne ungenutzt, das relativ einfach zu erheben und zu verarbeiten ist. Technische Domänen lassen sich im allgemeinen gut als simulationsfähige Modelle beschreiben, was zu Ansätzen der modellbasierten Diagnose geführt hat. Auf deren grundsätzliche Vorgehensweise soll hier nur kurz eingegangen werden, da im wesentlichen nur das in ihnen verwandte Wissen für den Ansatz in MoCAS/2 interessant ist

Da man gerade in technischen Domänen das Verhalten von Komponenten sehr genau kennt und sie gut beschreiben und simulieren kann, kann man - von diesem sicheren Domänenwissen ausgehend - sichere Schlüsse ziehen. Die Vorgehensweise dabei ist grob skizziert die folgende: Man versucht, die möglichen Fehlverhalten aller Bauteilen zu modellieren. Tritt ein Störfall auf, versucht man, über Simulation eines jeden Fehlverhaltens eines jeden Bauteils herauszufinden, ob die beobachteten Symptome durch dieses Verhalten hervorgerufen worden sein könnten.

Dabei steht man von mehreren Problemen:

- Alle möglichen Fehlverhalten sind nicht bekannt. Man kann zwar sehr gut beschreiben, wie sich ein Bauteil im Normalfall verhalten sollte (dafür ist es ja konzipiert), die möglichen Fehlverhalten sind jedoch unüberschaubar. Daher wird man nur diejenigen Fehler modellieren können, die bekannt sind, und dadurch bisher unbekanntes Fehlverhalten nicht finden können.
- Der Suchraum ist enorm groß, die Suche ist aufwendig. Selbst wenn man nicht „blind“ für jedes Bauteil alle möglichen Fehlverhalten durchprobiert, sondern sich Schritt für Schritt von den Symptomen zu möglichen Ursachen zurückarbeitet, steht man bei jedem Schritt zurück vor einer großen Menge von alternativen Begründungen für das jeweilig betrachtete Symptom. Diese Alternativen

müssen dann jeweils wieder weiter zurückverfolgt werden, wobei sich eine Baumstruktur ergibt.

- Da durch die beobachteten Symptome eine Ursache nicht determiniert ist, gelangt man zu verschiedenen Hypothesen. Zwischen diesen Hypothesen muß wiederum eine Auswahl getroffen werden, welche von ihnen als wahrscheinlichste Fehlerursache in Frage kommt und daher zuerst überprüft werden sollte.

Hier erkennt man schnell, daß man Erfahrungen mit der Maschine benötigt, adäquate Heuristiken, nach denen man auf potentielle Kandidaten und deren Fehlverhalten schließen kann. Dieses Wissen steht uns zur Verfügung - in Form von Fällen.

KAPITEL 2

Informelle Beschreibung von MoCAS/2

Wie wir im vorigen Kapitel gesehen haben, ist eine Ähnlichkeitsbewertung wie PATDEX/2 sie durchführt, nicht immer sinnvoll, da sie eine Art Clustering über den Attributwerten durchführt. Es ist jedoch nicht immer gegeben, daß ähnliche Fälle tatsächlich Cluster bilden, ein Beispiel dafür war die Modellierung von technischen Domänen, so wie sie in der Einführung vorgestellt wurde. Weiterhin war es nicht möglich, Fälle und deren Lösungen variabel auf verschiedene Situationen zu übertragen. Ein anderes Problem war auch noch, daß nicht zufriedenstellend begründet werden konnte, warum ein Fall zu einer Situation ähnlich ist.

Im Folgenden soll nun ein Ansatz vorgestellt werden, der die Ähnlichkeitsbewertung vornimmt, indem er zusätzliches (Hintergrund-) Wissen verwendet und damit die oben angeführten Probleme löst. Dieser Ansatz wurde im System MoCAS/2 (Model-Based Case Adaptation System) realisiert, das in späteren Kapiteln vorgestellt wird. Es baut auf Ergebnissen auf, die in einer ersten Implementierung [PewsWeiler92] gemacht wurden.

Zunächst werden die Problemstellung und der Lösungsansatz grob skizziert. Danach wird dargestellt, wie man zu einer Ähnlichkeitsbewertung für Fälle kommen kann, indem man auf einer Ähnlichkeitsbewertung für einzelne Komponenten und ihrem Verhalten aufbaut. Dabei wird auch auf die Aspekte Fallübertragung und Fallanpassung eingegangen.

2.1 Idee

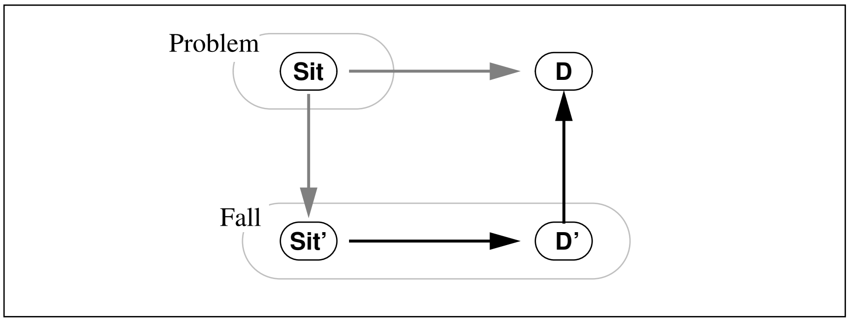
2.1.1 Die Problemstellung

Ein typisches Szenario bei der fallbasierten Diagnostik in einer technischen Domäne (wie etwa bei der Autoreparatur) stellt sich so dar: Gegeben sind eine Menge von Symptomen, die die Situation beschreiben in der sich das zu diagnostizierende Gerät befindet (z. B.: Licht brennt nicht, Motor dreht nicht, Radio funktioniert, Anlasser dreht). Gesucht ist dazu eine Diagnose, die das durch die Symptome beschriebene Verhalten erklärt. In einer Fallbasis sind Fälle gegeben, in denen zu früher aufgetretenen Situationen die damals gefundenen Diagnosen beschrieben sind. Man versucht also eine Lösung zu finden, indem man die aktuelle Situation mit den Fällen in der Fallbasis vergleicht und die Diagnose eines passenden Falls nutzt. Wie wir aber im letzten Kapitel gesehen haben, ist man durch einen rein syntaktischen Vergleich nicht immer in der Lage, sinnvoll Ähnlichkeiten zu bestimmen.

2.1.2 Die Lösung

Die Vorgehensweise beim fallbasierten Schließen ist noch einmal in Abbildung 7 dargestellt: Zu einem Problem Sit (Situation) ist eine Lösung D (Diagnose) gesucht (schraffierter Pfeil). Man sucht also nach einem ähnlichen Problem Sit' und der dazu bekannten Lösung D', um dieses D' dann zur Lösung des aktuellen Problems zu verwenden. Dabei ist der grau gezeichnete Pfeil nicht immer auffindbar. Auch ist nicht klar, wie diese Abbildung von Sit auf Sit' wieder rückwärts durchgeführt werden kann, um D aus D' zu erhalten. In MoCAS/2 werden diese Probleme durch die Nutzung von Modellwissen gelöst.

ABBILDUNG 7 Das Vorgehen beim Fallbasierten Schließen

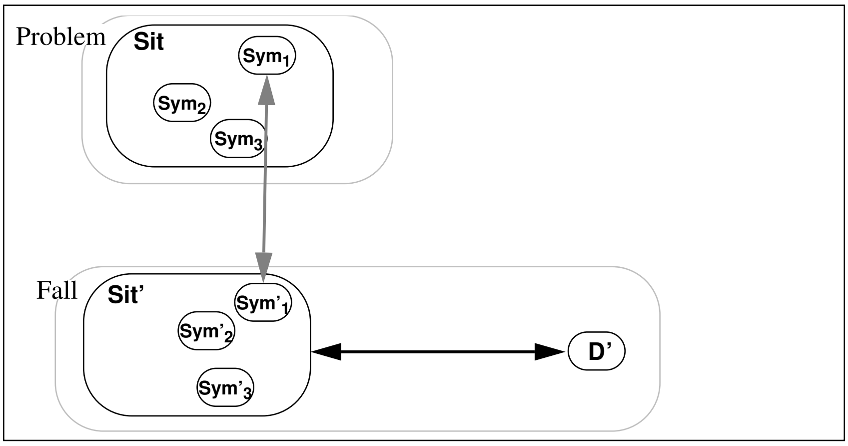


Die grundsätzliche Vorgehensweise in MoCAS/2 ist die: Zu den Fällen der Fallbasis wird eine Erklärung aufgebaut, die begründet, warum die dort beschriebene Diagnose die beobachteten Symptome hervorgerufen hat. Wenn eine Anfrage gestellt wird, wird versucht, bestimmte Teile dieser Erklärung auf die Anfragesituation zu übertragen. Um den Umfang dieser Übertragung festzulegen und um von vornherein nur möglichst erfolgversprechende Fallübertragungen durchzuführen, wird in der Situationsbeschreibung nach Anhaltspunkten dafür gesucht.

Eine Situationsbeschreibung besteht aus einer Menge von Symptomen. Einige dieser Symptome sollen als Anhaltspunkte dienen, um die Zuordnung zwischen Sit und Sit' zu finden. Man sucht in der Problembeschreibung Sit nach Anhaltspunkten, durch die Fälle mit ähnlichen Anhaltspunkten gefunden werden können. Das ist in Abbildung 8 dargestellt.

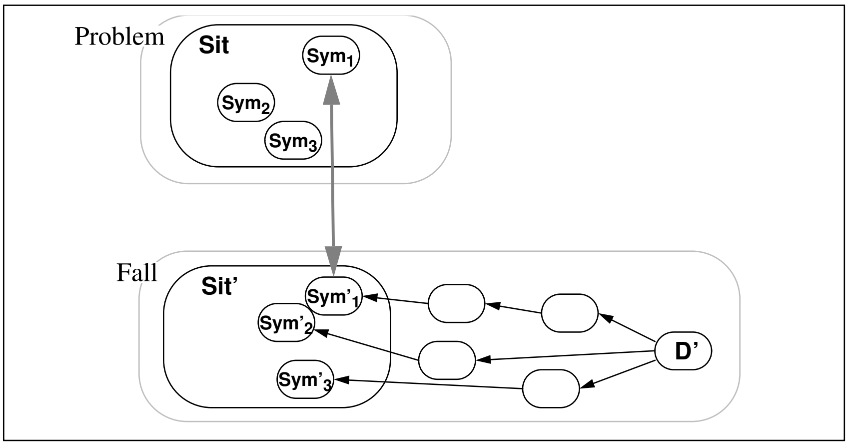
Solche Anhaltspunkte erhält man dadurch, daß das Verhalten der in Sit beschriebenen Bauteile analysiert wird. Zu jedem Bauteil wird festgestellt, ob das von ihm gezeigte Verhalten mit dem von ihm erwarteten Verhalten übereinstimmt. Ist das

ABBILDUNG 8 Zuordnung zwischen Symptomen in Situation und Fall



nicht der Fall, ist dieses unerwünschte Verhalten ein Anhaltspunkt. Auch in den Fallbeschreibungen lassen sich auf diese Art Anhaltspunkte finden. Gleichartige Anhaltspunkte liefern also eine Zuordnung zwischen Symptomen in Sit und Sit'. Das ist in der Abbildung durch den grauen Pfeil dargestellt.

ABBILDUNG 9 Die Wirkungen der Diagnose auf die Symptome

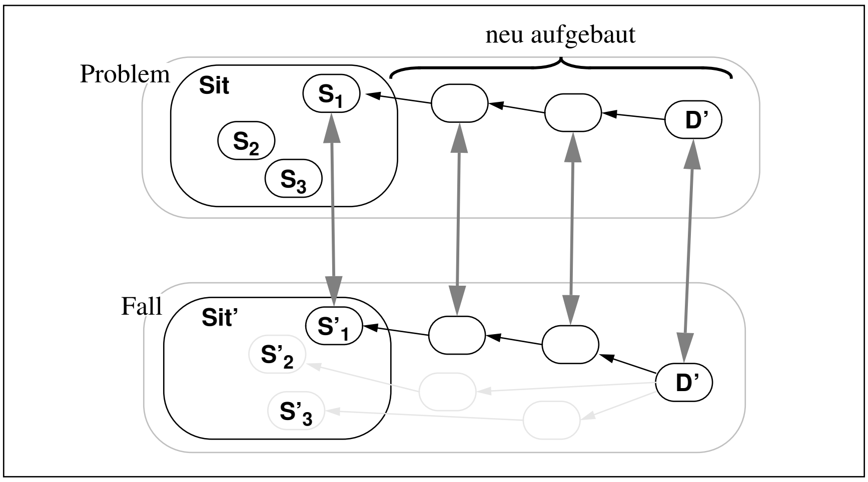


Diese Zuordnung zwischen Bauteilen in Sit und Sit' wird nun ausgeweitet, bis ein Bauteil in Sit identifiziert ist, das dem Bauteil entspricht, welches D' beschreibt. Das ist allerdings nur möglich, wenn die Auswirkungen von D' auf die Symptome in Sit' bekannt sind. In der Fallbeschreibung werden also Wirkungsketten von der Diagnose zu den beobachteten Symptomen gesucht. Ein Beispiel für eine solche Kette ist:

Batterie leer (Diagnose)
⇒ Anlasser dreht nicht
⇒ Motor dreht nicht
⇒ Auto fährt nicht.(Symptom)

In Abbildung 9 sind diese Ketten durch die schwarzen Pfeile angedeutet. Jeder Pfad von D’ zu einem Symptom in Sit’ soll eine solche Kette darstellen. Eine besonders interessante Wirkungskette ist die, die zu dem Symptom führt, das wir als Anhaltspunkt für den Fall ausgemacht hatten. Sie wird jetzt rückverfolgt, so wie es in Abbildung 10 dargestellt ist. Man geht von der Zuordnung zwischen Sym_1 und Sym'_1 aus und baut die Folge zwischen Sym'_1 und D’ nach, wobei man mit Sym_1 startet.

ABBILDUNG 10 Das Vorgehen in MoCAS/2



Hier kann man auch erkennen, daß nicht alle Symptome, die in der Situationsbeschreibung angegeben sind, an der Lösungsfindung beteiligt sind, sondern nur einige wenige, nämlich die, für die Anhaltspunkte gefunden worden waren. Genauso werden auch nicht alle Wirkungsketten im Fall in ihrer vollen Länge übertragen.

Der Ansatz in MoCAS/2 führt zu einer Verschmelzung von fallbasiertem und modellbasiertem Schließen. Das Modellwissen wird benutzt, um Wirkungsketten aufzubauen und Anhaltspunkte zu finden. Diese Anhaltspunkte liefern einerseits einen Hinweis auf erfolgversprechende Fallübertragungen, andererseits legen sie den Umfang fest, in dem die Wirkungsketten vom Fall auf die Problemstellung übertragen werden.

Im Gegensatz zu vielen anderen integrierten Systemen besteht MoCAS/2 nicht aus zwei separaten Moduln, die - ein jedes für sich - fallbasiert oder modellbasiert das Problem zu lösen versuchen, sondern hier unterstützt Modellwissen die drei Schritte Retrieve, Retain und Reuse.

Dieses zusätzliche Wissen ist gerade in technischen Domänen relativ einfach zu erheben, da bei der Konstruktion von Maschinen genaue Spezifikationen über Bauteile und Bauteilverhalten vorliegen. Oft wird die Konstruktion selbst auch rechnerunterstützt vorgenommen.

Bevor der Ansatz in MoCAS/2 im Detail erläutert wird, soll im nächsten Abschnitt näher darauf eingegangen werden, wie dieses Modellwissen beschaffen ist und wie es repräsentiert wird.

2.2 Wissen aus dem simulationsfähigen Maschinenmodell

Eine Komponente einer Maschine, ein Bauteil, wird beschrieben durch:

1. Ein- und Ausgänge und deren Wertebereiche[1]
2. Internen Aufbau (bei Baugruppen, die aus anderen Bauteilen zusammengesetzt sind)
3. Funktionsweise

Zu 1.: Die Ein- und Ausgänge eines Bauteils werden auch als *Ports* oder Schnittstellen bezeichnet, z. B. die beiden Enden eines Kabels. Man bezeichnet sie mit `ende1` und `ende2` und kommt so zu einer Frame-artigen Darstellung:

```
Kabel
    ende1
    ende2
```

Über diese Schnittstellen ist das Bauteil mit anderen Bauteilen verbunden und an ihnen sind Meßwerte abgreifbar. Da man im Allgemeinen nicht die Richtung einer solchen Schnittstelle angeben kann (bei einem Kabel gibt es ja schließlich keinen Ein- oder Ausgang), wollen wir diese Unterscheidung im Folgenden auch nicht treffen.

Die Meßwerte, die an den Ports abgegriffen werden können, liegen in einem dem Port zugeordneten Wertebereich, z. B. bei dem Kabel in einem Bereich für die Stromstärke von I = [0A..10A].

Zu 2.: Eine Baugruppe ist aus mehreren anderen Bauteilen zusammengesetzt, die untereinander verbunden sind. Ports einer Baugruppe leiten ihre Werte nach innen an die Ports ihrer einzelnen Unterbauteile weiter.

Zu 3.: Die Funktionsweise eines Bauteils kann man durch Funktionen beschreiben, die angeben, welche Werte bestimmte Ports in Abhängigkeit von anderen Ports annehmen. Bei dem Kabel im Beispiel sind diese Funktionen leicht anzugeben:

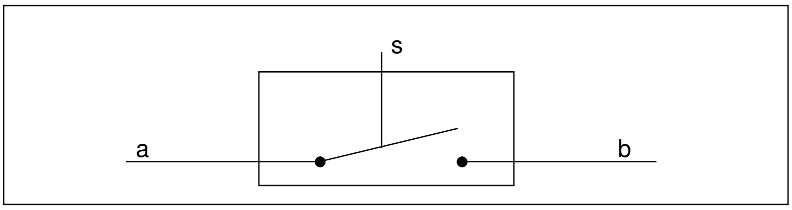
$f_{ende1}(ende2) = ende2$

$f_{ende2}(ende1) = ende1$

1. Auf interne Zustände (States) von Bauteilen soll hier nicht näher eingegangen werden, da sie prinzipiell durch einen rückgekoppelten Eingang realisierbar sind. Daher gelten die Überlegungen, die für Aus- und Eingänge gemacht werden, analog auch für die Zustände

Solche Funktionen können aber auch Fallunterscheidungen beinhalten, beispielsweise, wenn man einen Schalter modellieren will, so wie er in Abbildung 11 dargestellt ist. Sobald der Knopf s betätigt wird, werden a und b verbunden, der Schalter

ABBILDUNG 11 Modellierung eines Schalters



schließt. Die Ports dieses Schalters sind dann: s mit der Wertebereich W_s = {gedrückt, nicht_gedrückt} sowie a und b mit dem Wertebereich $W_a = W_b$ = [0A..10A]. In unserer Framedarstellung sieht das dann so aus:

```
Schalter
    a [0A..10A]
    b [0A..10A]
    s {gedrückt, nicht_gedrückt}
```

Wir können jetzt Funktionen angeben, die das Verhalten des Schalters beschreiben:

$$f_a(b,s) = \begin{cases} b, \text{ falls } s = \text{gedrückt} & \text{Verhalten: „schließen“} \\ 0, \text{ falls } s = \text{nicht\_gedrückt} & \text{„öffnen“} \end{cases}$$

$$f_b(a,s) = \begin{cases} a, \text{ falls } s = \text{gedrückt} & \text{Verhalten: „schließen“} \\ 0, \text{ falls } s = \text{nicht\_gedrückt} & \text{„öffnen“} \end{cases}$$

Da s nicht von a und b abhängig ist, müssen wir auch keine Funktion $f_s(a,b)$ modellieren. Der Schalter hat also zwei Verhaltensweisen: „öffnen“ und „schließen“.

In der Praxis bietet es sich an, die Funktionsweise eines Bauteils durch Regeln oder Constraints zu beschreiben, besonders wenn eine qualitative Modellierung gewählt wurde und die Wertebereiche dann keine Intervalle wie [0A..10A], sondern nur noch symbolische Aufzählungen wie {an, aus} sind. Dabei wird man dann jede Verhaltensweise durch eine Regel modellieren. Weitere Erläuterungen zur Modellierung von technischen Domänen sind auch in [Schuch92] zu finden.

2.3 Ähnlichkeit von Bauteilen und deren Verhalten

Nachdem das Wissen aus dem Maschinenmodell vorgestellt ist, soll jetzt gezeigt werden, wie man dieses Wissen strukturieren kann, um damit dann die Ähnlichkeit von einzelnen Bauteilen zu beurteilen. Zusätzlich wird auch eine Ähnlichkeitsbewertung für das Verhalten der Bauteile vorgestellt. Diese beiden Bewertungen kann man dann nutzen, um Ähnlichkeit zwischen Fall und Situation zu berechnen. Wir beginnen mit der Ähnlichkeit von Bauteilen an sich.

Die Idee hierbei ist es, Bauteile in Abstraktionshierarchien zu organisieren, wobei Bauteile als ähnlich betrachtet werden, falls sie eine gemeinsame Abstraktion besitzen.

2.3.1 Abstraktion von Bauteilen

Ein Beispiel für eine Abstraktionshierarchie in technischen Domänen ist in

ABBILDUNG 12 Abstraktionsgraph

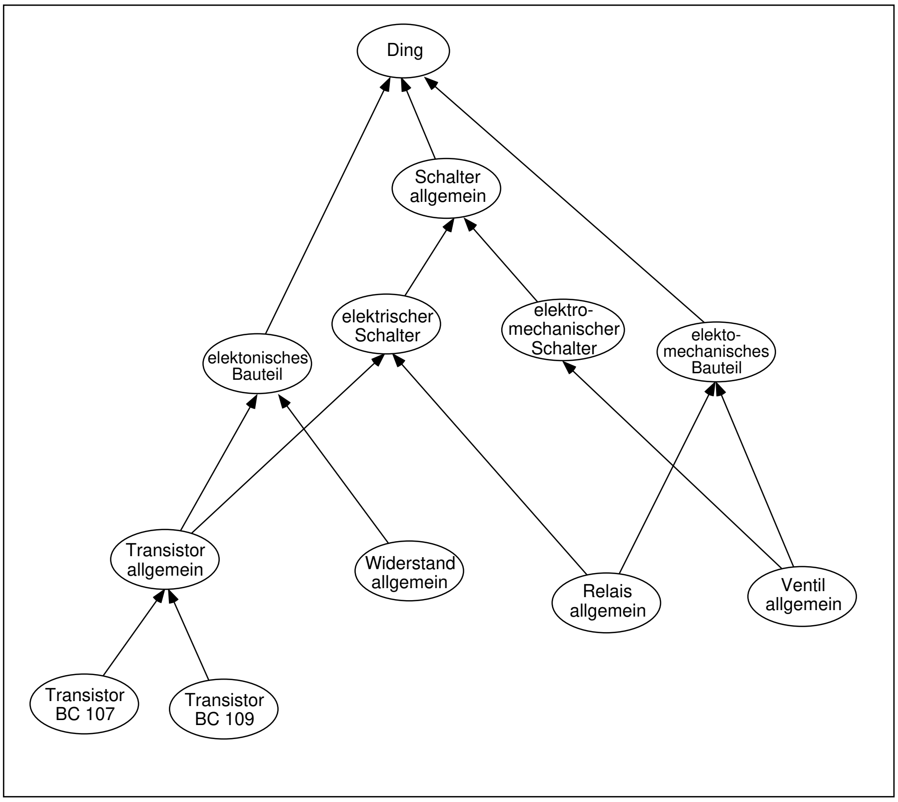
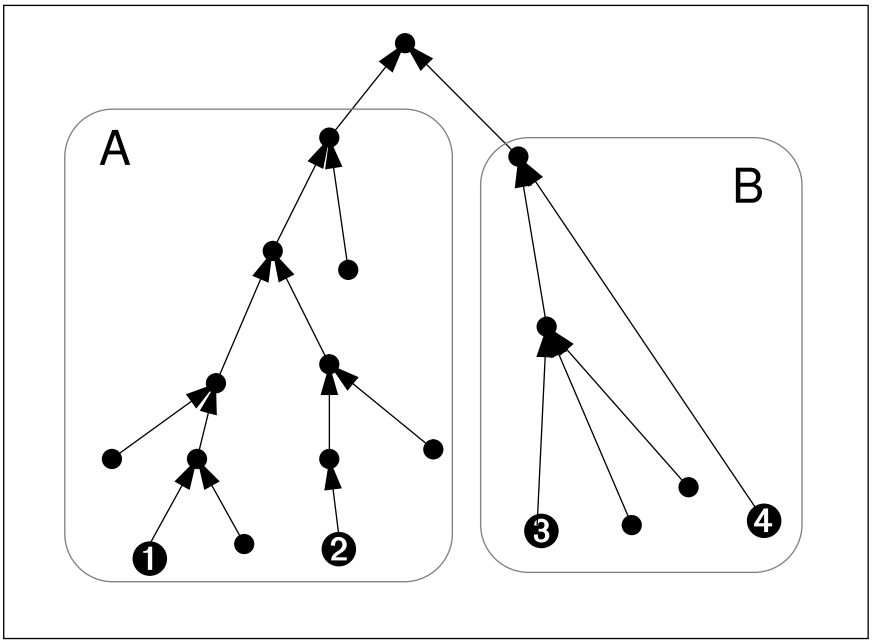


Abbildung 12 gezeigt. Ein Transistor vom Typ `BC 107` (links unten) kann auf verschiedenen Abstraktionsebenen betrachtet werden: Als Transistor allgemein, als elektrischer Schalter oder ganz einfach als ein Ding. Ein Transistor vom Typ `BC 109` ist ähnlich, da sie beide die gleiche Abstraktion (`Transistor allgemein`) besitzen. Hier erkennt man auch die Teilordnung über den Ähnlichkeiten: Ein Transistor ist zwar auch zu einem Ventil ähnlich (`Schalter allgemein`), Transistoren sind einander aber ähnlicher, da sie auf niedrigeren (konkreteren) Abstraktionsebenen Vaterknoten besitzen. Ob allerdings ein Relais zu einem Ventil ähnlicher ist als ein Transistor zu einem Widerstand, kann aus dem Diagramm nicht hergeleitet werden; die Abstraktionen sind nicht in Ebenen angeordnet.

Man könnte hier natürlich die Ähnlichkeit über die Anzahl der Knoten definieren,

ABBILDUNG 13 Abstände im Abstraktionsgraphen



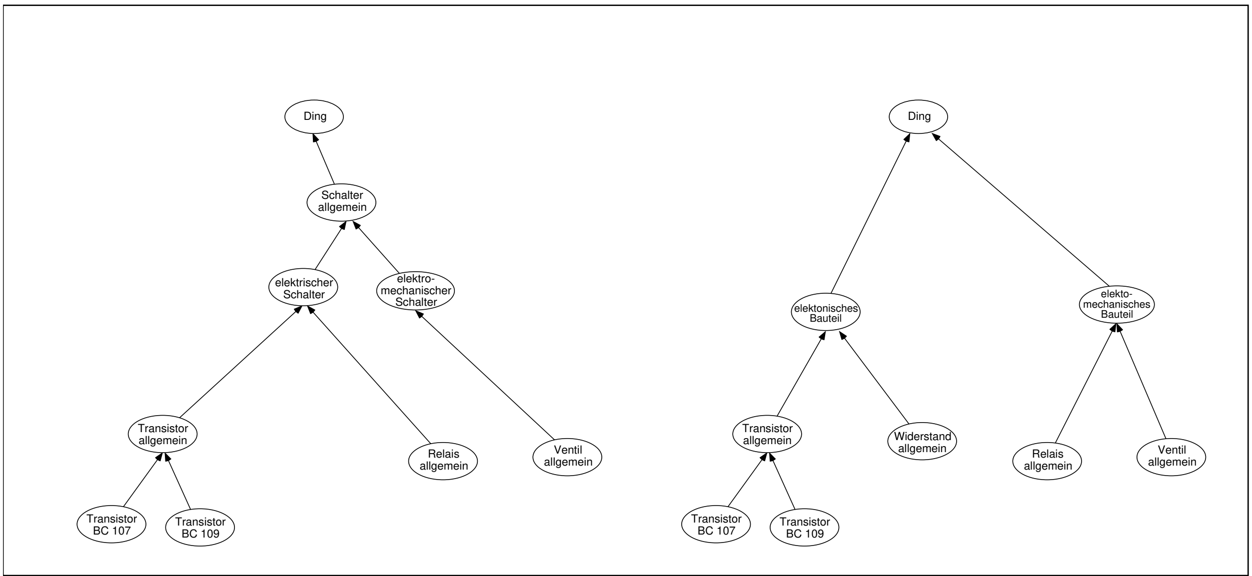
die bis zur gemeinsamen Abstraktionsebene übersprungen werden müssen. Das hätte dann aber unerwünschten Effekte, die in Abbildung 13 erkennbar sind: Bei besonders gut verstandenen Abstraktions-Teilgraphen sind viele Abstraktionsabstufungen bekannt, also werden viele Zwischenknoten eingeführt (Teilgraph A). Weniger gut durchdrungene Abstraktionsbeziehungen können nur mit weniger Zwischenknoten dargestellt werden (Teilgraph B). Zählt man nun nur die Zwischenknoten um den Abstraktionsgrad zu beurteilen, würden gut verstandene Teilgraphen benachteiligt. Der Abstand zwischen den Knoten 1 und 2 wäre größer als der zwischen den Knoten 3 und 4. Deshalb werden in MoCAS/2 solche Ähnlichkeitsvergleiche nicht unterstützt.

Glücklicherweise wird ein solcher Vergleich von Abständen auch nicht benötigt; er erscheint Menschen auch eher unnatürlich. (Ist ein Apfel einer Banane ähnlicher als ein Stuhl einem Tisch?)

Als nächstes fällt auf, daß der Graph in Abbildung 12 keine Hierarchie darstellt, Knoten haben mehrere Abstraktionen. Das liegt daran, daß der Begriff „Abstraktion“ immer unter einem bestimmten Aspekt gebraucht wird. So ist eine Abstraktion von `Transistor allgemein` unter dem Aspekt „Funktionsweise“ der `elektrischer Schalter`, unter dem Aspekt „Bauteilart“ hingegen `elek-`

`tronisches Bauteil`. Wir haben es hier also mit einer Überlagerung von

ABBILDUNG 14 Dekomposition des Abstraktionsgraphen in einzelne Abstraktionshierarchien



zwei Hierarchien zu tun, die in Abbildung 14 aufgelöst wird. Für den praktischen Einsatz ist es so, daß nur ein Abstraktionsaspekt in einem Problemlösungsschritt Anwendung findet. Für die Hypothesengenerierung bei der Diagnose wird nur der Aspekt „Funktionsweise“ benötigt, daher können wir uns im Folgenden auf eine Hierarchie beschränken. Auch in MoCAS/2 ist diese Einschränkung getroffen.

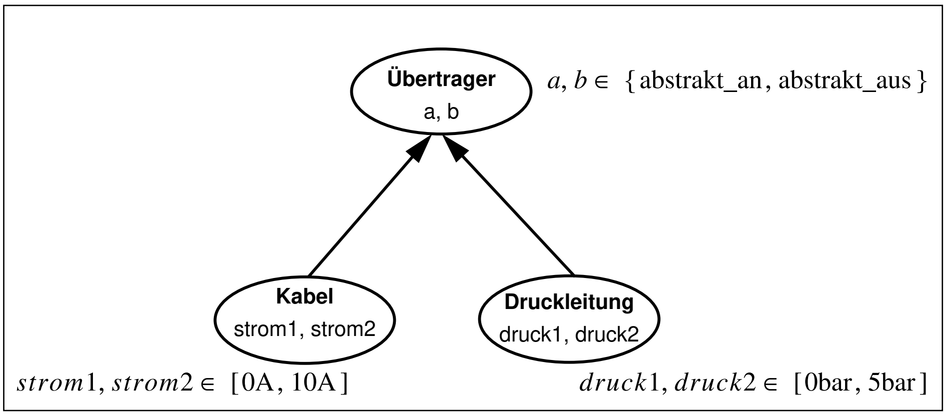
2.3.2 Abstraktion von Ports

Die Kanten des Abstraktionsgraphen beschreiben eine Abstraktionsabbildung für den funktionalen Aspekt. Die Funktion von Bauteilen kann man als Zustandsänderung beschreiben; die Zustände wiederum beschreibt man - wie im vorigen Kapitel erläutert - durch die Angabe von Werten an den Ports einer Komponente. Wenn wir die Bauteile nun abstrahieren, müssen wir auch Abbildungen für die Ports und deren Wertebereiche angeben, wie im folgenden Beispiel deutlich wird.

Beispiel **Abstraktion von Stromkabel und Druckleitung**

In Abbildung 15 ist gezeigt, wie zwei Bauteile und ihre gemeinsame Abstraktion modelliert sind: Ein Stromkabel, das die Ports strom1 und strom2 besitzt und eine Druckleitung, mit den Ports druck1 und druck2. Die Wertebereiche für das Kabel seien I = [0A..10A] und für die Leitung P = [0bar..5bar]. Das abstrakte Bauteil Übertrager habe nun die Schnittstellen `a` und `b` mit dem Wertebereich W = {`abstrakt_an`, `abstrakt_aus`}.

ABBILDUNG 15 Abstraktionshierarchie im Beispiel



Als Frames:

```
Kabel
    strom1 I=[0A..10A]
    strom2 I=[0A..10A]
Druckleitung
    druck1 P=[0bar..5bar]
    druck2 P=[0bar..5bar]
Übertrager
    a W={abstrakt_ein, abstrakt_aus}
    b W={abstrakt_ein, abstrakt_aus}
```

Die Abstraktionsabbildungen sind dann zwei Funktionen, die I bzw. P auf W abbilden. Bei dieser Abstraktionsabbildung gehen natürlich Informationen über die ursprünglichen Werte verloren. Es können weiterhin auch Ports zusammengefaßt werden; man kann sich ein Bauteil Datenbus mit 32 Ports vorstellen, die für eine Betrachtung auf einer höheren Abstraktionsebene zu einem Port zusammengefaßt werden.

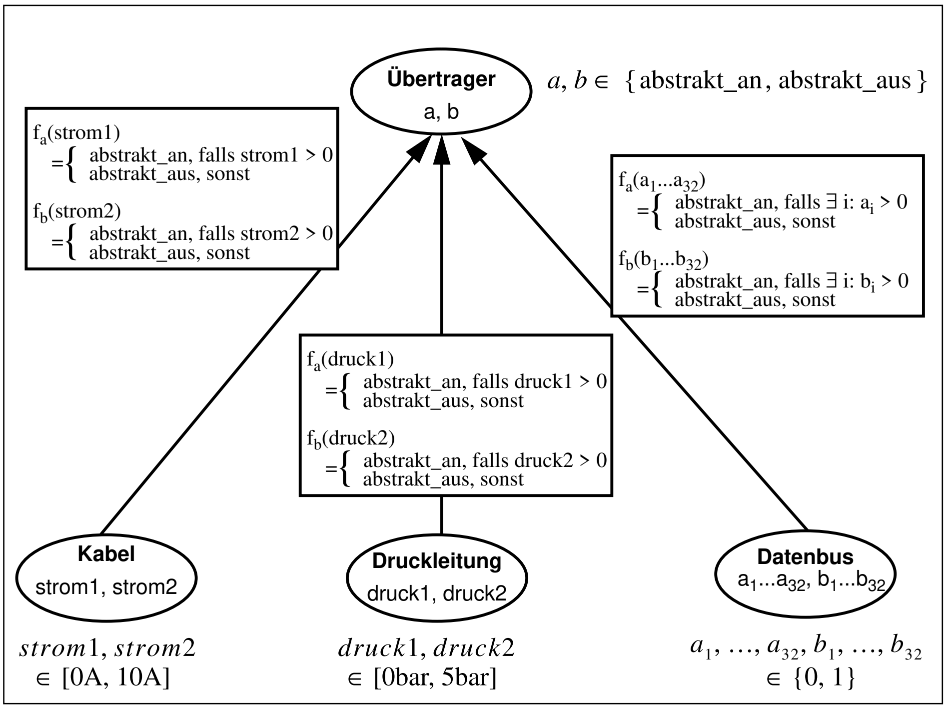
Durch diese Darstellung kann man Bauteile und die Werte, die an ihren Ports anliegen, auf verschiedenen Abstraktionsebenen betrachten.

Die in diesem Abschnitt beschriebenen Gedanken sind als Beispiel noch einmal in Abbildung 15 zusammengefaßt.

2.3.3 Abstraktion von Funktion und Verhalten

In der Modellbeschreibung der Domäne ist noch eine Funktionsbeschreibung der einzelnen Bauteile gegeben. Das ist im allgemeinen Fall eine Funktion, die die Ein-

ABBILDUNG 16 Abstraktionen für Ports



gabewerte auf Ausgabewerte abbildet. Für unser Kabel im obigen Beispiel war das die Identitätsfunktion:

strom1 = f_{strom1}(strom2) = strom2.

Analoge Funktionen kann man sich für die Druckleitung und den Übertrager denken, sie enthalten jeweils nur ein einziges Verhalten, nämlich „strom weiterleiten“, im allgemeinen Fall können aber auch mehrere Verhaltensweisen beschrieben sein.

Auch Verhaltensweisen sollen in der Abstraktionshierarchie aufeinander abgebildet werden, wir benötigen also wieder Abstraktionsabbildungen, die ein konkretes Verhalten einem abstrakten Verhalten zuordnen. Diese Abbildungen müssen mit den zuvor festgelegten Abbildungen für Wertebereiche und Ports verträglich sein. In unserem Beispiel von Leitung und Übertrager hieße das: Die Abstraktionsabbildung für Verhalten ordnet dann dem Verhalten „strom weiterleiten“ der Leitung einem entsprechenden Verhalten „weiterleiten“ des Übertragers zu.

Auch bei dieser Abbildung können wieder einzelne Verhaltensweisen zu abstrakteren Verhaltensweisen zusammengefaßt werden, so daß auch hier Information verlorengeht.

Anmerkung: Die Hierarchie von Verhalten wurde hier so eingeführt, daß sie mit der Hierarchie der Bauteile identisch ist. Das hat für den praktischen Gebrauch bei der Modellierung Vorteile, ist aber nicht unbedingt notwendig. Es kann in bestimmten Sonderfällen sinnvoll sein, in der Abstraktionshierarchie des Verhaltens noch weitere Zwischenknoten einzuziehen, durch die Verhaltensweisen zunächst zu kleinen Gruppen zusammengefaßt werden, bevor eine weitere Gruppierung im Vaterknoten erfolgt. Darauf soll hier nicht weiter eingegangen werden.

2.3.4 Abstraktion durch Zusammenfassen

Eine anders gelagerte Art von Abstraktion ist eine Abstraktion, die dadurch entsteht, daß man mehrere Bauteile zu einer größeren Einheit zusammenfaßt. Hier gibt es im allgemeinen wieder sehr viele Möglichkeiten, solche Zusammenfassungen vorzunehmen. Da wir uns bei der Diagnose aber auf die funktionalen Aspekte von Bauteilen beschränken, können wir hier - genauso wie bei den Abstraktionsaspekten - wieder eine Vereinfachung treffen: Es werden nur Zusammenfassungen betrachtet, die schon als Baugruppen definiert worden sind. Im Gegensatz zu willkürlichen Zusammenfassungen haben Baugruppen eine fest umrissene, sinnvolle Funktionsweise. Baugruppen kann man dann genau wie einzelne Bauteile in unsere Abstraktionshierarchie einfügen.

Die daraus resultierende Gleichstellung von atomaren Bauteilen und Baugruppen hat dann allerdings die Folge, daß Defekte eventuell nur auf eine Baugruppe zu begrenzen sind und nicht für ein einzelnes Unterbauteil angegeben werden können. Dies kann aber durchaus erwünscht sein, da in realen Diagnosesituationen in den seltensten Fällen einzelne Widerstände oder Relais, sondern gleich ganze Steckkarten und Module ausgetauscht werden.

2.3.5 Zusammenfassung

Ähnlichkeit kann man mit der vorgestellten Abstraktionshierarchie nicht nur für Bauteile an sich, sondern auch für ihr Verhalten angeben. Die Ähnlichkeit für Verhalten läßt sich auf zwei Weisen feststellen:

- *Durch Abstraktion der Verhaltensfunktionen*: Man kennt die Verhaltensfunktion, die korrekte Verhaltensweisen des Bauteils beschreibt. Ähnliches Verhalten von Söhnen eines Vaterknotens wird auf dasselbe Verhalten des Vaterknotens abgebildet.
- *Durch Abstraktion der Portwerte*: In der Regel kann man für Defektverhalten eines Bauteils a priori keine Verhaltensfunktion angeben. Man kann jedoch die Zustände der Bauteile vergleichen. Sie sind über die Werte an den Ein- und Ausgängen festgelegt und können ebenfalls abstrahiert werden.

Weiterhin können Bauteile auf verschiedenen Abstraktionsebenen beschrieben werden: Ein spezielles Kabel etwa allgemein als Kabel, allgemeiner als Übertrager oder ganz allgemein als Ding. Dadurch gewinnt man Flexibilität im Umgang mit der Datenbasis, da alle Objekte nur auf dem Detaillierungsgrad betrachtet werden, der gerade notwendig ist.

Nachdem wir nun den ersten Punkt der Grundidee, nämlich die Ähnlichkeitsbestimmung von Komponenten durch Verwendung von Modellwissen, geschildert haben, steht nun der zweite Punkt an: Die Unterstützung von Fallsuche und Fallübertragung durch Modellwissen.

2.4 Fallsuche und Fallübertragung

Wie wir sehen werden, macht es keinen Sinn, die beiden Vorgänge der Fallsuche und des Fallübertragens bzw. -anpassens in dem hier vorgestellten Ansatz zu trennen. Beide stützen sich auf das gleiche Wissen; in der Tat ist ja der gesuchte Fall derjenige, der sich besonders leicht an die gegebene Situation anpassen läßt (oder gar nicht erst angepaßt werden muß).

Der Diagnosevorgang läßt sich in zwei Schritte aufteilen:

1. Der Generierung von Hypothesen, die die beobachteten Symptome erklären können. Da der Defekt normalerweise durch die beobachteten Symptome nicht determiniert ist, erhält man hier eine Menge von Hypothesen, zwischen denen im nächsten Schritt differenziert werden muß.
2. Die Testauswahl. Es müssen geeignete Tests vorgenommen werden, mit denen man Hypothesen bestätigen oder widerlegen kann. Durch geschickte Auswahl der Tests will man im Diagnosevorgang möglichst schnell zu einer Lösung kommen.

Die Testauswahl ist nicht Gegenstand dieser Arbeit. Das dazu erforderliche Strategiewissen ist nicht vorhanden, man könnte auch hier Heuristiken in Form von Fällen benutzen oder aufgrund des Modellwissens Tests auswählen. MoCAS/2 läßt dem Benutzer die Wahl, welche Hypothese er verifizieren möchte, stellt dabei aber statistisches Wissen aus der Fallbasis zur Verfügung. Dabei handelt es sich um Ausfallraten eines speziellen Bauteils und gleichartiger Bauteile.

2.4.1 Die Vorgehensweise

Der Vorgang der Hypothesengenerierung gliedert sich in drei Schritte:

1. **Bestimmung von relevanten Symptomen**

   Eine Simulation des gewünschten Verhaltens wird mit den tatsächlich gelieferten Symptomen verglichen. Abweichungen vom Sollwert bezeichnen ungewolltes Verhalten und werden als relevant erkannt. Sie werden zur gezielten Suche in der Fallbasis benutzt, indem ähnliche Abweichungen bei ähnlichen Bauteilen aufgespürt werden. „Ähnlich“ ist dabei durch die Abstraktionshierarchie im vorigen Abschnitt bestimmt[1].
2. **Fallübertragung und Hypothesengenerierung**

   Dann wird versucht, die gefundenen Fälle auf die aktuelle Situation zu übertragen. Das geschieht dadurch, daß im Fall eine Wirkungskette von der Diagnose (dem defekten Bauteil) zu den beobachteten, pathologischen Symptomen aufgebaut wird. Diese Wirkungskette ist im Prinzip eine Folge von Verhaltensweisen von Bauteilen, derart, daß ein Verhalten eines Bauteils das darauffolgende Verhalten des nächsten Bauteils verursacht. Diese Wirkungskette wird auf die aktuelle Situation ähnlich übertragen. Auch hier bedeutet „ähnlich“ eine ähnliche Verhaltensweise im Sinne des letzten Abschnitts.

1. Wenn im Folgenden der Begriff „ähnlich“ gebraucht wird, ist damit die Ähnlichkeit gemeint, wie sie im vorigen Abschnitt durch die Abstraktionshierarchie eingeführt wurde. Diese Ähnlichkeit ist für das System berechenbar.

3. Überprüfen der Hypothesen

Gelingt diese Übertragung, erhält man als Hypothese ein der Diagnose entsprechendes Bauteil mit ähnlichen Port-Werten. Diese Hypothese wird durch eine Simulation verifiziert.

Diese Art der Hypothesengenerierung läßt sich auf verschiedenen Abstraktionsebenen durchführen. Findet man also auf einer niedrigen (konkreten) Abstraktionsebene keine Lösung, kann der Vorgang auf einer höheren Ebene wiederholt werden. Das setzt sich so lange fort, bis man beim Wurzelknoten der Abstraktionshierarchie angekommen ist. Spätestens dann läßt sich jeder Fall übertragen.

Wenn wir die Vorgehensweise jetzt genauer betrachten, müssen die drei vorgestellten Schritte allerdings noch ergänzt werden:

1. Ableiten von resultierenden Werten
2. Prüfen auf sofort feststellbare Defekte
3. Bestimmung von relevanten Symptomen und Retrieval
4. Fallübertragung und Hypothesengenerierung
5. Hypothesenüberprüfung

2.4.2 1. Schritt: Ableiten von resultierenden Werten in der Situationsbeschreibung

In unserer Modellbeschreibung sind für Bauteile Verhaltensregeln oder -funktionen angegeben, die ihre Funktion beschreiben. Weiterhin sind für Baugruppen die Verbindungen zwischen ihren Bauteilen gegeben. Diese Verbindungen sehen wir als „sicher" an, d. h. sie können nicht defekt sein. Wollten wir einen Defekt einer solchen Verbindung diagnostizieren, könnten wir bei der Modellierung an dieser Stelle beispielsweise ein Bauteil „Lötstelle" einführen, das wiederum ein Verhalten hat und damit auch defekt sein könnte.

Verbindungen zwischen Bauteilen einer Baugruppe sind also keine physikalischen Verbindungen im Sinne einer Lötstelle oder Klemme, sondern Verbindungen in der Modellierung. Werte werden entlang dieser Verbindungen propagiert und dadurch wird die Situationsbeschreibung um sichere Werte erweitert.

2.4.3 Schritt 2: Prüfen auf sofort feststellbare Defekte

In unserer Modellbeschreibung sind für jedes Bauteil Verhaltensregeln gegeben. Wenn die bisher ermittelten Werte diesen Verhaltensregeln wiedersprechen, haben wir den Fehler sofort gefunden, wir können also an dieser Stelle sofort eine sichere Diagnose stellen.

Beispiel **Kurzschluß**

Wenn zu einem Kabel bekannt ist, daß an einem Ende Strom hineingeht, am anderen Ende aber kein Strom herausführt, können wir einen Verstoß gegen die Verhaltensfunktionen f_{ende1}(ende2) = ende2 und f_{ende2}(ende1) = ende1 feststellen. Es gibt keine Verhaltensfunktion, die auf die gegebenen Werte paßt, die gegebenen Werte widersprechen den Verhaltensfunktionen. Damit ist ein Defekt im Kabel gefunden; ein weiteres fallbasiertes Vorgehen wird überflüssig.

2.4.4 Schritt 3: Bestimmung von relevanten Symptomen und Retrieval

Relevant sind Symptome, die auf einen Fehler hinweisen, d. h. sie beschreiben ein Verhalten, daß wir nicht gewollt haben. Ein solches Verhalten wird im Folgenden als *unintendiert* bezeichnet. Unintendierte Symptome kann man relativ einfach ermitteln: Man führt eine Simulation der in der Problemstellugn gegebenen Situation durch, um herauszubekommen, wie die Maschine sich eigentlich hätte verhalten sollen. Danach vergleicht man die beobachteten Symptome in der Situationsbeschreibung mit den entsprechenden Werten der Simulation.

Charakteristisch für einen Maschinendefekt sind Teile, die ein unintendiertes Verhalten zeigen. Solches Verhalten bedeutet dabei nicht, daß das Bauteil defekt ist. Es bedeutet lediglich, daß das beobachtete Verhalten nicht dem Verhalten entspricht, daß man eigentlich erwartet.

Beispiel **Auto**

Die Autobatterie ist leer. Aus dem Kabel, das zum Scheinwerfer führt, kommt kein Strom. Die Birne im Scheinwerfer brennt nicht. Die Birne zeigt also ein unintendiertes Verhalten, ist aber selbst nicht defekt.

Intendiertes Verhalten von Bauteilen ist zwar auch relevant, um später Hypothesen auszuschließen, es liefert aber keinen Hinweis auf einen Defekt.

Weiterhin sind Folgefehler, also unintendiertes Verhalten, das aus anderem, unintendierten Verhalten resultiert, nicht so charakteristisch für einen Defekt wie das verursachende Fehlverhalten. In unserem Beispiel ist die Tatsache, daß aus dem Kabel zum Scheinwerfer kein Strom kommt, interessanter als die Tatsache, daß die Birne nicht leuchtet. Wir sind durch das Kabel der Ursache näher als durch die Birne.

Von diesem Gedankengang abgeleitet erfolgt die Indexierung:

Im Fall: Zu jedem Fall wird festgestellt, welche Symptome auf unintendiertes Verhalten der Bauteile beschreiben und *alle* diese Symptome werden entsprechend als *relevant* gekennzeichnet.

In der Situationsbeschreibung (Anfrage): Erfolgt nun eine Anfrage an das System, werden zunächst alle die Symptome, die auf unintendiertes Verhalten hinweisen, herausgesucht. Unter diesen wiederum werden die ermittelt, die am ursächlichsten sind, also keine Folge eines anderen Symptoms sind. Nur diese werden auch als *relevant* gekennzeichnet.

Die Suche in der Fallbasis erfolgt dann so, daß Fälle herausgesucht werden, die *relevante* Symptome haben, die zu den *relevanten* Symptomen der Situation ähnlich sind. Mit diesen Fällen wird dann weitergearbeitet. Die Markierung von Symptomen als *relevant* kann man dabei als eine Art Indexing betrachten. Dieser Index ermöglicht es, größere Fallbasen effizienter zu verwalten und erspart das lineare Durchsuchen der Fallbasis.

Diskussion dieser Indexierung

Warum diese Art der Indexierung gewählt wurde, wird deutlich, wenn man sie alternativen Indexierungsmöglichkeiten gegenüberstellt.

In der Fallbasis werden alle Bauteile mit nicht-intendiertem Verhalten als relevant markiert. Es werden also auch Folgeverhalten markiert und nicht nur die verursachenden Symptome. Warum?
Die Vorgehensweise in der Diagnostik ist normalerweise die, daß man mit sehr allgemeinen Fehlerbeschreibungen beginnt (z. B. „Automotor läuft nicht"), um sich dann immer weiter an den Defekt heranzutasten. Das wird auch bei der Benutzung unseres Diagnosesystems so sein. Man hat zwar als letztes Fehlverhalten, bevor der eigentliche Defekt lokalisiert ist, ein sehr spezielles Fehlverhalten (z. B. „Zündkerzen zünden nicht" für den Defekt „Verteilerkopf naß"). Dieses ist auch ein sehr guter Hinweis auf den Defekt, wenn es aber nur allein als Index benutzt würde, fehlt der „Einstieg" für den Diagnoseprozeß. (Im Beispiel wird man als erstes Symptom angeben, daß der Motor nicht läuft und erst später zu den Zündkerzen kommen. Würde das Symptom „Motor läuft nicht" nicht als relevant gekennzeichnet, dann würde der Fall nicht gefunden)

Warum wird dann in der Situationsbeschreibung der Problemstellung nicht genauso verfahren, also alle resultierenden Fehlverhalten bestimmt und als Index benutzt?
Zugrunde liegt die Heuristik, daß bei gleichen Fehlern auch wieder die gleichen Symptome genannt werden. Das bedeutet, daß keine zusätzlichen Anhaltspunkte („Einstiege") für einen bestimmten Fall nötig sind, da ein erneuter „Einstieg" in den Fall wieder über den gleichen Weg erfolgt. Werden tatsächlich verschiedene Einstiege verwendet, die dann zur gleichen Diagnose führen, kann man entweder den neuen Fall mitsamt „Einstieg" in die Fallbasis aufnehmen oder besser den alten Fall in der Fallbasis um diesen „Einstieg" erweitern.

Bei einer Anfrage hingegen ist es sinnvoll, sich zunächst nur auf die besonders charakteristischen Symptome (ohne Folgefehler) zu beschränken. Man hat sich durch ihr Auffinden ja besonders nahe an den Defekt herangetastet, deshalb sollte man sich zunächst beim Retrieval nur auf diese Teile beschränken. Läßt sich über diese Symptome kein Fall finden, der zur Erzeugung einer Hypothese geeignet ist, kann man dann auf die weniger charakteristischen Symptome ausweichen.

Trotzdem: Diese hier beschriebenen Vorauswahl bzw. Indexierung der Fälle ist eine Heuristik. Eigentlich müßte das Retrieval dadurch erfolgen, daß versucht wird, jeden einzelnen Fall der Fallbasis auf die konkrete Anfrage zu übertragen. Gelingt diese Übertragung, so ist der Fall ähnlich, das Resultat der Übertragung ist eine Hypothese. Die Fallübertragung und -anpassung ist jedoch mit Aufwand verbunden, daher lohnt es sich, die Fälle vorzufiltern.

Es ist nicht gesagt, daß einerseits alle Fälle, die die Vorauswahl passieren, tatsächlich auch ähnlich sind, andererseits können durch die Vorauswahl auch Fälle zurückgehalten werden, aus denen sich eine korrekte Diagnose ableiten ließe. Eine solche Fehlauswahl tritt aber nur auf, wenn keines der im Fall gegebenen pathologischen Symptome in der Situation auftritt, d. h. wenn es in der Situation also nicht den geringsten Hinweis (in Form eines auffälligen Symptoms) auf diesen Fall gibt. Daher ist diese Einschränkung auch keine wesentliche Beeinträchtigung für den Diagnosevorgang.

2.4.5 Schritt 4: Die Fallübertragung

Als Ergebnis der Vorauswahl erhalten wir zu einer Anfrage eine Menge von Fällen, die möglicherweise zur gegebenen Situation ähnlich und zur Lösungsfindung nützlich sind. Daher versuchen wir, diese Fälle auf die Situation zu übertragen. Wir wollen jetzt aber nicht den gesamten Fall auf unsere Situation übertragen, sondern nur den Teil, der für uns nützlich ist.

Beispiel **Autopanne**

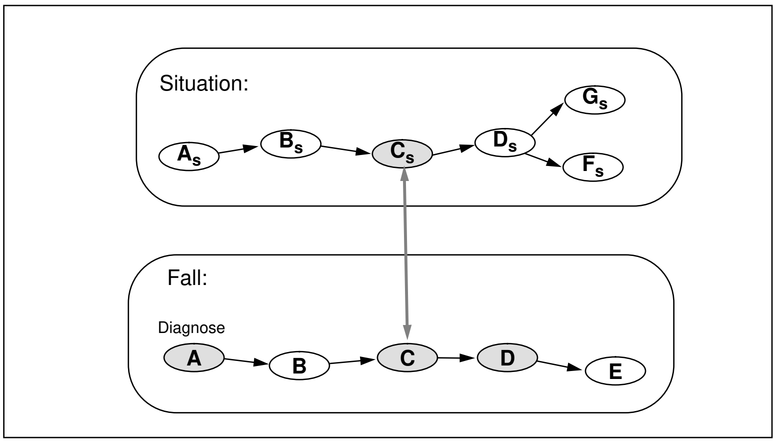
In der Fallbasis haben wir einen Fall, der beschreibt, wie bei einer Autopanne das Rad hinten links platt ist. In der Situation, in der wir uns befinden, ist das Rad vorne rechts platt. Übertragen wollen wir also nur den Teil des Falls, der sich auf das Rad bezieht, der größere Kontext (die Position des Rads am Wagen) soll nicht übertragen werden

Der Teil des Falls, den wir übertragen wollen - der Kontext - läßt sich durch Fehlersimulation bestimmen. Dabei verfolgen wir die Auswirkungen des defekten Bauteils im Fall (der Diagnose) auf alle anderen Bauteile. Wir erhalten eine Kette von Auswirkungen. In unserem Eingangsbeispiel war eine solche Kette gegeben:

Nagel im Reifen ⇒ Reifen vorne rechts platt ⇒ Auto zieht nach rechts

Eine solche Kette ist abstrakt in Abbildung 17 dargestellt. Das defekte Bauteil A verursacht ein Fehlverhalten von Bauteil B, dieses wiederum ein Fehlverhalten von Bauteil C usw. Die für unseren Indexing-Schritt benutzten relevanten Symptome sind schraffiert dargestellt. In unserer Situation waren wir auf das Bauteil C_s gestoßen, das unintendiertes Verhalten gezeigt hatte. Wir haben es durch das Indexing mit dem Bauteil C des Falls identifiziert (grauer Pfeil).

ABBILDUNG 17 In einer Wirkungskette wird der Kontext der Übertragung bestimmt



Jetzt ist auch ersichtlich, wie der Fall auf die Situation übertragen wird: Man verfolgt im Fall die Wirkungskette von C nach A zurück. Parallel dazu wird in der Situation eine gleichartige Kette rückwärts von C_s ausgehend aufgebaut. Dabei verlangt man bei jedem Schritt zurück, daß die Bauteile ähnlich sind und ähnliches Verhalten zeigen, sofern das aus der Situationsbeschreibung bekannt ist; außerdem sollen sie noch ähnliche Wirkungen aufeinander haben. Alles das ist aus der Ähnlichkeitsfestlegung durch die Abstraktionshierarchie bestimmbar. Während man diesen Pfad aufbaut, werden also immer zwei Bauteile miteinander identifiziert, eines aus dem Fall mit einem aus der Situation. Wenn es gelingt, diese Kette aufzubauen, ist die Übertragung des Falls gelungen; das Bauteil, das mit der Diagnose identifiziert ist, ist die generierte Hypothese.

Wenn man so vorgeht, kann es passieren, daß aufgrund des Indexing mehrere Übertragungen möglich sind. So könnte in Abbildung 17 das Bauteil C_s nicht nur zu C, sondern auch zu E ähnlich sein. Dann erhält man zwei Übertragungsmöglichkeiten, aus denen unter Umständen zwei Hypothesen generieren werden. Das kann durchaus vernünftig sein, eine solche Situation tritt dann auf, wenn man Bauteile betrachtet, die mehrmals hintereinander in der Maschine vorkommen, wie etwa Kabel.

Den Schritt der Fallübertragung führen wir mit jedem Fall durch, der nach unserer Vorauswahl übrig geblieben ist. Wir erhalten jetzt also eine Menge von Hypothesen, die wir im nächsten Schritt durch unser Modellwissen überprüfen werden.

2.4.6 Schritt 5: Überprüfen der generierten Hypothesen

Die Idee dazu ist, eine Simulation mit der gefundenen Hypothese durchzuführen und zu überprüfen, ob sich daraus Widersprüche zur Situationsbeschreibung ergeben. Um eine Hypothese zu widerlegen, hat man zwei Möglichkeiten:

- Es wurde unintendiertes Verhalten an Bauteilen festgestellt, dieses Verhalten kann durch die Hypothese aber gar nicht hervorgerufen werden.
- Die Hypothese würde unintendiertes Verhalten von Bauteilen hervorrufen; es wurde aber beobachtet, daß diese Bauteile sich korrekt verhielten.

Um die Hypothese zu testen, muß man eine Fehlersimulation durchführen. Sie ist, wie schon bemerkt, nicht so einfach wie eine Simulation des korrekten Verhaltens. Hier befinden wir uns aber in der Lage, daß das Fehlverhalten des defekten Bauteils genau bekannt ist: es ist in der Diagnose beschrieben; es ist bereits in ähnlicher Weise auf die Hypothese übertragen. Alle anderen, nicht-defekten Bauteile verhalten sich nach wie vor korrekt. Daher kann man bei der Fehlersimulation so vorgehen: Man simuliert alle Bauteile wie in einer normalen Simulation auch, nur bei dem Bauteil, das durch die Hypothese beschrieben ist, verfährt man anders. Dort werden die Portwerte direkt aus der Hypothese übernommen, statt das Verhalten dieses Teils zu simulieren.

2.4.7 Die Kontrollstruktur

Die Schritte 3 bis 5 finden immer auf einer ganz bestimmten Ähnlichkeitsstufe statt, auf der geprüft wird, ob Bauteile, Verhalten, etc. ähnlich waren oder nicht. Diese Ähnlichkeitsstufe ist frei wählbar. Sinnvollerweise beginnt man dabei auf einer möglichst konkreten Stufe, da bei jeder Abstraktion Informationen verloren gehen. Ist es allerdings nicht möglich, eine Hypothese zu generieren, muß stärker abstrahiert werden. Spätestens auf der höchsten Stufe „Ding" ist jeder Fall übertragbar: Dann verursacht ein Ding, das irgendein Verhalten hat, irgendein Verhalten eines anderen Dinges. Wenn man auf dieser Ebene eine Übertragung vornimmt, gelangt man also von dem Bauteil, an dem unintendiertes Verhalten festgestellt wurde, zu all jenen Bauteilen, die es unmittelbar beeinflussen. Diese Strategie entspricht dann einer Rückverfolgung von Fehlverhalten, wie sie bei modellbasierten Verfahren stattfindet.

2.4.8 Zusammenfassung

Wir sind mit diesem Vorgehen in der Lage, die Ähnlichkeit von Fällen und Situationen auf einer höheren, semantischen Ebene zu beurteilen. Die Relevanz von einzelnen Symptomen kann ermittelt werden, Fälle und Situationen werden kontextgerecht einander angepaßt. Dies alles kann auf verschiedenen Abstraktionsebenen geschehen. Die generierten Hypothesen sind mit dem Modellwissen konsistent und durch die Anwendung von Erfahrungswissen (Fällen) entstanden.

2.5 Erklärung für Hypothesen

Um eine generierte Hypothese in einem fallbasierten System zu erklären, ist der einfachste Weg, den Fall anzugeben, durch den diese Hypothese generiert worden ist. Das vereinfacht die Erklärung aber nicht unbedingt, da man dann begründen muß, warum der Fall und die Situation ähnlich sind. In MoCAS/2 kann eine Hypothese auf zwei Weisen begründet werden:

1. **Durch das Ergebnis der Fehlersimulation**: Bei der Überprüfung der Hypothese wurden genau die Punkte geprüft, die man auch jetzt angeben muß: Auf

welche Art und Weise kann ein Defekt, so wie er durch die Hypothese beschrieben ist, die in der Situation beobachteten Symptome hervorrufen? Hier liegt es nahe, Wirkungsketten von der Hypothese zu den pathologischen Symptomen anzugeben. Diese Wirkungsketten kann man dann der Übersichtlichkeit halber noch verkürzen, indem man etwa Kabel, Lötstellen, Leiterbahnen und ähnliche Bauteile wegläßt.

2. **Durch den Fall**: Um nun zu erklären, warum gerade dieses bestimmte Bauteil in der Hypothese gewählt wurde und nicht irgendeine der anderen möglichen Ursachen, kann man den Fall angeben: Ein ähnlicher Defekt ist bereits aufgetreten.

   Die Ähnlichkeit des Falls läßt sich dann durch die übertragene Wirkungskette begründen: Die Teile entlang dieser Kette sind einander ähnlich, sie haben ähnliche Wirkungen aufeinander, ihr Verhalten ist ähnlich.

KAPITEL 3 Formale Beschreibung

In diesem Kapitel soll der Ansatz, der im vorigen Kapitel informell und erläuternd vorgestellt worden ist, formal in algorithmischer Form dargestellt werden. Dieses Kapitel ist sicher nicht geeignet, um die Vorgehensweise von MoCAS/2 zu erläutern. Es soll die Überlegungen des letzten Kapitels präzisieren und aufzeigen, wie eine Implementierung realisierbar sein könnte.

Zunächst wird eine Terminologie für die grundlegenden Begriffe eingeführt, um damit anschließend die Vorgehensweise zur Diagnose in technischen Domänen zu beschreiben.

3.1 Terminologie

3.1.1 Komponenten, Ports und States

Idee: Wir folgen auch hier der im letzten Kapitel eingeführten Framedarstellung. Ein Frame stellt ein in der Maschine vorkommendes Bauteil dar. Gleichartige Bauteile werden zu Bauteilklassen zusammengefaßt. Dadurch kann man Eigenschaften, die für alle Bauteile einer Klasse gleich sind, einfacher beschreiben. Später werden die Bauteilklassen benutzt, um eine Abstraktionshierarchie aufzubauen, so wie sie in ,,Abstraktion von Bauteilen“ auf Seite 31 eingeführt wurde.

Repräsentation: Ein Bauteil B wird als Frame dargestellt, seine Ports und States entsprechen Slots. Jeder Slot bzw. Port hat einen ihm zugeordneten Wertebereich. Diese Bauteile sind dann Instanzen der zugehörigen Bauteilklasse BK.

3.1.2 Symptome

Idee: Ein Symptom beschreibt einen Wert, der an einem Port oder einem State einer Bauteilinstanz anliegt. Dieses Symptom ist ein einzelner Meßwert an einem bestimmten Bauteil der Maschine. Es bietet sich natürlich an, alle Symptome, die an demselben Bauteil auftreten, gemeinsam zu betrachten. Diese Menge von Symptomen beschreibt dann den Zustand des Bauteils.

Repräsentation: Ein Symptom $S = (B, P, W)$ ist ein Tripel aus einem Bauteil B, einem Port/State P und einem Wert W.

3.1.3 Regeln und Verhalten

Idee: Das Verhalten eines Bauteils wird in Regeln dargestellt. Da das Verhalten für alle Bauteile einer Klasse gleich ist, bietet es sich an, die Regeln auf Bauteilklassen zu definieren. Eine Regel besteht aus einem Bedingungsteil und einem Aktionsteil. Der Bedingungsteil beschreibt einen Zustand eines Bauteils, der Aktionsteil enthält Wertzuweisungen, die ausgeführt werden, wenn sich das Bauteil in diesem Zustand befindet.

Repräsentation: Eine Regel hat die Form $BK\colon Vorbed \to Akt$. $Vorbed$ ist eine Konjunktion von Vorbedingungen, Akt eine Konjunktion von Aktionen. Eine einzelne Bedingung hat die Form: $P = x$, wobei x entweder ein Wert W oder eine Variable ist. BK bezeichnet eine Bauteilklasse, P einen Port/State dieser Klasse.[1]

Das Erfüllen von Bedingungen ist im üblichen Sinne definiert, Variablen in Bedingungen können gebunden werden und in anderen Bedingungen wie Werte benutzt werden. Die Aktionen haben die Form $P \leftarrow x$, dabei ist x wieder Wert oder Variable. Wird eine Aktion ausgeführt, wird dem in P spezifizierten Port/State ein Wert zugewiesen.

Eine Regel feuert, wenn alle ihre Vorbedingungen erfüllt sind; dabei werden alle Aktionen ausgeführt.

3.1.4 Verbindungen

Idee: Baugruppen bestehen aus mehreren Einzelbauteilen, die untereinander verbunden sind. Diese Verbindungen werden als sicher angesehen, da sie nicht physikalisch, sondern nur im Sinne der Modellierung bestehen. (Diese Unterscheidung wurde in: ,,1. Schritt: Ableiten von resultierenden Werten in der Situationsbeschreibung“ auf Seite 38 bereits ausführlich betrachtet). Eine solche Verbindung von A nach B wird durch zwei Regeln angegeben: $A \to B$ und $B \to A$.

Repräsentation: Um auszudrücken, daß in einer Baugruppe BG der Port P_1 von Bauteil B_1 mit dem Port P_2 von Bauteil B_2 verbunden ist, führt man zwei Regeln ein:

$BG\colon ((B_1P_1 = x) \to (B_2P_2 \leftarrow x))$ und $BG\colon ((B_2P_2 = x) \to (B_1P_1 \leftarrow x))$.

1. Man kann natürlich auch Regeln für Baugruppen angeben. Dabei können sich die P auf verschiedene Unterbauteile dieser Baugruppe beziehen. Allerdings muß dann bei der Angabe von P nicht nur der Port/State, sondern auch der Name des Unterbauteils angegeben werden, also statt „Emitter“ muß „Transistor3.Emitter“ geschrieben werden.

3.1.5 Situationsbeschreibung, Zustandsbeschreibung einer Komponente

Idee: Eine Situationsbeschreibung erklärt, unter welchen Umständen ein Störfall in der Maschine aufgetreten ist. Diese Beschreibung informiert zum einen darüber, was eigentlich geschehen sollte, und zum anderen, was tatsächlich beobachtet worden ist. Die Beschreibung dessen, was eigentlich passieren sollte (intendiertes Verhalten) ist durch eine Zustandsbeschreibung derjenigen Bauteile möglich, mit denen der Benutzer die Maschine bedient. Eine solche Beschreibung besteht aus Symptomen. In der gleichen Form läßt sich dann auch das tatsächlich beobachtete Verhalten angeben. Wie bei der Einführung von Symptomen schon erwähnt, werden Symptome, die am gleichen Bauteil auftreten, zusammen betrachtet.

Repräsentation: Eine Situationsbeschreibung Sit besteht aus zwei Mengen von Symptomen: Int und Obs. Int enthält Zustandsbeschreibungen, durch die das intendierte Verhalten der Maschine ausgedrückt wird. Das beobachtete Verhalten der Maschine ist in der gleichen Form in Obs angegeben. Die Zustandsbeschreibung eines Bauteils in Sit ist die Teilmenge derjenigen Symptome, die an diesem Bauteil auftreten.

3.1.6 Diagnose

Idee: Eine Diagnose beschreibt ein Bauteil, das ein Defektverhalten gezeigt hat. Dieses Verhalten ist wieder durch eine Zustandsbeschreibung für das Bauteil darstellbar[2].

Repräsentation: Eine Diagnose D ist eine Situationsbeschreibung, die nur die Ports/States des defekten Bauteils beschreibt.

3.1.7 Fall

Idee: Ein Fall gibt an, welche Diagnose in einer bekannten Situation gestellt worden ist.

Repräsentation: Ein Fall $F = (Sit, D)$ ist ein Paar aus einer Situationsbeschreibung Sit und einer Diagnose D.

3.1.8 Fallbasis

Eine Fallbasis FB ist eine Menge von Fällen F.

3.1.9 Abstraktionsgraph

Idee: In einem Abstraktionsgraph ist die Abstraktionshierarchie beschrieben, die zwischen den betrachteten Bauteilklassen besteht. Weiterhin finden sich dort Abstraktionsabbildungen, mit denen man Slots, Werte und Verhaltensregeln auf

2. Natürlich kann eine Diagnose auch mehrere defekte Bauteile beschreiben. Darauf wird später im Algorithmus aber nicht näher eingegangen, um diesen nicht unnötig zu komplizieren. Eine entsprechende Erweiterung ist einfach vorstellbar. Damit ist aber nicht gemeint, daß eine neue Diagnose gefunden werden könnte, die aus den Einzeldiagnosen verschiedener Fälle zusammengesetzt ist, sondern daß die Diagnose eines Falls mehrere defekte Bauteile beschreibt und diese die Lösung für eine Anfrage bilden

höhere Abstraktionsebenen transformieren kann. Ausführlich ist diese Hierarchie in Kapitel 2 unter 2.3 ,,Ähnlichkeit von Bauteilen und deren Verhalten“ beschrieben.

Repräsentation: Ein Abstraktionsgraph $A = (N, E)$ ist ein Baum, dessen Knoten N mit Bauteilklassen BK beschriftet sind. Eine Kante E von N_1 nach N_2 ist mit zwei Mengen von (Abstraktions-) Funktionen beschriftet. Die erste Menge enthält Funktionen, die die Werte der Ports von N_1 auf die entsprechenden Werte von N_2 abbilden. Die zweite Menge enthält Funktionen, die einer Verhaltensregel der Bauteilklasse in N_1 eine entsprechende Verhaltensregel in N_2 zuordnen.

Mit Hilfe dieser Funktionen kann man eine Bauteilinstanz (einen Frame) einer Klasse BK_1 in eine abstraktere Darstellung der Klasse BK_2 überführen, falls im Abstraktionsgraph eine Kante von BK_1 nach BK_2 existiert.

3.2 Der Algorithmus

Damit der Algorithmus möglichst verständlich bleibt, wurde an vielen Stellen eine natürlichsprachlich-informelle Beschreibung des Vorgehens gewählt. Zusätzlich sind Kommentare in Klammern mit Sternchen eingefügt, um den angegebenen Schritt zu erläutern. Befehlswörter wurden unterstrichen. Zunächst wird der Algorithmus auf Top-level vorgestellt, die aufgerufenen Unteralgorithmen folgen danach. Im Basisalgorithmus verweisen die Kommentare auf die in Kapitel 2, 2.4 ,,Fallsuche und Fallübertragung“ eingeführten Schritte.

3.2.1 Der Basisalgorithmus

Gegeben sind:

Eine Fallbasis $FB = \{F_1, F_2, \ldots, F_n\}$
Eine Situationsbeschreibung Sit
Ein Abstraktionsgraph A
Zu jeder Bauteilklasse BK_i in A eine Menge von Verhaltensregeln R_{BK_i}
Eine Menge $MV = \{V_1, V_2, \ldots, V_m\}$ von Verbindungsregeln.

*(*Schritt 1: Ableiten sicherer Werte*)*

<u>`Setze`</u> Sit_{erw} := `Leite_Werte_ab`(*Sit*).

*(*Schritt 2: Prüfen auf sofort bestimmbare Defekte*)*

<u>`Setze`</u> $Defekt$:= `Finde_bestimmbare_Defekte`.

<u>`Falls`</u> $Defekt \neq \emptyset$

<u>`Dann`</u> STOP mit Ausgabe $Defekt$.

*(*Schritt 3a: Berechnen relevanter Symptome*)*

*(*Berechnen rel. Symptom in der Situationsbeschreibung*)*

Setze Rel_{Sit} := `Berechne_relevante_Symptome_für_Sit`.

*(*Berechnen von rel. Symptomen in den Fallbeschreibungen*)*

Setze $Rel_{FB} := \emptyset$.

Wiederhole für jedes $F_i \in FB$:

  Setze Rel_{F_i} := `Berechne_relevante_Symptome_für_Fall` (F_i).

  Setze $Rel_{FB} := Rel_{FB} \cup \{Rel_{F_i}\}$.

Ende.

Setze $Abstraktionsebene := 0$.

Setze $Hypothesen := \emptyset$.

Wiederhole bis $Hypothesen \neq \emptyset$

  Wiederhole bis $Kandidaten \neq \emptyset$

    Setze $Abstraktionsebene := Abstraktionsebene + 1$.

*(*Schritt 3b: Retrieval*)*

    Setze $Kandidaten$:= `Finde_möglich_passende_Fälle_auf_Abstraktionsebene`($Abstraktionsebene$).

  Ende

*(*Schritt4: Hypothesengenerierung*)*

  Setze $Hypothesen$:= `Generiere_Hypothesen_aus`($Kandidaten$, $Abstraktionsebene$).

*(*Schritt5: Hypothesenüberprüfung*)*

  Setze $Hypothesen$:= `Prüfe_Hypothesen`($Hypothesen$, $Abstraktionsebene$).

Ende.

^$Hypothesen$

3.2.2 Die aufgerufenen Unteralgorithmen

ALGORITHMUS:Leite_Werte_ab(*Sit*)

*(*Alle Symptome, die in Sit nicht gegeben sind, werden als Symptome mit dem Wert unbekannt in Sit neu aufgenommen. Sit enthält danach alle denkbaren Symptome, entweder mit echten Werten, oder mit dem Wert unbekannt*)*

Wiederhole für alle Bauteilinstanzen B_i:

  Wiederhole für alle Ports P_{ij} von B_i:

    Falls kein Symptom $S = (B_s, P_s, W_s)$ in Sit enthalten ist, Dann Setze $Sit := Sit \cup \{ (B_i, P_i, unbekannt) \}$

Ende

*(*Alle in der Situation gegebenen Werte werden entlang der Verbindungen, die in MV spezifiziert sind, propagiert*)*

Wiederhole für alle Symptome $S_i = (B_i, P_i, x_i)$ mit $x_i \neq unbekannt$ aus Sit:

*(*Suche eine passende Verbindungsregel aus MV und propagiere den Werte, falls möglich*)*

Feuere alle Regeln aus MV, bis keine neuen Werte mehr zugewiesen werden.

Ende

^Sit

ALGORITHMUS:Finde_bestimmbare_Defekte

*(*Betrachte zu jedem Bauteil alle seine Verhaltensregeln. Falls die Symptome, die zu dem Bauteil angegeben sind, einer Regel widersprechen, ist ein Defekt gefunden. Ein Widerspruch entsteht dadurch, daß in den Symptomen andere Werte gegeben sind als die Werte, die durch die Regeln hergeleitet würden. Unbekannte Werte werden bei dieser Prüfung ignoriert*)*

Setze $Backup := Sit_{erw}$

*(*Prüfe Fehlverhalten für jedes Bauteil*)*

Wiederhole für alle Bauteilinstanzen B_i:

*(*Prüfe für jede Verhaltensregel des Bauteils, ob ein Verstoß vorliegt*)*

Wiederhole für jede Regel aus der Bauteilklasse zu B_i:

Feuere Regel, falls möglich; dadurch verändert sich Sit_{erw}

Vergleiche $Backup$ mit Sit_{erw}.

*(*Prüfe, ob gegen die Regel verstoßen wurde. Das ist der Fall, wenn durch die Regel ein anderer Wert erzeugt wurde, als in der Situation angegeben worden ist. Wenn ein bisher unbekannter Wert auf einen echten Wert umgesetzt worden ist, ist das kein Verstoß*)*

Falls sich ein Symptom in $Backup$ vom entsprechenden Symptom in Sit_{erw} unterscheidet, wobei das Symptom in $Backup$ nicht den Wert *unbekannt* hat

Dann ^Verhaltensbeschreibung von B_i in Form von allen Symptomen, die für B_i gegeben sind.

Ende

Ende

Setze $Sit_{erw} := Backup$.

^∅

ALGORITHMUS:Berechne_Relevante_Symptome_für_Sit

*(*Simuliere aus Int das intendierte Verhalten der Maschine. Vergleiche dieses mit den tatsächlich beobachteten Symptomen. Unterschiede zeigen unintendiertes Verhalten. Finde von allen unintendierten Verhalten die ursächlichsten, d. h. sortiere unintendiertes Folgeverhalten aus*)*

*(*Simuliere intendiertes Verhalten*)*

<u>Setze</u> $Sit_{intendiert}$:= Simuliere_Verhalten_aus (Int).

*(*Diff ist die Menge aller Symptome, die in der Situation unintendiert sind, also keine intendierten Symptome sind*)*

<u>Setze</u> $Diff$:= Menge aller Symptome $S = (B, P, W)$ mit:
$(B, P, x) \in Obs$ und
$(B, P, y) \in Sit_{intendiert}$ und
$(x \neq y)$ und
$(x \neq unbekannt)$

*(*Finde unintendiertes Folgeverhalten. Übernimm dazu ein Symptom aus $Diff$ in Int, d. h. simuliere die Maschine mit einem zusätzlichen unintendierten Verhalten. Treten im Ergebnis dieser Simulation dann andere Symptome aus $Diff$ auf, sind das Folgen des gewählten unintendierten Symptoms*)*

<u>Wiederhole</u>:

<u>Setze</u> S_{test} := Menge aller Symptome aus $Diff$, die an einem Bauteil auftreten; also eine Zustandsbeschreibung dieses Bauteils in $Diff$

*(*Für S_{test} sollen keine neuen Werte simuliert werden, sondern die gegebenen Symptomwerte übernommen werden*)*

Verhindere temporär eine Simulation des Bauteils von S_{test}.

<u>Setze</u> Sit_{test} := Simuliere_Verhalten_aus ($Int \cup \{S_{test}\}$).

Streiche alle Symptome aus $Diff$, die in $Sit_{test} \backslash S_{test}$ enthalten sind.

<u>Bis</u> alle Bauteile in $Diff$ einmal ausgewählt worden sind.

^$Diff$

ALGORITHMUS: Berechne_relevante_Symptome_für_Fall (*Fallbeschreibung*)

*(*Vorgehen wie in ALGORITHMUS: ,,Berechne_Relevante_Symptome_für_Sit“. Das Ausdünnen der unintendierten Folgesymptome wird nicht durchgeführt*)*

ALGORITHMUS:Simuliere_Verhalten_Aus (*Symptommenge*)

*(*Bei einer Simulation werden ausgehend von den gegebenen Symptomen alle Symptome abgeleitet, die sich über Verhaltensregeln oder Modellverbindungen ableiten lassen*)*

Setze $Simuliert := Symptommenge$.

Feuere alle anwendbaren Regeln (Verhaltensregeln in den R_{BK_i} und Modellverbindungen in MV) bis keine Regel mehr anwendbar ist. Erweitere dabei $Simuliert$ durch die neu abgeleiteten Werte.

^$Simuliert$

ALGORITHMUS:Finde_möglich_passende_Fälle_auf_Abstraktionsebene (AE)

*(*Alle Betrachtungen finden auf der Abstraktionsebene AE statt. Suche Fälle, für die dann gilt: Sie besitzen relevante Symptome, die zu den relevanten Symptomen der aktuellen Situation gleich sind. Die Tripel (S_{Sit}, S_{Fall}, F) aus einem Symptom der Situation, dem entsprechenden Symptom eines Falls und dem Fall selbst werden gesammelt und sind der Rückgabewert des Algorithmus*)*

Setze $Gefunden := \varnothing$.

Wiederhole für jedes Symptom S_{Sit} aus Rel_{Sit}

  Setze $S_{Sit(abstrakt)}$:= Transferiere_auf_Abstraktionsebene(AE, S_{Sit}).

  Wiederhole für jeden Fall F_i aus FB:

    Wiederhole für jedes Symptom S_{Fall} aus Rel_{F_i}

      Setze $S_{Fall(abstrakt)}$:= Transferiere_auf_Abstraktionsebene(AE, S_{Fall})

      Falls $S_{Fall(abstrakt)} = S_{Sit(abstrakt)}$

        Dann Setze $Gefunden := Gefunden \cup (S_{Sit}, S_{Fall}, F_i)$

    Ende

  Ende

Ende

^$Gefunden$.

ALGORITHMUS:Transferiere_auf_Abstraktionsebene (AE, $Symptom$)

*(*Im Abstraktionsgraphen sind für Ports und deren Werte Abstraktionsabbildungen festgelegt. Diese werden jetzt benutzt, um ein*

konkretes Symptom auf der Abstraktionsebene AE zu beschreiben. Dabei ist die Abstraktionsebene eine natürliche Zahl, 1 bezeichnet die Identität, 2 die nächsthöhere Ebene, etc. Daher wird $AE - 1$ mal ein Abstraktionsschritt vorgenommen. Um bestimmte Abstraktionen durchzuführen, müssen noch zusätzliche Symptome an dem Bauteil bekannt sein. Der Einfachheit halber ist das hier nicht dargestellt; diese zusätzlich benötigten Symptome sind aber immer bekannt, wenn dieser Unteralgorithmus aufgerufen wird und könnten daher als Parameter übergeben werden.)*

Setze $S_{abstrakt} := Symptom$.

(*Abstrahiere $AE - 1$ mal*)

Wiederhole $AE - 1$ mal:

Setze $BK_{konkret}$:= Bauteilklasse des in $S_{abstrakt}$ beschriebenen Bauteils.

Setze $BK_{abstrakt}$:= Bauteilklasse des Vaterknotens von $BK_{konkret}$ im Abstraktionsgraph A.

Setze S' auf einen neue, leere Bauteilinstanz der Klasse $BK_{abstrakt}$.

Berechne für den Wert in $S_{abstrakt}$ die Abstraktion mittels der Abstraktionsabbildung, mit der die Kante zwischen $BK_{abstrakt}$ und $BK_{konkret}$ beschriftet ist.

Setze $S_{abstrakt} := S'$.

Ende

$^\wedge S_{abstrakt}$

ALGORITHMUS:Generiere_Hypothesen_aus(*Tripelmenge*, *AE*)

$Hypothesen := \emptyset$.

Wiederhole für jedes Tripel T aus $Tripelmenge$:

H := Generiere_einzelne_Hypothese_aus(T, AE).

Falls $H \neq Fehlschlag$

Dann Nimm das Quadrupel aus dem Tripel und H in $Hypothesen$ auf.

Ende.

$^\wedge Hypothesen$.

ALGORITHMUS:Generiere_einzelne_Hypothese_aus(*Tripel*, *AE*)

*(*Die Tripel, aus denen die Hypothesen erzeugt werden, sind in ,,Finde_möglich_passende_Fälle_auf_Abstraktionsebene ()“ bei der Suche nach passenden Fällen erzeugt worden. Ein Tripel* (S_{Sit}, S_{Fall}, F) *besteht aus einem Symptom der Situation* S_{Sit}, *einem Symptom des Falls* S_{Fall}, *das dazu ähnlich ist und dem Fall* F, *dem* S_{Fall} *entnommen wurde. Zu jeden Tripel wird versucht, die*

Zuordnung zwischen S_{Sit} und S_{Fall} fortzusetzen, bis in der Situation Symptome an Bauteilen gefunden sind, die der Diagnose im Fall F entsprechen. Dabei wird eine Simulation des Falls vorgenommen, um herauszufinden, über welche Zwischenstufen die Diagnose das Symptom S_{Fall} hervorgerufen hat. Eine Wirkungskette wird herausgegriffen und auf der Abstraktionsebene AE von S_{Sit} zurückverfolgt. Falls dies möglich ist, wird eine Hypothese ausgemacht.)*

*(*__1.: Finde im Fall die Wirkungskette__ von der Diagnose zu dem Symptom. Simuliere dazu das durch die Diagnose beschriebene Fehlverhalten.*
Zur Erinnerung: Ein Fall F besteht aus einer Situationsbeschreibung Sit_{Fall} und einer Diagnose D. D ist ebenfalls eine Menge von Symptomen. Sit_{Fall} gliedert sich noch einmal auf in: Int_{Fall} (Symptome, die die Intention beschreiben) und Obs_{Fall} (Observierte Symptome).)*

*(*Die Simulation startet mit der Intention Int_{Fall} und dem Fehlverhalten aus D*)*

Setze $Sim := Int_{Fall} \cup D$.

Erweitere Sim, indem alle Regeln (Verhaltensregeln der Bauteile, sowie Verbindungsregeln in MV) gefeuert werden.

Protokolliere dabei die ausgeführten Regeln in folgender Weise:

Bilde für jede Regel R, die gefeuert hat, das Tripel (S, R, S'), mit:
S: Menge der Symptome, die im Bedingungsteil der Regel aufgetreten waren
(Symptome, die die Regel feuern ließen)
S': Menge der Symptome, die im Aktionsteil der Regel gesetzt wurden.

*(*Aus dem Ergebnis der Simulation wird jetzt eine Wirkungskette zwischen der Diagnose und dem Symptom konstruiert*)*

Sammle diese Tripel in der Menge $Wirkungen$.

Bilde eine Folge $Wirk_{Fall} = (S_1, R_1, S_2, R_2, \ldots, R_{n-1}, S_n)$ mit:
$S_1 \in D$
$S_n = S_{Fall}$
für jeden Schritt S_{i-1}, R_{i-1}, S_i gibt es ein Tripel (S, R, S') aus $Wirkungen$
mit: $S_{i-1} \in S$ und $R_{i-1} = R$ und $S_i \in S'$, $1 < i \le n$.

*(*Diese Folge ist endlich, weil sonst die Simulation nicht gehalten hätte*)*

*(*__2.: Übertragung der gefundenen Wirkungskette__ auf die Situation: Konstruktion einer analogen Wirkungskette in der Situation, d. h. einer gleichen Wirkungskette auf der gerade gewählten Abstraktionsebene AE.*).*

*(*Konstruiere identische Kette für das Symptom aus der Situation S_{Sit}. Diese Kette ist dann auch eine Erklärung für die gefundene Hypothese.*)*

Baue eine identische Kette $Wirk_{Sit}$ auf, indem jeder Schritt $S_{Fall}, R_{Fall}, S'_{Fall}$ aus $Wirk_{Fall}$ übertragen wird. Starte mit $Wirk_{Sit} = (S_{Sit})$.

Das Vorgehen dabei ist: Bekannt ist die Zuordnung eines Symptoms S_{Fall} zu einem Symptom S_{Sit}. Versuche dabei die Zuordnung fortzusetzen, also ein Äquivalent S'_{Sit} zu S'_{Fall} finden. Benutze dabei die Regel R_{Fall}

Unterscheide bei der Übertragung eines Schritts zwei Fälle:

1) Die Regel R_{Fall} ist eine Verhaltensregel eines Bauteils: Das Symptom S'_{Fall} wird auf AE abstrahiert. Es enthält abstrakt einen Port. Dieser Port wird jetzt wieder für das Bauteil in S_{Sit} konkretisiert, man erhält einen konkreten Port.

*(*Da die Abstraktionsfunktion für Ports i. a. nicht umkehrbar ist, können auch mehrere Ports gefunden werden. Darauf wird später eingegangen*)*

2) Die Regel R_{Fall} ist eine Verbindungsregel aus MV.

*(*Hier kann die Übertragung fehlschlagen. S_{Fall} beschreibt einen Wert, der an einem Port eines Bauteils anliegt. Da es sich um eine Verbindung handelt, liegt dieser Wert auch an einem Port eines anderen Bauteils an. Dies ist durch S'_{Fall} beschrieben. Diese Verbindung soll nun auf die Situation übertragen werden, d. h. dort soll ein ähnlicher Port mit einem ähnlichen Port eines ähnlichen Bauteils verbunden sein. „Ähnlich“ bedeutet hier wieder Gleichheit auf Abstraktionsebene AE.*).*

Ähnlich zu 1) wird der Port in der Situation gesucht, der dem Port in S_{Fall} entspricht. Anhand der Verbindungsregeln aus MV wird festgestellt, mit welchem Bauteil das Bauteil in S_{Sit} an diesem Port verbunden ist. Dieses Bauteil wird mit dem verbundenen Bauteil in S'_{Fall} verglichen: sie müssen auf AE gleich sein.

Falls das nicht der Fall ist, dann ^$Fehlschlag$

Sonst ist S'_{Sit} ein Symptom an diesem Bauteil. Die Ausprägung und der Port dieses Symptoms sind für die Übertragung irrelevant.

Falls die Übertragung erfolgreich zu Ende geführt wurde, ist das letzte Symptom der Folge $Wirk_{Sit}$ die Hypothese (Relevant an dieser Hypothese ist zunächst nur das Bauteil)

^Bauteil, das in S_n aus $Wirk_{Sit}$ beschrieben ist.

*(*Anmerkung: Diese Übertragung ist in der Praxis schwieriger, als hier dargestellt. Eine Maschine kann noch durch das Angeben von Baugruppen stärker strukturiert sein. Dadurch treten mehr Verbindungsregeln auf, die nicht übertragen werden. Ein weiteres Problem ist die zusammenfassung von mehreren konkreten Ports zu einem abstrakten Port. Hierbei kann eine abstrakte Verbindung auf der konkreten Ebene mehrere Äquivalente haben. Das hat zur Folge, daß mehrere Wirkungsketten in der Situation generiert werden und auch mehrere Hypothesen gebildet werden können. In diesem Fall liefert dieser Unteralgorithmus mehrere Hypothesen. Das ist durchaus erwünscht, alle Hypothesen werden sowieso noch überprüft.*)*

```
ALGORITHMUS:Prüfe_Hypothesen(Hypothesen, AE)
    Geprüfte Hypothesen := ∅.
    Wiederhole für jede Hypothese H aus Hypothesen:
        Falls Prüfe_einzelne_Hypothese (H, AE) ≠ Fehlschlag
            Dann nimm H in Geprüfte Hypothesen auf
    Ende.
    ^Geprüfte Hypothesen
```

```
ALGORITHMUS:Prüfe_einzelne_Hypothese(Hypothese, AE)
                    (*Die Überprüfung der Hypothesen geschieht durch eine Simulation, bei der für das in der Hypothese beschriebene Bauteil ein zur Diagnose äquivalentes Verhalten angenommen wird. Eine Hypothese ist ein Quadrupel (S_Sit, S_Fall, F, H). Dabei sind S_Sit und S_Fall die Symptome, aufgrund derer der Fall gefunden wurde, F der Fall und H das Bauteil, an dem hypothetisch ein zur Diagnose äquivalenter Defekt angenommen wird.*)
                    (*In der Menge Symptome werden alle die Symptome aufgesammelt, die zur Simulation erforderlich sind.*)
                    (*1. Die Intention in der Situation*)
    Symptome := Int (aus Sit).
                    (*2. Der hypothetische Defekt*)
    Transferiere die Zustandsbeschreibung der Diagnose des Falls auf AE.
    Setze diese Werte in H ein und konkretisiere soweit wie möglich.
    Nimm die Symptome von H in Symptome auf.
                    (*Simuliere den hypothetischen Defekt*)
    Verhindere temporär die Simulation des Bauteils H.
    Simuliert := Simuliere_Verhalten_Aus (Symptome).
                    (*Vergleiche alle Symptome, die in der Situationsbeschreibung bekannt sind, mit dem Ergebnis der Simulation.*)
    Wiederhole für jedes Symptom S ∈ Sit_erw:
        Falls S auf AE nicht identisch ist mit dem entsprechenden Symptom aus Symptome
            Dann ^Fehlschlag
    Ende
```

KAPITEL 4 Die Implementierung

In diesem Kapitel wird die Implementierung von MoCAS/2 beschrieben. Dazu wird zunächst ein Überblick über den Aufbau des Systems gegeben. Dann folgt eine Übersicht über die Repräsentation des Wissens, wobei besonders auf die Darstellung von Verhalten und Abstraktion durch Regeln eingegangen wird. Die Propagierung der Regeln leitet schließlich über zur Implementierung des simulationsfähigen Maschinenmodells.

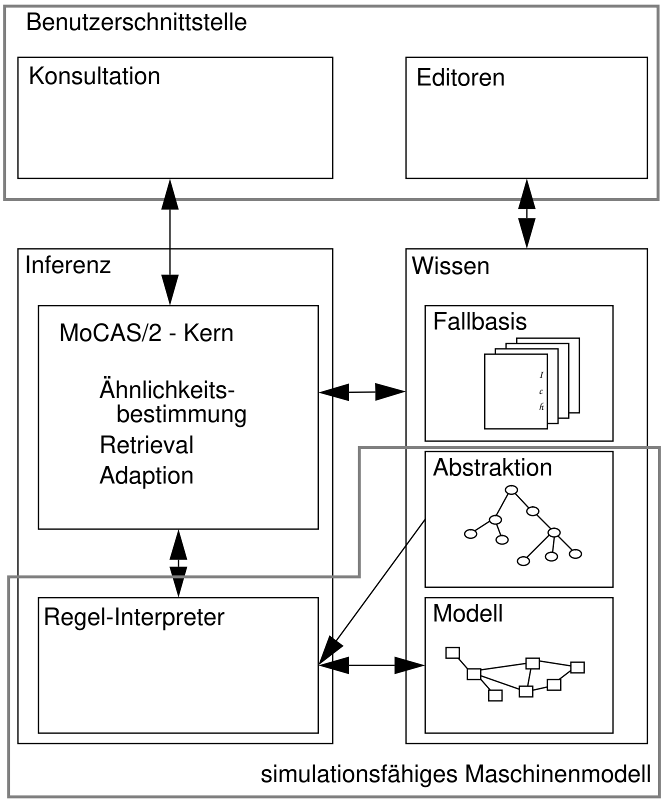
4.1 Implementierungssprache und -voraussetzungen

MoCAS/2 ist implementiert in der Programmiersprache Smalltalk80, Version 4.1. Als notwendige Systemerweiterungen werden ParcPlace VisualWorks (ein graphischer Interfacebuilder) benötigt. Weiterhin benutzt MoCAS/2 das an der Universität Kaiserslautern entwickelte Expertensystemtool CoMo-Kit [Maurer93] und setzt zur Verwaltung der Fallbasis auf Teilen des Systems INRECA auf, die im Rahmen von ESPRIT ebenfalls in Kaiserslautern entwickelt worden sind [Althoff et al. 93] [ManagoAlthoff et al. 93]. Implementierungsplattform waren SUN Sparc Workstations, durch die Portabilität von Smalltalk80 [Goldberg83] [GoldbergRobson83] ist MoCAS/2 aber auch auf anderen Hardwareplattformen (MS-Dos, UNIX allgemein, Macintosh, u. a.) lauffähig.

4.2 Ein Überblick über MoCAS/2

Abbildung 18 gibt eine Übersicht über die wichtigsten Komponenten des Systems:

ABBILDUNG 18 Der Aufbau von MoCAS/2



Die Darstellung des Wissens teilt sich in drei Teile auf:

1. **Die Abstraktionshierarchie.** In dieser Hierarchie ist das Modellwissen organisiert. Dort ist die Struktur und das Verhalten der einzelnen Bauteile und Baugruppen festgehalten, sowie Regeln, mit denen sich Abstraktionsabbildungen durchführen lassen. Die Maschine an sich ist lediglich eine besonders komplexe Baugruppe.
2. **Die Fallbasis.** Um einen Zustand der Maschine darzustellen, bietet es sich natürlich an, eine Instanz der komplexen Baugruppe „Maschine“ mit den entsprechenden Werten zu füllen. Diese Werte können dann noch ausgezeichnet werden, je nachdem ob sie intendiertes Verhalten, observiertes Verhalten oder die Diagnose beschreiben.
3. **Das Modell.** Eine weitere Instanz der Maschine bildet das Modell (das Abbild der Maschine), in dem Simulationen durchgeführt werden können.

Der Regelinterpreter kann im Modell Werte umsetzen, indem er Regeln anwendet, die in der Abstraktionshierarchie beschrieben sind. In Regelform dargestellt sind sowohl Abstraktions- und Konkretisierungs-Abbildungen als auch Verhalten. Dadurch erhält man ein simulationsfähiges Modell, in dem man auf verschiedenen Abstraktionsebenen simulieren kann.

Die Steuerung des Vorgehens wird vom MoCAS/2-Kern vorgenommen, dabei kann der Benutzer den Prozeß durch eine graphische Benutzerschittstelle kontrollieren. Weiterhin kann auch die Wissensbasis durch Editoren verändert werden.

4.3 Wissensrepräsentation

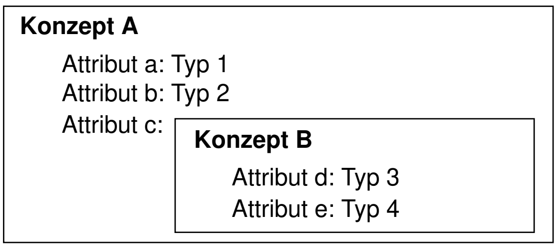
Nun soll ausführlicher auf die Darstellung des Wissens in MoCAS/2 eingegangen werden, dabei sollen folgende Repräsentationen erläutert werden:

1. Bauteile und ihr Aufbau
2. Fälle
3. Verhalten von Bauteilen
   - Regeln
   - Automatisches Propagieren von Regeln
4. Abstraktion
   - Bauteile
   - Werte und Verhalten

4.3.1 Bauteile und ihr Aufbau

Bauteile selbst werden mit Hilfe von CoMo-Kit spezifiziert. CoMo-Kit ist ein Expertensystemtool, von dem in MoCAS/2 hauptsächlich die Wissensrepräsentationskomponente benutzt und teilweise auch erweitert wird. In CoMo-Kit werden Entitäten als *Konzepte* modelliert, ihre Eigenschaften als *Attribute*. Werte von Attributen sind wiederum neue Konzepte. Atomare Konzepte (d. h. Konzepte, die ihrerseits nicht wieder Konzepte als Attributwerte besitzen) werden als *Typen* bezeichnet. Ein Beispiel für eine solche Struktur ist in Abbildung 19 dargestellt.

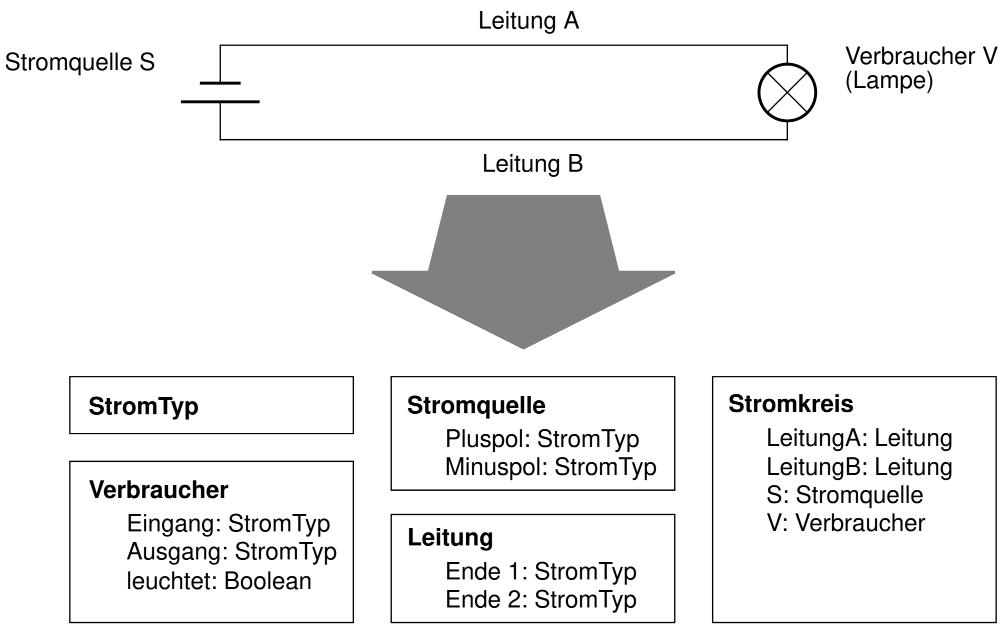
ABBILDUNG 19 Konzeptbeschreibung in CoMo-Kit



CoMo-Kit übersetzt diese Darstellung in eine entsprechende Smalltalk-Klassenstruktur. Dabei werden Konzepte und Typen auf Klassen abgebildet. Attribute entsprechen Instanzenvariablen. Eingehender ist dieser Vorgang in [Beste94] beschrieben.

In MoCAS/2 wird ein Bauteil als Konzept angesehen, seine Ports und States entsprechen Attributen. Jedem Wertebereich eines Ports (bzw. States) entspricht ein CoMo-Kit-Typ. Baugruppen bestehen selbst wieder aus anderen Bauteilen, die durch ein Attribut bezeichnet sind. Diese Repräsentation ist beispielhaft in Abbildung 20 dargestellt. Dort wird die Baugruppe „Stromkreis“ modelliert, die

ABBILDUNG 20 Repräsentation von Bauteilen mit CoMo-Kit



aus vier Unterbauteilen besteht: zwei Leitungen, einer Stromquelle und einem Verbraucher. Die zugehörigen Smalltalk-Klassen werden erzeugt und können nun instantiiert werden.

4.3.2 Fälle

Eine Instanz einer solchen Baugruppenklasse kann nun mit Werten gefüllt werden und gibt damit einen Zustand dieser Baugruppe an. Damit sinnvolle Fälle gebildet werden können, erweitern wir den Stromkreis von oben noch um einen Schalter, der den Zustand „geschaltet“ oder „nicht geschaltet“ annehmen kann:

Stromkreis
    LeitungA: Leitung
        Ende1: 0A
        Ende2: *unbekannt*
    LeitungB: Leitung
        Ende1: *unbekannt*

Ende2: 0A
Schalter: Druckschalter
Eingang: 0A
Ausgang: 0A
Zustand: geschaltet
etc...

Jeder Wert, der in dieser Beschreibung angegeben wird, ist ein Symptom. Einige Symptome werden noch ausgezeichnet: das intendierte Verhalten und die Diagnose. Das geschieht dadurch, daß diese Zustandsbeschreibung in drei verschiedene Beschreibungen aufgespalten wird: je eine für die Diagnose, das intendierte Verhalten und die Symptome.

Für das Beispiel Stromkreis soll jetzt ein Fall angegeben werden, der folgendes beschreibt: die Birne ist durchgebrannt, deshalb brennt sie nicht, obwohl der Schalter betätigt wurde (Unbekannte Werte wurden ausgelassen).

<u>Intendiertes Verhalten</u>

Stromkreis
Schalter: Druckschalter
Zustand: geschaltet

<u>Diagnose</u>

Stromkreis
V: Verbraucher
Eingang: 0A
Ausgang: 0A
leuchtet: false

<u>Symptome</u>

Stromkreis
V: Verbraucher
leuchtet: false
LeitungA: Leitung
Ende1: 0A

4.3.3 Verhalten von Bauteilen

Das Verhalten von Bauteilen wird durch Regeln beschrieben. Da das Verhalten für alle Bauteile einer Klasse gleich ist, können die Regeln in der Bauteilklasse festgelegt werden. Eine Regel hat die Form: Vorbedingungen → Nachbedingungen mit der Bedeutung, daß die Nachbedingungen (Aktionen) gültig sein sollen, wenn alle Vorbedingungen gelten. Zunächst soll jetzt die Implementierung der Bedingungen, dann die der Regeln beschrieben werden, um dann zur automatischen Propagierung der Regeln zu kommen.

4.3.3.1 Bedingungen und Variablen

Um eine objektorientiere Implementierung von Regeln zu realisieren, bietet es sich an, eine Klasse „Bedingung“ zu definieren. Diese Klasse hat drei Instanzenvariablen:

1. Einen Bezeichner, der angibt, auf welche Instanzenvariable eines anderen Objekts sich die Bedingung beziehen soll.
2. Ein Wert, der mit dem Wert des anderen Objekts verglichen werden soll
3. Die Bedingung, mit der der Vergleich ausgeführt werden soll.

Als Beispiel soll eine Bedingung erstellt werden, die sich auf den oben vorgestellten Stromkreis bezieht und die prüfen soll, ob das Ende1 der Leitung A keinen Strom führt. Der Bezeichner für den Slot lautet dann: „LeitungA.Ende1“, die Vergleichsoperation ist „=“ und der Wert, mit dem verglichen wird, ist „0A“.

Um eine Bedingung mit einem Wert zu matchen, also eine Bedingung der Form „LeitungA.Ende1 = X“ darzustellen, führt man eine weitere Klasse „Variable“ ein, die einen Slot „Wert“ besitzt. Das mutet zunächst umständlich an, hat seine Begründung aber darin, daß ein Variablen-objekt in mehreren Bedingungen benutzt wird. Wird dann bei der Auswertung einer Bedingung der Wert der Variable gebunden, ist er automatisch in allen anderen Bedingungen an diesen Wert gebunden.

Um mit Bedingungen zu arbeiten, implementiert man zwei Methoden in der Klasse Bedingung: „prüfeFür: <objekt>“ und „weiseWertZu: <objekt>“. Die erste Methode liefert einen Boole’schen Wert, je nachdem, ob die Bedingung erfüllt ist oder nicht. Als Seiteneffekt kann sie Variablenbindungen durchführen. Die zweite Methode beschreibt eine Aktion, d. h. der Wert der Bedingung wird dem in ihr spezifizierten Slot zugewiesen.

Da die Modellierung der Domäne in MoCAS/2 qualitativ (z. B. „an“/„aus“) erfolgt, beschränkt sich die Vergleichsoperation in den Bedingungen auf „=“.

4.3.3.2 Regeln

Auch für Regeln wird eine eigene Klasse eingeführt. Die Klasse „Regel“ hat die zwei Instanzenvariablen „Vorbedingungen“ und „Aktionen“, die beide eine Liste von Bedingungen enthalten.

Um eine Regel anzuwenden, wird die Methode „wendeAnAuf: <objekt>“ implementiert. Sie iteriert über alle Vorbedingungen, indem sie ihnen die Methode „prüfeFür: <objekt>“ schickt. Evaluieren alle Aufrufe zu „wahr“, werden alle Aktionen mit der Methode „weiseWertZu: <objekt>“ ausgeführt.

4.3.3.3 Die automatische Auswertung von Regeln

Die automatische Auswertung von Regeln sollte in einer objektorientierten Implementierung von den Objekten unterstützt werden, auf die die Regeln angewendet werden sollen. Sobald einer Instanzenvariable ein neuer Wert zugewiesen wird, werden alle Regeln, die sich in ihren Bedingungen auf diese Variable beziehen, gefeuert.

Diese Unterstützung der Regelausführung erfordert eine Erweiterung der Objekte. Es werden zwei neue Instanzenvariablen angelegt: „PropagierungsVerzeichnis“ und „RückverfolgungsVerzeichnis“. Im PropagierungsVerzeichnis findet sich unter dem Namen einer Instanzenvariable eine Liste von Regeln, die angestoßen werden sollen. Kann eine dieser Regeln feuern (und damit Werte im Objekt umsetzen), trägt sie im Rückverfolgungsverzeichnis unter dem Namen der geänderten Instanzenvariable einen Verweis auf sich selbst ein. Dadurch kann die Abarbeitung von Regeln zurückverfolgt werden, um herauszufinden, wie ein bestimmter Wert zustande gekommen ist. Will man die Rücknahme von Regeln unterstützen, kann man hier auch den ursprünglichen (überschriebenen) Wert der Instanzenvariable eintragen.

4.3.4 Abstraktion von Bauteilen und deren Verhalten

Bisher wurde dargestellt, wie die Komponenten der Domäne und ihr Verhalten repräsentiert werden. Ein zentraler Punkt in MoCAS/2 ist das Bereitstellen von Abstraktionen. Zunächst soll gezeigt werden, wie Komponenten an sich abstrahiert werden. Dabei muß auch eine Abstraktion von Symptomen (Werten) erfolgen. Schließlich benötigt man noch passend dazu eine Abstraktion der Verhaltensregeln.

4.3.4.1 Bauteilabstraktion

Bauteile werden, wie anfangs erläutert, mit Hilfe von CoMo-Kit spezifiziert und letztendlich auf Smalltalk-Klassen abgebildet. Zur Definition von Konzepthierarchien bietet CoMo-Kit eine IsA-Relation an, die auf die Klassenhierarchie in Smalltalk abgebildet wird. Ein Beispiel für eine solche Abstraktion ist in

ABBILDUNG 21 IsA-Darstellung in MoCAS/2 mit CoMo-Kit

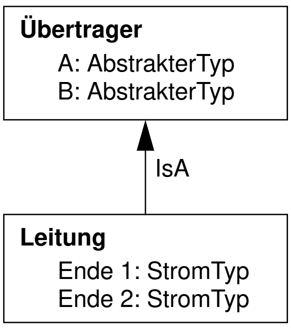


Abbildung 21 dargestellt. Die Umsetzung nach Smalltalk macht die Unterschiede in der Semantik zwischen CoMo-Kit und MoCAS/2 deutlich: Die von CoMo-Kit erzeugte Klasse „Leitung“ erbt die Instanzenvariablen der Klasse „Übertrager“. Eine Unterklasse stellt hier eine Verfeinerung der Oberklasse dar, indem sie mehr Instanzenvariablen besitzt; die Idee dahinter ist, daß gleichartige Strukturen und Methoden von Klassen nur einmal in einer gemeinsamen Oberklasse spezifiziert werden. Das ist die natürliche Art objektorientiert zu programmieren bzw. zu modellieren und oft hilfreich.

Die Abstraktion in MoCAS/2 hingegen beschreibt einen Wechsel der Betrachtungsebene. Ein Objekt, das eben noch als Leitung betrachtet wurde, wird nun mit anderen Mitteln (Instanzenvariablen) in einer alternativen, abstrakteren Betrachtungsweise beschrieben.

Um nun beides - sowohl die Is-A-Hierarchie als auch die Abstraktionshierarchie - nutzen zu können, wurde CoMo-Kit für MoCAS/2 um eine neue Art von Attribut erweitert. Dieses MoCASAttribut ist technisch eine Unterklasse des CoMo-Kit-Attributs, wird aber im Gegensatz zu diesem nach außen hin nicht vererbt. Natürlich wird es, was ja für die objektorientierte Programmierung und den Wechsel von Betrachtungsweisen sinnvoll ist, intern dennoch vererbt - nur ist dies für den Benutzer unsichtbar. Es werden immer nur die Attributwerte angezeigt, die für die Abstraktionsebene gelten, auf der der Benutzer das Objekt betrachtet.

4.3.4.2 Abstraktion von Werten

Wenn die Betrachtungsebene für eine Komponente wechselt, ändern sich auch die Werte, die an ihr gemessen werden. Die Leitung, durch die am Ende1 3A Strom fließt, ist dann (abstrahiert) ein Übertrager, an dessen Port A der abstrakte Wert „an“ anliegt. Passend zum konkreten Wert am konkreten Port „Ende1“ muß also ein abstrakter Wert an Port „A“ gesetzt werden.

Diese Abbildung läßt sich durch eine Regel beschreiben. Das hat den angenehmen Effekt, daß der Propagierungsmechanismus für Regeln auch hier benutzt werden kann, falls eine eindeutige Konkretisierung des abstrakten Werts möglich ist, kann auch eine Regel dafür angegeben werden. Damit ist es egal, auf welcher Abstraktionsebene man sich befindet - alle anderen eindeutig bestimmbaren Abstraktionsebenen werden automatisch mit berechnet.

4.3.4.3 Abstraktion von Verhaltensregeln

Auch die Verhaltensregeln sind in einer Abstraktionshierarchie angeordnet. Bei der Implementierung ist das dadurch möglich, daß MoCAS/2 selbst wieder auch zum größten Teil mit CoMo-Kit spezifiziert worden ist. Dadurch können Regeln mit den Mitteln von CoMo-Kit in Hierarchien organisiert werden.

Diese Regelhierarchie ist parallel zur Bauteilhierarchie angeordnet, was die Handhabung erleichtert. Ein Abstraktionsschritt in der Bauteilhierarchie entspricht dann genau einem Abstraktionsschritt in der Regelhierarchie. Eine solche Organisation ist aber nicht zwingend, man kann sich vorstellen, daß die Verhaltensregeln für ein einzelnes Bauteil noch über mehrere Zwischenstufen abstrahiert werden.

4.4 Die Simulationskomponente

Die automatische Anwendung von Verhaltensregeln, so wie sie im letzten Abschnitt beschrieben wurde, ist besonders dann sinnvoll, wenn man das Verhalten von Bauteilen simulieren will. Wenn gleichzeitig auch noch die Abstraktionsregeln aktiv sind, hat das den Effekt, daß man gleichzeitig auf verschiedenen Abstraktionsebenen abstrahieren kann. Man kann eine abstrakte, effiziente Simulation auf einer konkreten Ebene fortsetzen, wenn das nötig erscheint.

Ein Problem, das auftreten kann, wenn die Regeln automatisch gefeuert werden sobald ein Wert gesetzt wird, ergibt sich, falls dieser Wert wieder überschrieben werden kann. Dann war die Ausführung aller Regeln unnötig, die aufgrund der ersten Änderung durchgeführt wurde. Dem ist allerdings entgegenzuhalten, daß man normalerweise eine Modellierung wählt, in der eine solche Überschreibung nicht erwünscht ist, da Verhaltensregeln eindeutig feuern und nicht ihre Werte gegenseitig überschreiben sollten. Zum anderen ist die Implementierung der Simulation genügend schnell, um auch kleinere Umwege bei der Regelanwendung zu tolerieren.[1] Dieser Nachteil wird durch die Einfachheit des Simulationsverfahrens mehr als wettgemacht.

1. Auf einer SUN SPARC ELC Workstation benötigt man etwa eine Sekunde zur Ausführung von eintausend Regeln.

KAPITEL 5 Bewertung und Ausblick

In diesem Kapitel weden zunächst die Ergebnisse der Arbeit noch einmal kurz zusammengefaßt und bewertet. Danach wird der in dieser Arbeit vorgestellte Ansatz mit anderen, verwandten Ansätzen verglichen, um Unterschiede herauszuarbeiten. Schließlich folgt ein Ausblick auf Möglichkeiten der Weiterentwicklung, sowohl für das System selbst als auch für andere Problemgebiete.

5.1 Ergebnisse der Arbeit

In dieser Arbeit wurde ein Ansatz zur Diagnostik in technischen Domänen entwickelt und implementiert. Der Ansatz ermöglicht folgendes:

1. **Eine semantische Ähnlichkeitsbestimmung von Diagnosefällen** durch die Verwendung von Modellwissen. Fälle, die identische Diagnosen mittels verschiedener Symptome beschreiben, können dadurch miteinander identifiziert werden.
2. **Fallübertragung.** Fälle, die ein ähnliches Fehlverhalten an physikalisch unterschiedlichen Bauteilen beschreiben, können ebenfalls miteinander identifiziert werden.

   Das wichtigste Hilfsmittel dazu ist die Verwendung einer Abstraktionshierarchie. Ähnlichkeit wird damit zurückgeführt auf eine Gleichheit auf höheren Abstraktionsebenen. Damit die semantische Ähnlichkeitsbestimmung auch auf höheren Abstraktionsebenen funktioniert, muß alles Wissen, das in Punkt 1 benutzt wird, analog hierarchisch organisiert werden.
3. **Indexierung der Fallbasis.** Größere Fallbasen erfordern Verwaltungsmechanismen, die einen effizienten Umgang mit ihnen erst möglich machen. Daher wurde ein Indizierungsverfahren entwickelt, mit dem gezielt auf diejenigen Fälle zugegriffen werden kann, deren Nutzung erfolgversprechend erscheint.

Um einerseits zu zeigen, daß der Ansatz realisierbar ist und um andererseits praktische Erfahrungen zu erhalten, wurde ein System implementiert. Dieses System wurde an einer Beispielmaschine getestet und erzielte dort die gewünschten Resultate. Fälle konnten übertragen und die Ähnlichkeit über Wirkunsmechanismen in der Maschine bestimmt werden.

5.2 Bewertung

Den Vorteilen, die der Ansatz in MoCAS/2 bringt, stehen einige Nachteile und Beschränkungen entgegen. Auf beides soll nun eingegangen werden, beginnend mit den Vorteilen:

- **Es werden nur sinnvolle Hypothesen generiert**. Alle Hypothesen, die das System vorschlägt, sind durch eine Simulation auf Konsistenz mit dem Modellwissen geprüft.
- **Die Fallbasis kann wesentlich verkleinert werden.** Gegenüber einem herkömmlichen fallbasierten Ansatz (Instance-Based-Learning) oder verschiedenen induktiven Techniken benötigt MoCAS/2 wesentlich weniger Fallbeispiele. Das resultiert aus den beiden Fähigkeiten:
  - Semantische Ähnlichkeitsbestimmung an identischen Bauteilen.
  - Fallübertragung auf physikalisch andere Bauteile.
- **Zu den generierten Hypothesen können Erklärungen angegeben werden.** Der Vorgang der Hypothesengenerierung geschieht explizit und ist daher transparent und nachvollziehbar; insbesondere kann eine Begründung hergeleitet werden, die für Benutzer verständlich ist.

Nachteile für MoCAS/2 ergeben sich daraus, daß der Ansatz relativ aufwendig ist, es entstehen hohe Kosten für Wissensakquisition und benötigtem Rechenaufwand:

- **Eine Simulation für große Maschinen durchzuführen kann aufwendig werden.** Je genauer die Simulation sein soll und je komplexer die Maschine ist, umso rechenintensiver ist die Simulation. Wenn man statt qualitativ quantitativ simulieren möchte, verschärft sich das Problem noch.
- **Es wird viel Wissen über die Domäne benötigt.** Die wechselseitigen Wirkungsbeziehungen in der Domäne müssen bekannt und gut verstanden sein. Außerdem muß das Wissen noch eingegeben werden, was selbst mit geeigneten Benutzerschnittstellen sehr zeitintensiv sein kann.

Aus diesen Punkten läßt sich ablesen, wann der Ansatz geeignet ist. Absolute Voraussetzung ist eine Domäne, in der man die Wirkungszusammenhänge kennt, die für die Problemstellung relevant sind. Danach muß man abwägen zwischen den Vor- und Nachteilen des Ansatzes: Wenn Fehldiagnosen unbedingt ausgeschlossen sein müssen, oder wenn überhaupt nur wenige Fälle zur Verfügung stehen, wird man den Aufwand für Wissensrepräsentation und -akquisition in Kauf nehmen.

Auf der anderen Seite kann es sein, daß dieses Wissen schon in maschinell verwertbarer Form verfügbar ist. Das kann beispielsweise in technischen Domänen der Fall sein, wenn CAD-erstellte Konstruktionspläne vorhanden sind. Bauteilspezifikationen über das Ein-Ausgabeverhalten werden sind genormt oder von den Herstellern angegeben. In solchen Anwendungen ist die Nutzung des hier vorgestellten Ansatzes deutlich einfacher. Aber auch hier ist abzuwägen, ob die Vorteile von MoCAS/2 wirklich zum Tragen kommen: Wenn gelegentliche Fehldiagnosen nicht störend sind oder wenn auf Fallübertragung kein Wert gelegt wird, wird man sicher einen einfacheren Ansatz bevorzugen. Typische Anwendungen, in denen auf Fallübertragung verzichtet werden kann, ergeben sich, wenn in der Domäne kaum gleichartige Strukturen vorhanden sind. Ebenso kann es sein, daß sich eine Fallbeschreibung ergibt, die das Clustering unterstützt, welches implizit oder explizit von vielen Algorithmen des maschinellen Lernens durchgeführt wird. Auch dann kann sicher ein Ansatz gewählt werden, der weniger aufwendig ist.

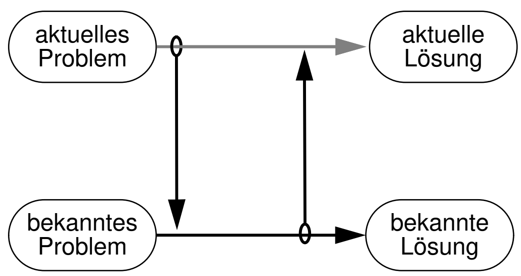
5.3 Vergleich mit anderen Ansätzen

In diesem Abschnitt wird MoCAS/2 mit drei anderen Ansätzen verglichen, zu denen eine gewisse Verwandschaft besteht: Derivational Analogy [Carbonell86], [VelosoCarbonell93] und dem Ansatz in PROTOS. Sie werden jeweils zunächst kurz vorgestellt, bevor auf die Unterschiede zu MoCAS/2 eingegangen wird.

5.3.1 Derivational Analogy

Wie in dieser Arbeit bereits ausführlich erläutert wurde, ist oft der Problemstellung allein noch nicht anzusehen, welche Fälle ähnlich sind, d. h. welche Fälle zur Lösungsfindung verwendet werden können. Ähnlichkeiten zu anderen Fällen treten erst zutage, wenn man beginnt, das Problem „herkömmlich“ zu lösen. Dabei bemerkt man dann, daß bestimmte Problemlöseschritte bei bereits bekannten Problemen ähnlich durchgeführt wurden. Die Ähnlichkeit zwischen Problem und Fall wird also mit Hilfe des Lösungswegs festgestellt. Der bekannte Lösungsweg kann dann den Aufbau des gesuchten Lösungswegs unterstützen (Abbildung 22)

ABBILDUNG 22 Das Vorgehen bei Derivational Analogy



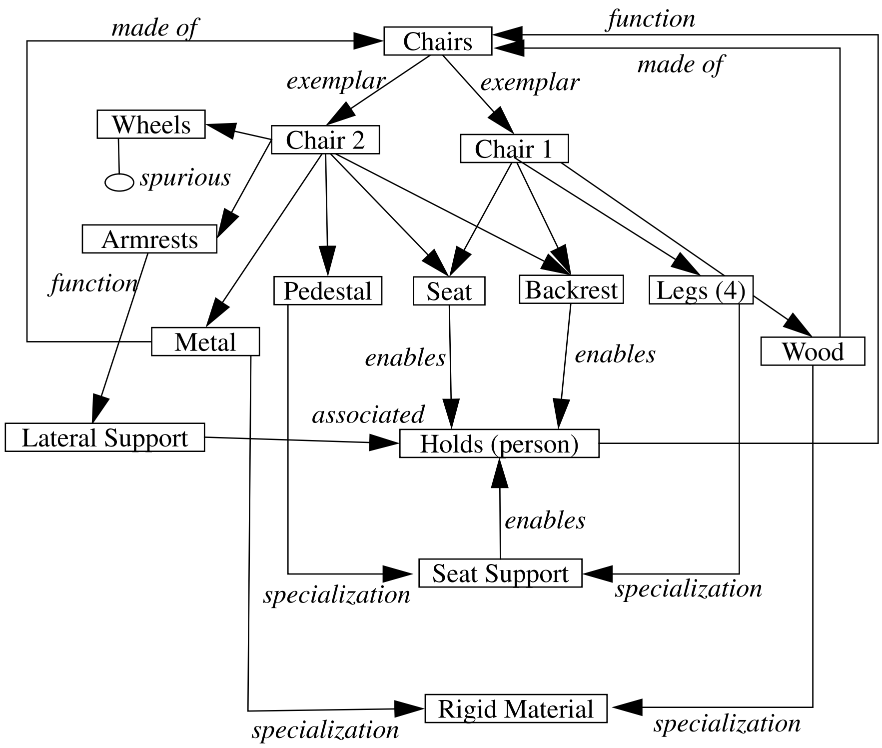
Man kann man MoCAS/2 als einen speziellen Ansatz, der Derivational Analogy verwirklicht, auffassen. Die Berechnung einer „herkömmlichen“ Lösung beginnt damit, herauszufinden, welche Bauteile sich anders als erwünscht verhalten haben. Mit diesem Wissen ist es dann möglich, ähnliche Fälle zu finden. Diese Überlegung erlaubt es allerdings, viele Arten von Indexing als Derivational Analogy anzusehen. Es ist keine scharfe Grenze zwischen Derivational Analogy und Fallsuche mit anschließender Lösungsanpassung (Transformational Analogy) zu ziehen, da eine Lösungsanpassung in der Regel den Lösungsweg berücksichtigen muß.

5.3.2 PROTOS

PROTOS wurde von Ray Bareiss und Bruce Porter and der University of Texas in Austin entwickelt [Bareiss89], [PorterBareiss89]. Aufgabe dieses Systems war es, Klassifikationen während der Arbeit mit dem System zu lernen. Die Vorgehensweise dabei war folgende: Ein menschlicher Lehrer präsentiert dem System Fallbeispiele. Falls das System falsch klassifiziert, gibt der Lehrer eine Erklärung ein, durch die das System den Fall in Zukunft richtig klassifizieren kann. Dadurch erlernt PROTOS Konzepte und akquiriert Domänenwissen und verbessert stetig seine Klassifikationsfähigkeit, wobei es in der Lage ist, Klassifikationen zu erklären.

Fälle sind in PROTOS als semantisches Netz repräsentiert, eine Erklärung begründet, warum sich zwei Fälle ähnlich sind. Dazu werden vom Benutzer Relationen eingeführt, die als Links in das Netzwerk eingebaut werden. PROTOS gibt dabei keine Struktur der Erklärungen vor, es gibt zwar einige vordefinierte Relationen, sie sind allerdings sehr allgemein gehalten, um das System universell einsetzen zu können. Ein Beispiel dafür, wie PROTOS das Konzept „Stuhl“ lernt, ist in Abbildung 23 gegeben.

ABBILDUNG 23 Die Struktur, mit der PROTOS das Konzept „Stuhl“ kategorisiert (Nach [BareissPorterWier87])



Kanten ohne Beschriftung bezeichnen Eigenschaften der Objekte. Die beiden Stühle „Stuhl 1“ und „Stuhl 2“ haben teilweise gemeinsame Eigenschaften (Sitz, Lehne), teilweise aber auch verschiedene Eigenschaften: Stuhl 1 ist aus Metall, Stuhl 2 hingegen aus Holz. Eine Erklärung in PROTOS versucht nun, diese unterschiedlichen Eigenschaften durch einen Pfad im semantischen Netz zu verbinden. Das gelingt, da sowohl Holz als auch Metall Spezialisierungen eines harten Materials sind. Hätte PROTOS diese Verbindung selbst nicht herstellen können, dann hätte das System den Benutzer nach einer solchen Erklärung gefragt. Da man in einem großen Netzwerk fast alle Knoten mittelbar miteinander verbunden sind, werden die Kanten in PROTOS gewichtet. Eine Erklärung (Eine Verbindung zwischen Eigenschaften) muß dann einen bestimmten Schwellwert überschreiten.

Erklärungsstrukturen werden auch in MoCAS/2 aufgebaut, allerdings sind sie anderer Art. Eine Erklärung in MoCAS/2 ist eine Begründung, warum eine Diagnose bestimmte Symptome hervorgerufen hat, diese Erklärung wird also innerhalb des Falls aufgebaut. Zwei Fälle sind ähnlich, wenn ihre Erklärungen ähnlich - also auf einer höheren Abstraktionsebene identisch - sind. In PROTOS hingegen werden Erklärungen zwischen zwei Fällen aufgebaut, sie beschreiben, warum sich die Fälle ähnlich sind.

Weiterhin ist in MoCAS/2 durch die innere Struktur der Domäne (Weschselwirkungen, Verbindungen der Bauteile) auch eine Struktur für die Erklärungen bereits vorgegeben, so daß sie vom System selbst generiert werden können. Die Intention von MoCAS/2 ist ja auch eine andere als die von PROTOS. Dort liegt das Ziel in der Akquisition von Wissen und dessen Anwendung, dort soll die Art und Struktur einer Erklärung während der Benutzung noch herausgefunden werden. In MoCAS/2 wird die Wissensakquisition als bereits vorgenommen vorausgesetzt.

Problematisch in PROTOS ist die Möglichkeit für den Benutzer, beliebige Erklärungsstrukturen durch Relationen anzugeben und sie zu gewichten. Das führt einerseits zu unübersichtlichen Strukturen, andererseits muß sich der Benutzer sehr klar darüber sein, auf welche Art eine Erklärung aufgebaut werden soll. So wurden bei der Anwendung von PROTOS (im Bereich Audiologie) vom Experten oft zu einfache Erklärungen eingegeben.

5.4 Ausblick

Der in dieser Arbeit aufgezeigte Ansatz läßt sich in zwei Richtungen weiterentwickeln: Zum einen kann das Diagnosesystem MoCAS/2 noch erweitert und leistungsfähiger gemacht werden, zum anderen bieten sich interessante Möglichkeiten, die Ergebnisse auch für andere Gebiete, wie etwa die Planung, nutzbar zu machen. Auf beides soll jetzt eingegangen werden.

5.4.1 Erweitungsmöglichkeiten des Diagnosesystems MoCAS/2

1. **Verfeinerung des simulationsfähigen Modells.** MoCAS/2 arbeitet in der hier vorgestellten Form mit einem qualitativen Modell, das Zeitabhängigkeiten weitgehend unberücksichtigt läßt. Eine Erweiterung auf quantitative Modelle oder auf Modelle, die das zeitliche Verhalten von Bauteilen simulieren, wäre mög-

lich. Dabei ist allerdings abzuwägen, daß solche Modelle einen sehr viel höheren Aufwand bei der Simulation erfordern und nicht trivial zu realisieren sind.

2. **Entwicklung von Strategien zur Testauswahl.** Der vorgestellte Ansatz behandelt bisher nur die Generierung von Hypothesen und läßt die Testauswahl unberücksichtigt. Das Vorhandene Modellwissen könnte aber auch dazu sinnvoll eingesetzt werden. Falls auch für die Testauswahl ein fallbasierter Ansatz gewählt würde, ließen sich mit ähnlichem Vorgehen wie bei der Hypothesengenerierung Tests generieren. Dazu wäre dann beispielsweise eine zweite Abstraktionshierarchie sinnvoll, die statt nach Funktionalität (wie zur Hypothesengenerierung) nach Störanfälligkeit (mechanische vs. elektronische Bauteile) abstrahiert.
3. **Verbesserung der Indexierungsstrategie.** Die Indexierung wird heuristisch vorgenommen und kann daher sicher noch verbessert werden. Dazu sind allerdings noch ausführliche Test notwendig.
4. **Ausführliche Tests.** Alle Weiterentwicklungen sollten von Tests begleitet werden, die eine Abschätzung der Verbesserungen erlauben. Insbesondere stehen noch Tests aus, die MoCAS/2 gegen andere Verfahren des Maschinellen Lernens abgrenzen, selbst wenn die Ergebnisse vorhersehbar sind.
5. **Beurteilung der Fallübertragung.** Ein weiterer Testpunkt ist, festzustellen, bis zu welcher Abstraktionsebene eine Fallübertragung überhaupt noch sinnvoll ist. Eine Übertragung zwischen zwei gleichartigen Stromkreisen ist sicherlich sinnvoll, bei einer Fallübertragung zwischen einem Stromkreis und einem Wasserkreislauf, die in MoCAS/2 durchaus möglich ist, ist der Nutzen jedoch fraglich.

5.4.2 Parallelen zur Planung

Bei der Arbeit an dem Ansatz in MoCAS/2 wurden einige Parallelen zur Planung deutlich. Deshalb sollen hier Planung und die Diagnose in MoCAS/2 gegenübergestellt werden. Sie sind ausfürlich in [BergmannPewsWilke94] und [BergmannWilke94] beschrieben.

Die Verhaltensweisen von Bauteilen entsprechen einzelnen Operatoren, in beiden Bereichen bewirken sie eine Änderung von Zuständen der Domäne. Die Problemstellung ist ähnlich. Diagnose: Ein Startzustand der Maschine ist bekannt (Beschreibung des intendierten Verhaltens), ein Endzustand wurde erreicht (beobachtete Symptome). Beide Zustände müssen nicht vollständig angegeben sein. Gesucht ist jetzt ein (Fehl-) Verhalten irgendeines Bauteils, durch das erklärt werden kann, wie der Anfangszustand in den Endzustand transformiert wurde. Falls es mehrere Lösungen gibt, kann man die Güte einer Lösung kaum bewerten. Es ist allenfalls möglich, aufgrund von bisherigen Ausfallquoten für einzelne Bauteile Ausfallwahrscheinlichkeiten anzugeben. Planung: Hier sind Start- und Zielzustand vorgegeben, gesucht wird eine Folge von Operatoren, die den ersten Zustand in den zweiten überführt. Oft ist es möglich Kriterien anzugeben, mit deren Hilfe man abschätzen kann, wie gut ein Plan ist.

Der wesentliche Unterschied in der Planung liegt im Verlust der internen Struktur der Domäne. Bei der Diagnose sind die Bauteile fest angeordnet, ihre Reihenfolge und Wechselwirkungen sind vorgegeben. Man kann aus einem Diagnosefall einen ähnlichen Fall erzeugen, indem man das Verhalten eines Bauteils durch ein neues, ähnliches Verhalten ersetzt. Genauso kann man einen Plan modifizieren, indem man einen einzelnen Operator durch einen ähnlichen Operator ersetzt. Während dieses Vorgehen in der Diagnose die Regel ist, tritt bei der Planung das Problem auf, einen Teilplan vollkommen neu zu erstellen, der von seiner Struktur her unbekannt ist. In der Diagnose hat das seine Entsprechung darin, daß dort das Verhalten von Baugruppen mit dem einzelner Bauteile identifiziert wird. Dort gibt es neben der in der Maschine vorgegebenen Zusammenfassung der Bauteile zu Baugruppen noch viele andere Möglichkeiten, einzelne Bauteile zu gruppieren.

ANHANG 1

Ein Beispiel einer Diagnosesitzung

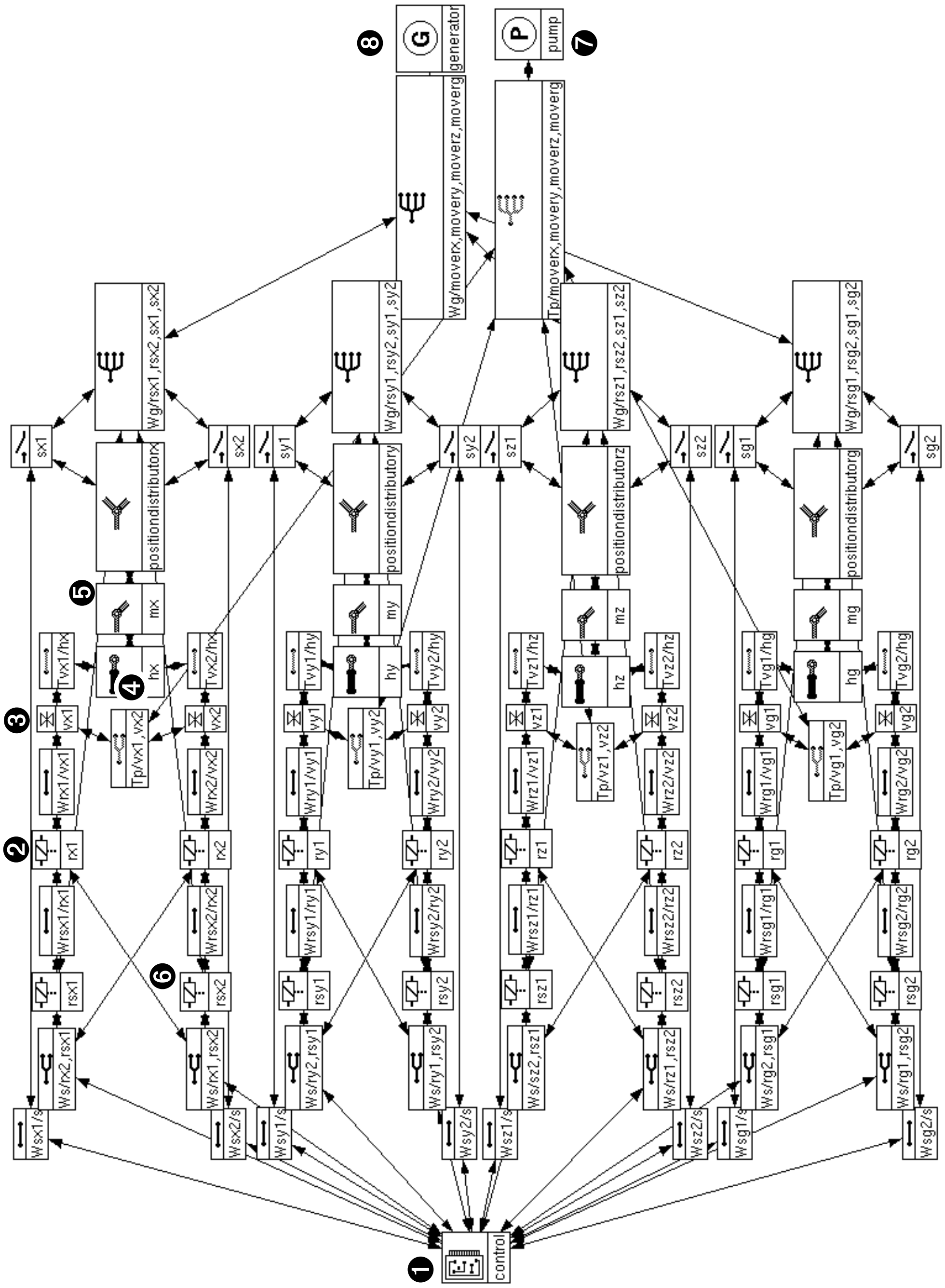
In diesem Kapitel wird eine Diagnosesitzung mit MoCAS/2 beschrieben. Dazu wird zunächst einmal die Maschine und ein Störfall beschrieben, anschließend folgen eine Reihe von Bildschirmkopien einer Diagnosesitzung in der dieser Störfall bearbeitet wird.

1.1 Die Situation

Die Maschine, die jetzt vorgestellt werden soll, diente auch als Beispieldomäne bei der Entwiickung von MoCAS/2. Sie ist relativ einfach aufgebaut und besteht nur aus 110 Bauteilen mit insgesamt etwa 350 Ports. Die Wertebereiche dieser Ports sind qualitativ modelliert und können zwischen drei und sechs Werte annehmen. Trotz der recht einfachen Struktur sind für diese Maschine etwa 10^{138} verschiedene Fälle angebbar, das ist weitaus mehr als die Anzahl aller Elektronen, Protonen und Neutronen im ganzen Universum und macht die Notwendigkeit deutlich, kleine Fallbasen effektiv zu nutzen.

Die Maschine, deren Schaltplan in Abbildung 24, “Aufbau und Funktionsweise des Werkzeuggreifers,” auf Seite 76 dargestellt ist, funktioniert im Wesentlichen so: Ein Steuersignal wird von der Kontrolleinheit ① an ein Relais ② gegeben. Dieses Relais schaltet ein Ventil ③, welches wiederum Druck zu einer Hydraulik ④ leitet. Diese Hydraulik bewegt eine Mechanik ⑤, den Werkzeuggreifer. Diese Ansteuerung gibt es sowohl für Vorwärts- als auch für Rückwärtsbewegungen. Damit nicht gleichzeitig die Bewegung in beide Richtungen ausgelöst werden kann, wird - sobald das Relais für eine Richtung angesteuert wird - ein Sicherheitsrelais ⑥ in der Gegenrichtung betätigt, das die Steuersignale für die Gegenrichtung kappt. Die Energieversorgung des Systems wird durch eine Hydraulikpumpe ⑦ und einen Generator ⑧ gewährleistet.

ABBILDUNG 24 Aufbau und Funktionsweise des Werkzeuggreifers



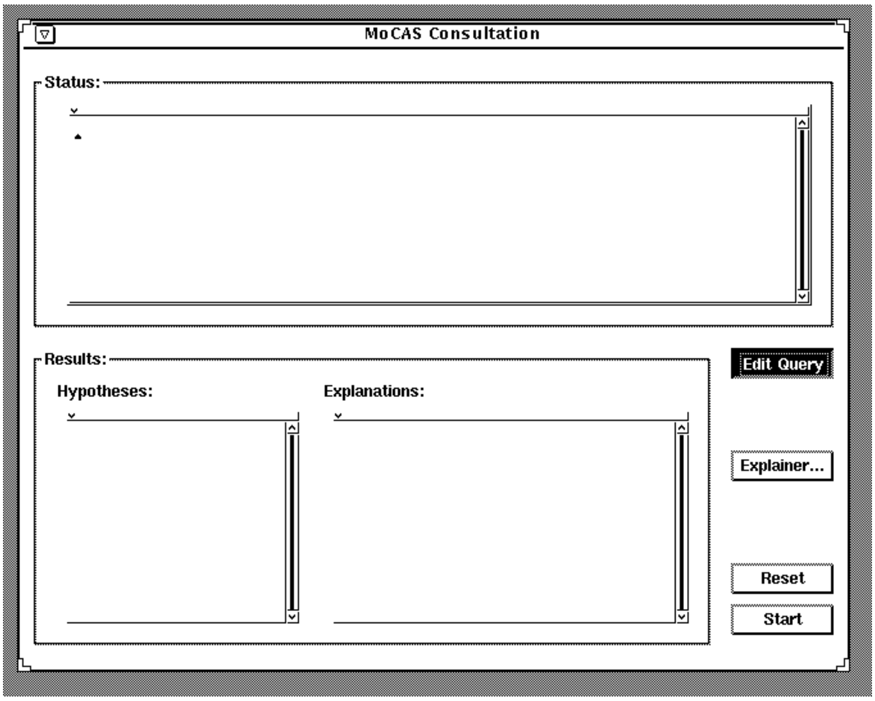
Wir nehmen jetzt als Störfall an, daß das Ventil vx_1 3 defekt ist, d. h. es schließt nicht, wenn es vom Relais rx_1 2 angesteuert wird. Daher geht kein Druck an die Hydraulik, der Greifer bewegt sich nicht.

Die gesamte Störfallbeschreibung sieht nun so aus: Der Generator und die Hydraulikpumpe sind eingeschaltet, der Greifer steht auf der Position d2. Von der Steuereinheit geben wir nun die Steuerbefehle zum Vorwärtsfahren, der Greifer soll sich auf die Position d3 bewegen. Das tut er aber nicht, er bleibt unverändert auf der Position d2. Um die Ursache für diesen Störfall zu finden, wird eine Diagnosesitzung mit MoCAS/2 durchgeführt

1.2 Die Diagnosesitzung

MoCAS/2 zeigt zunächst das Fenster für eine Konsultation. (Abbildung 25). Da

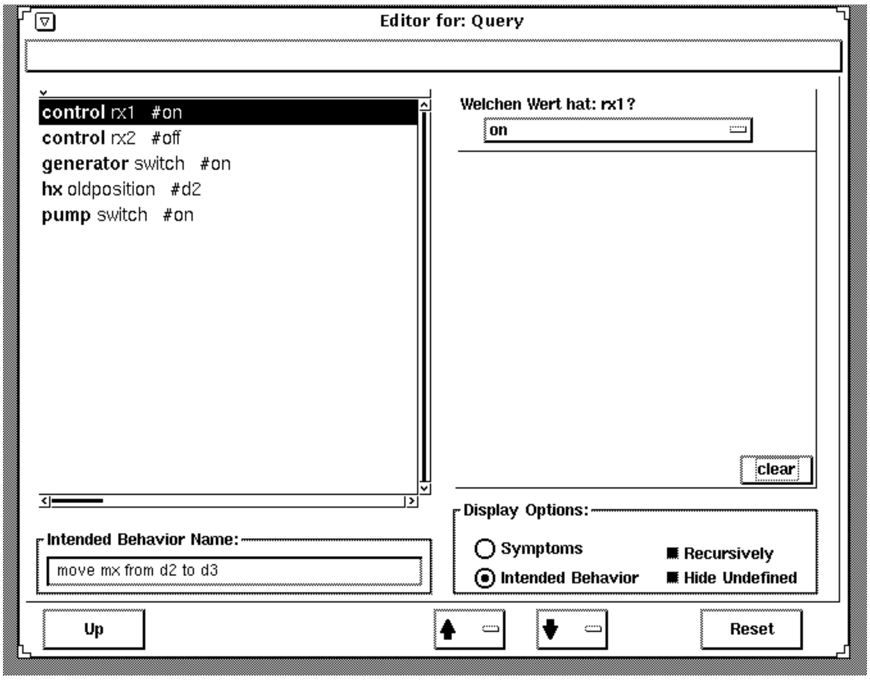
ABBILDUNG 25 Die Konsultation wird gestartet



zunächst einmal eine Anfrage eingegeben werden muß, wird nach Betätigen des "Edit Query"-Knopfs im der Editor (Abbildung 26, "Das intendierte Verhalten der Anfrage") zunächst die Ausgangslage und das gewünschte, intendierte Verhalten eingegeben.

Dieses Verhalten trägt den Namen "Move mx from d2 to d3" und beschreibt die Ausgangssignale der Steuereinheit ("control rx1 #on", "control rx2 #off"), die Tatsache, daß Pumpe und Generator eingeschaltet sind ("pump switch #on", "generator switch #on") und die Ausgangsposition der Hydraulik und damit des Greifers ("hx oldposition #d2").

ABBILDUNG 26 Das intendierte Verhalten der Anfrage



Als Symptom ist zunächst nur bekannt, daß sich der Greifer nicht bewegt hat, also wird nur die gegenwärtige Position des Greifers angegeben. An der Größe der Symptomliste kann man ungefähr erkennen, wie viele Symptome potentiell eingegeben werden könnten. Da aber keine weiteren Symptome erhoben worden sind, wird nur mit ("mx out d2") die Greiferposition eingegeben (Abbildung 27).

Nachdem diese Werte eingegeben worden sind, kann die Bearbeitung beginnen. Dabei erscheinen im Statusfenster der Konsultationsoberfläche die folgenden Meldungen:

Consultation started

Calculating derived symptoms...done.

Zunächst werden alle zusätzlichen Symptome, die sicher mit Hilfe des Domänenwissens herleitbar sind, aus den gegebenen Symptomen abgeleitet.

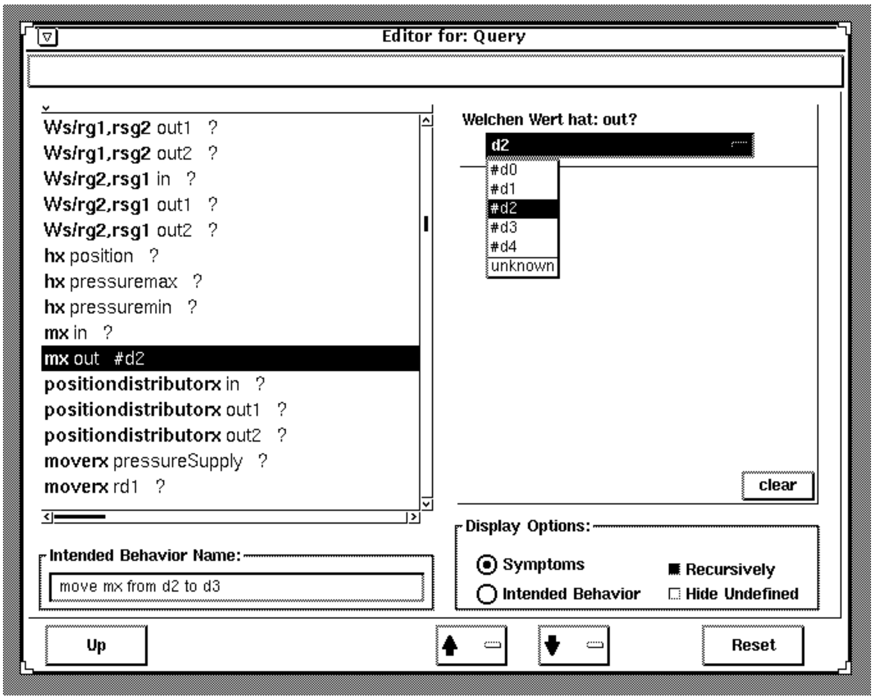
Searching for faulty behavior...done

Dann wird geprüft, ob bereits ein Defekt beschrieben worden ist, z. B. ob von einer Druckleitung angegeben wurde, daß Druck hinein- aber nicht wieder herauskommt.

Searching for unintended symptoms...done

Unintended symptoms detected in:

ABBILDUNG 27 Das Startsymptom wird eingegeben



#mx Value in query: d2 (expected: d3)

Das System stellt fest, daß die Mechanik an der falschen Stelle steht und daß dieses Verhalten nicht intendiert ist.

Searching for relevant symptoms...done

Relevant symptoms detected in:

#mx

Aus den unintendierten Symptomen werden jetzt die Symptome herausgesucht, auf die das System zunächst fokussiert. Da im Moment nur ein einziges Symptom gegeben ist, ist das natürlich das Symptom an mx

Retrieving cases...

#CASE21 rejected

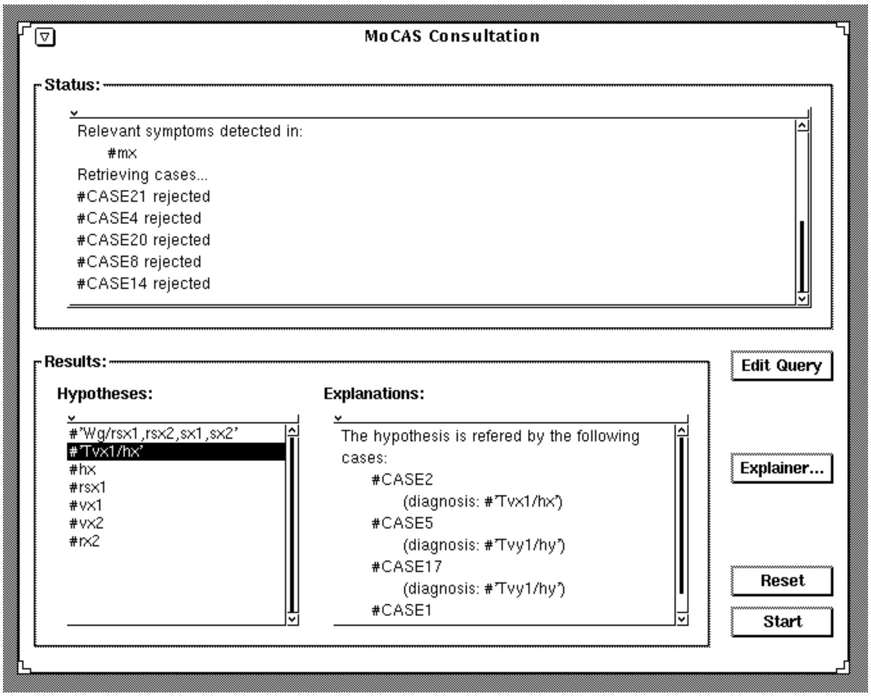
#CASE4 rejected

#CASE20 rejected

#CASE8 rejected

#CASE14 rejected

ABBILDUNG 28 Der erste Konsultationslauf



Um danach das Aussortieren unähnlicher Fälle besser verfolgen zu können, werden die Namen der Fälle, bei denen keine Übertragung auf die Anfrage möglich ist, ausgegeben.

Man sieht, daß nur wenige Fälle zurückgewiesen werden und viele verschiedene Hypothesen generiert worden sind. Der Benutzer kann sich nun zu jeder Hypothese die Fälle zeigen lassen, aus denen sie hergeleitet wurde. Weiterhin kann er sich die Fälle und deren Übertragung am Schaltplan der Maschine zeigen lassen. In Abbildung 29 wird gerade ein Fall angezeigt. Zu allen geschwärzten Bauteilen sind im Fall Symptome angegeben worden, und man erkennt, daß der Fall sich an physikalisch vollkommen anderen Bauteilen der Maschine zugetragen hat. Der Benutzer kann sich jetzt die Übertragung des Falls auf die Situation anzeigen lassen. Abbildung 30 zeigt dabei erst einmal, welcher Defekt im Fall aufgetreten war und wie er sich auf das zu mx analoge Bauteil mg ausgewirkt hatte. Man erkennt eine Kette von geschwärzten Bauteilen die vom Ursachenbauteil (rsg_1) zu mg führt. Ein gleichartiger Störfall könnte auch in unserer Situation aufgetreten sein, wie die Übertragung auf die Bauteile, deren Verhalten wir beobachtet haben, deutlich macht.

Diese Übertragung ist in Abbildung 31 gezeigt. Dabei wird dann als plausible Hypothese ein Versagen des Bauteils rsx_1 ausgemacht. (Dieses Bauteil schaltet die Steuersignale für die Vorwärtsrichtung aus, wenn die Rückwärtsrichtung aktiviert ist.) Tatsächlich könnte ein Versagen dieses Bauteils das von uns beobachtete Ver-

ABBILDUNG 29 Die Darstellung des Falls (Fall 16) am Schaltplan

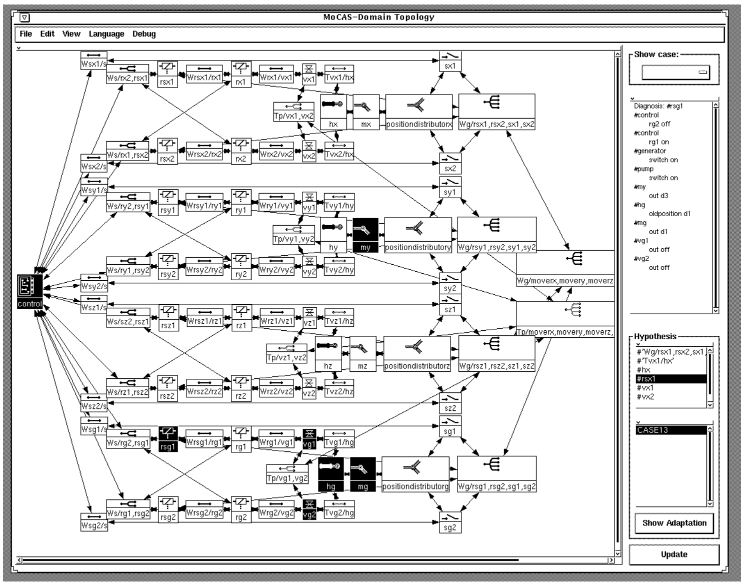
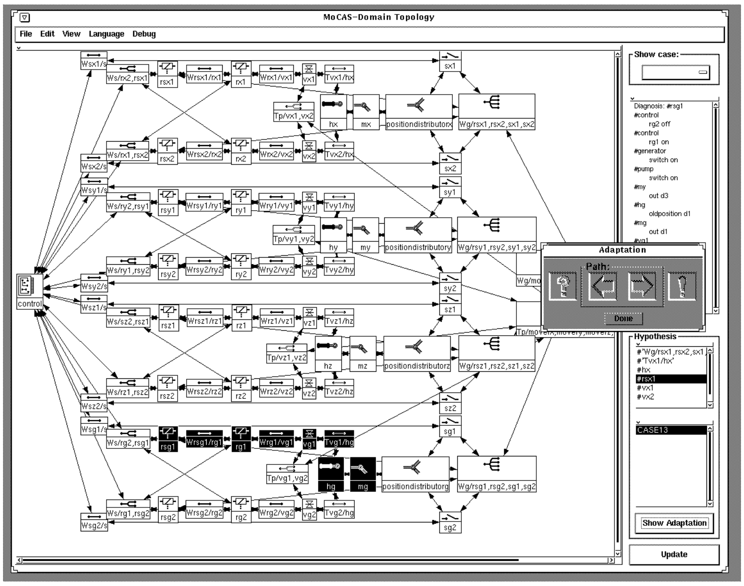
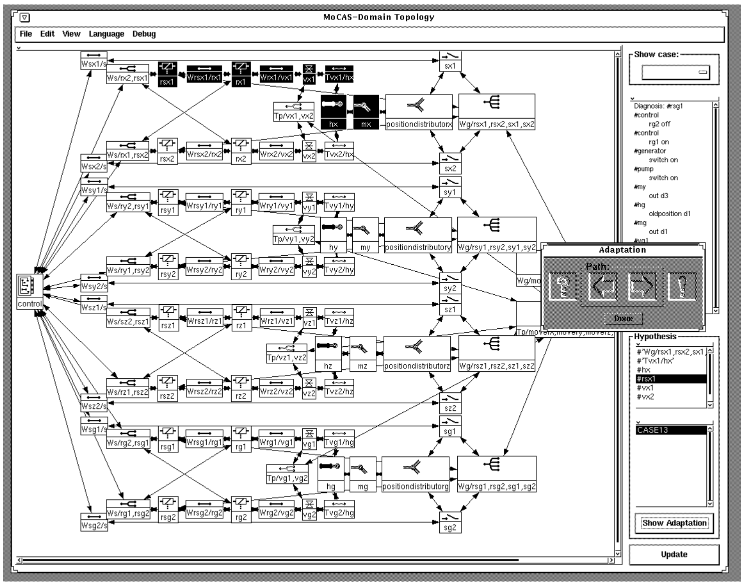


ABBILDUNG 30 Die Störung im Fall



halten des Greifers ausgelöst haben, und es gibt keine bekannten Symptome, die

ABBILDUNG 31 Die Übertragung des Falls auf die Anfrage



gegen diese Hypothese sprechen würde. Allerdings sind alle anderen vom System vorgeschlagenen Hypothesen genauso plausibel.

Daher ist es dem Benutzer jetzt freigestellt, weitere Symptome an einem Bauteil seiner Wahl zu erheben. Dabei kann er sich auf die von MoCAS/2 vorgeschlagenen Hypothesen stützen, wenn er das möchte. In unserem Fall erheben wir ein weiteres Symptom an der Druckleitung $T_{vx1/mx1}$, weil diese als Hypothese vorgeschlagen und leicht erreichbar ist. Dieses Symptom wird wieder eingegeben (Abbildung 32).

Anschließend wird wieder die Konsultation gestartet und diesemal erscheinen folgende Anzeigen im Statusfenster (Abbildung 33):

Consultation started

Calculating derived symptoms...done.

Searching for faulty behavior...done

Searching for unintended symptoms...done

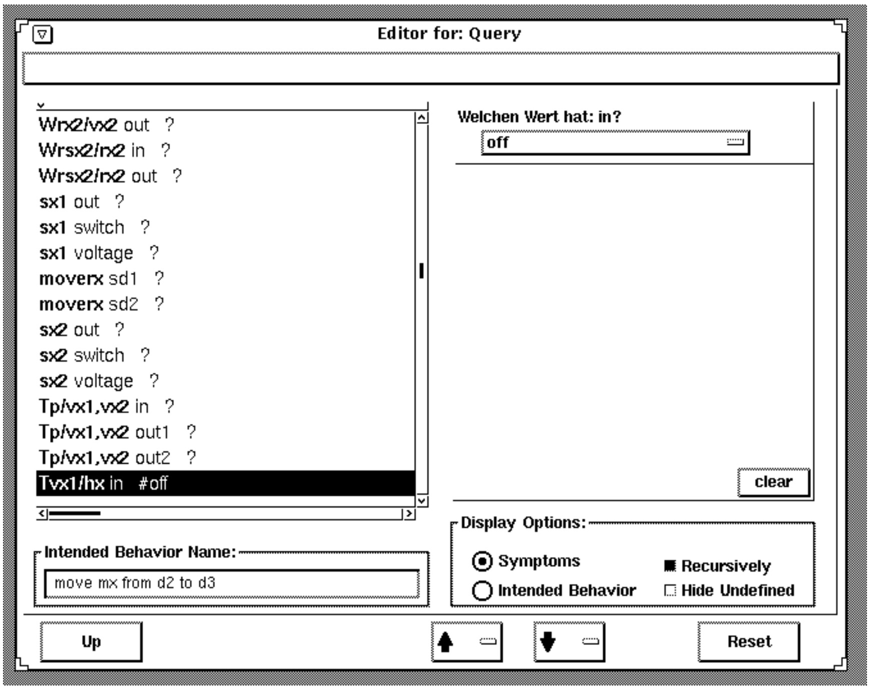
Unintended symptoms detected in:

#'Tvx1/hx' Value in query: off (expected: on)

#mx Value in query: d2 (expected: d3)

Hier wird festgestellt, daß auch das neu eingegebene Symptom ein unintendiertes Verhalten wiederspiegelt

ABBILDUNG 32 Ein zusätzliches Symptom wird eingegeben



Searching for relevant symptoms...done

Relevant symptoms detected in:

#'Tvx1/hx'

Da das unintendierte Verhalten der Mechanik mx offensichtlich daraus resultiert, daß schon aus der Druckleitung kein Druck kommt, obwohl das eigentlich geschehen sollte, kann das System hier auf die Druckleitung fokussieren. Dadurch werden dann in ersten (nicht ausgegebenen) Filterschritten schon wesentlich mehr Fälle ausgesondert. Von den restlichen Fällen werden dann noch die folgenden Fälle zurückgewiesen, da sie sich nicht auf die Situation übertragen lassen.

Retrieving cases...

#CASE4 rejected

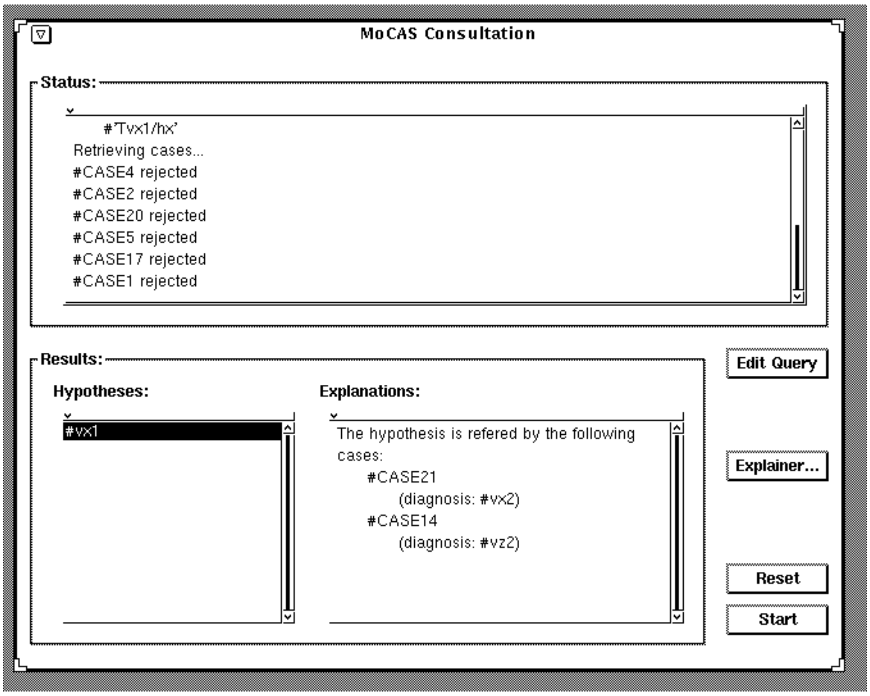
#CASE2 rejected

#CASE20 rejected

#CASE5 rejected

#CASE17 rejected

ABBILDUNG 33 Der zweite Konsultationslauf

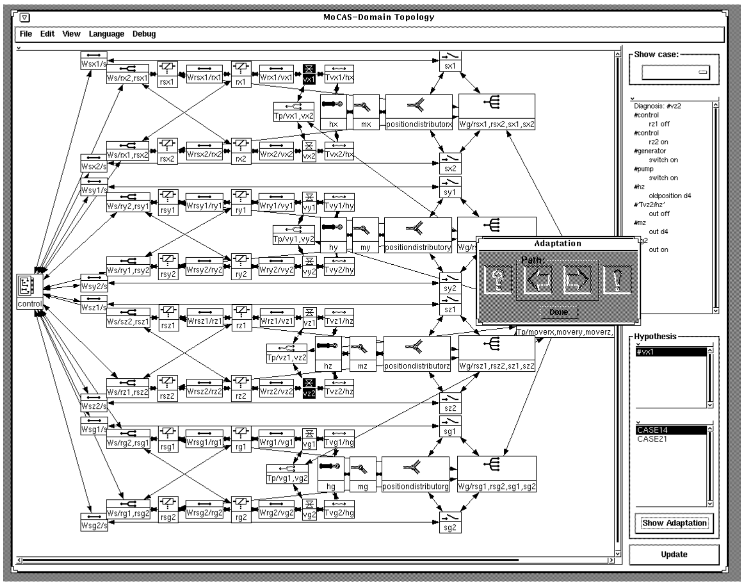


#CASE1 rejected

Jetzt sind nur noch so wenige Fälle übriggeblieben, daß nur noch eine Hypothese generiert werden kann: vx1.

Auch hierfür kann nun die Übertragung am Schaltplan dargestellt werden (Abbildung 34). Hier wurde eine Darstellung gewählt, in der die einander entsprechenden Bauteile in Situation und Fall geschwärzt worden sind. Da die nun angegebene Hypothese vx1 (hergeleitet aus der Diagnose von Fall 14, vz2) tatsächlich die zu Anfang angenommene Störursache darstellt, können wir die Konsultation hier beenden. Falls wir hier noch nicht auf eine Lösung unseres Problems gestoßen wären, hätte sich der Konsultationsvorgang noch weiter wiederholt, der Fokus wäre wieder umgesetzt worden, bis schließlich die Diagnose gefunden worden wäre.

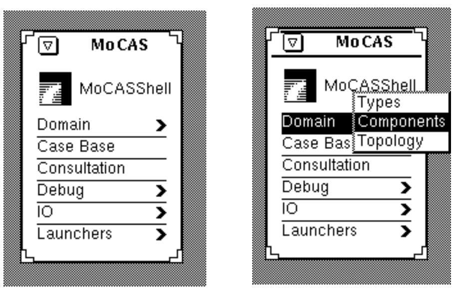
ABBILDUNG 34 Übertragung der Diagnose des neuen Falls auf die Bauteile der Ausgangssituation



ANHANG 2 Die Benutzung der Werkzeuge in MoCAS/2

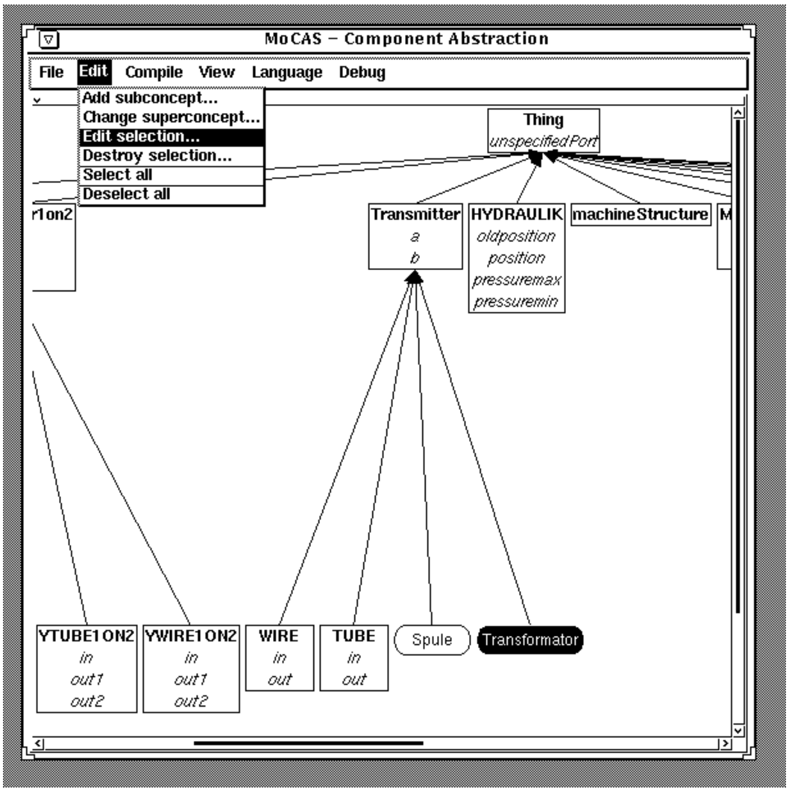
Um die Funktionsweise der Werkzeuge in MoCAS/2 zu demonstrieren, greifen wir auf das Beispiel “Transformator in der CNC-Maschine” auf Seite 26 zurück. Dort wird ein Transformator modelliert und in die Abstraktionshierarchie als Konkretisierung eines allgemeinen Übertragers eingebunden. Wir gehen in unserem Beispiel nun davon aus, daß das Konzept Übertrager und das Konzept Kabel bereits modelliert sind. Wir betätigen dazu im MoCAS-Starter (Abbildung 35) den Knopf

ABBILDUNG 35 Der MoCAS-Starter



“Domain” und wählen den Menüeintrag “Components”. Daraufhin erscheint der Editor zur Modellierung der Abstraktionshierarchie, in dem wir das Konzept “Übertrager” bzw. “Transmitter” anwählen und durch den Menüeinrtag”add Subconcept...” (unter “Edit”) die Konzepte “Spule” und “Transformator” anlegen. Diese haben zunächst ein anderes Aussehen als die übrigen Konzepte, da ja noch keine Instanzenvariablen definiert sind. Mit dem Befehl “Edit selection...” rufen wir jetzt den Konzepteditor zunächst für das Konzept “Spule” auf (Abbildung 36, “Der Editor für Bauteilhierarchien”). In diesem Editor können wir zunächst die

ABBILDUNG 36 Der Editor für Bauteilhierarchien



Namensgebung und das Icon für das Konzept verändern, wie man in Abbildung 37, “Der Editor für die Bezeichner und das Icon eines Konzepts” erkennen kann. Im “Edit”-Menü kann man nun zwischen weiteren Eigenschaften des Konzepts auswählen, die man nun noch editieren kann. Dazu zählt eine textuelle Beschreibung unter dem Eintrag “Description”, welche einen Texteditor aufruft (nicht im Bild gezeigt). Weiterhin können die Attribute für die Spule festgelegt werden (a und b), die beide vom Typ Strom sind. Das wird in Abbildung 38, “Der Editor zur Definition der Attribute” dargestellt. Die Spule soll nur eine einzige Verhaltensregel haben, nämlich die, daß aller Strom, der in sie hineinfließt, auch wieder aus ihr herausfließt. Diese Regel kann man unter dem Eintrag “Rules” im “Edit” -Menü finden. Man kann wählen, ob diese Regel in beide Richtungen gelten soll (in unserem Fall ist das so) und ob diese Regel immer gilt oder nur, falls das Bauteil nicht defekt ist. (Dazu kann man den Knopf “intended behavior only” aktivieren oder deaktivieren. Dies ist in Abbildung 39, “Beschreibung des Verhaltens der Spule durch eine Regel” dargestellt. Um für die hier definierte Regel die passende Abstraktionsregel des abstrakteren Konzepts “Transmitter” zu setzen, wird im nächsten Editor (Eintrag “Rule Abstraction”) der Editor für Regelabstraktionen gewählt. Diese Regel-

ABBILDUNG 37 Der Editor für die Bezeichner und das Icon eines Konzepts

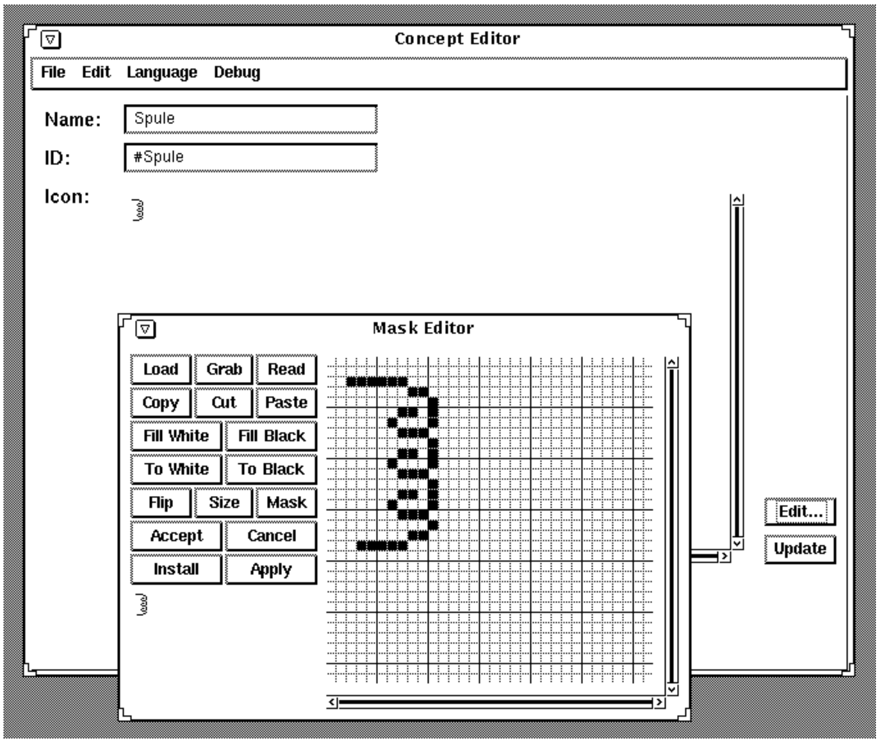


ABBILDUNG 38 Der Editor zur Definition der Attribute

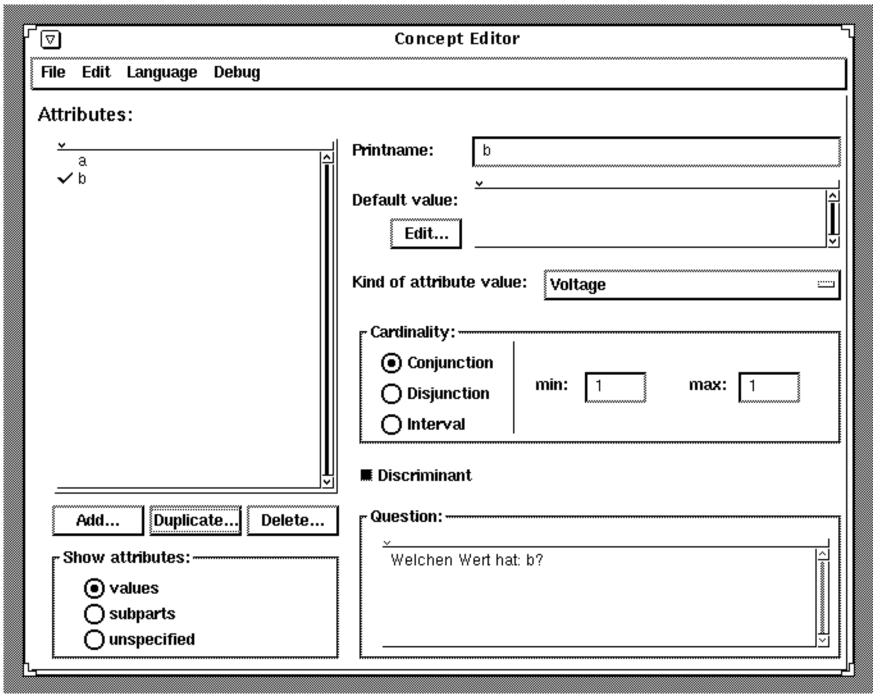


ABBILDUNG 39 Beschreibung des Verhaltens der Spule durch eine Regel

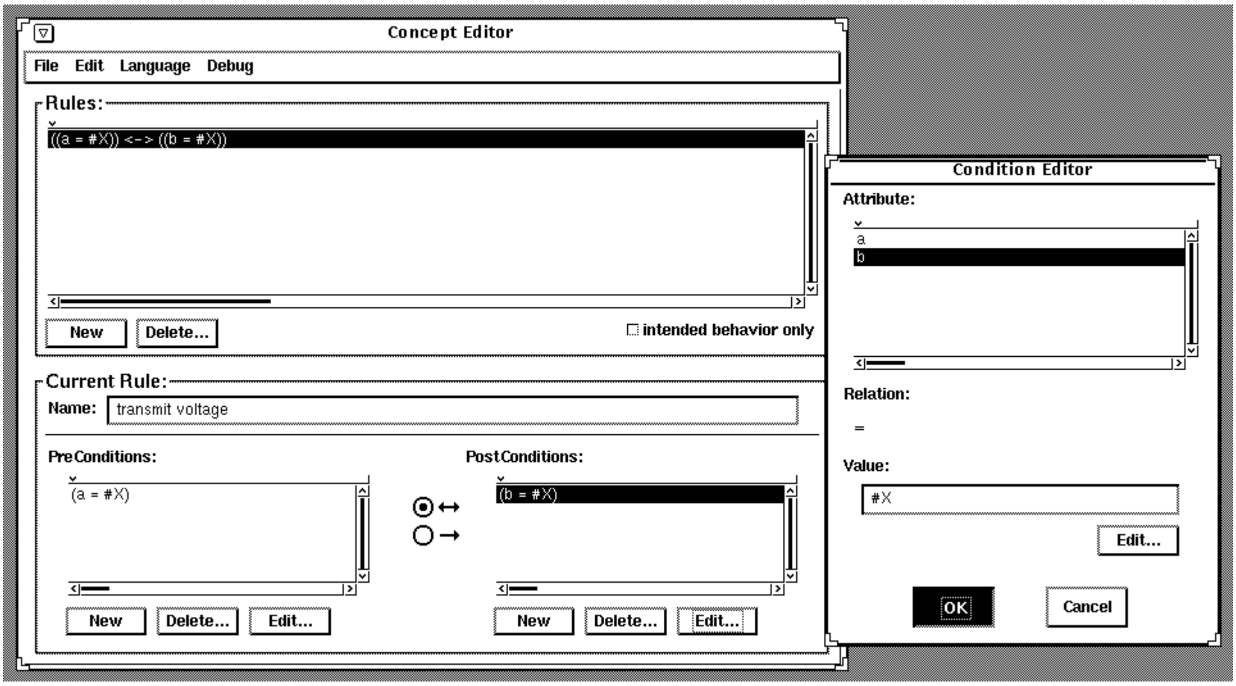
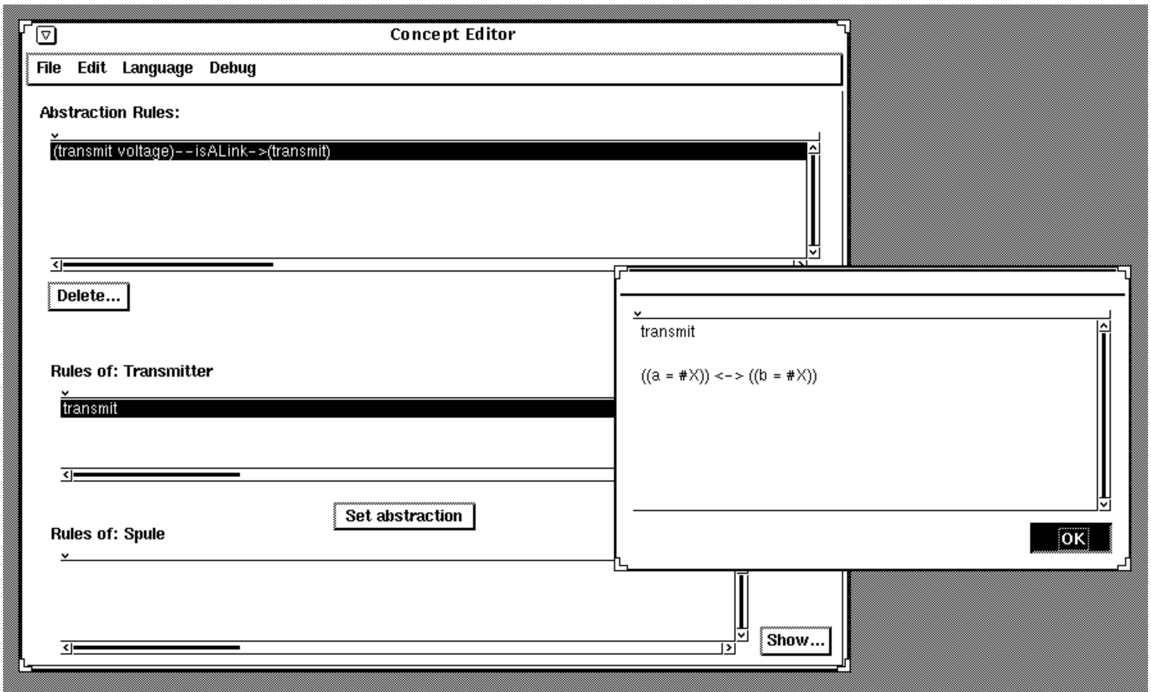


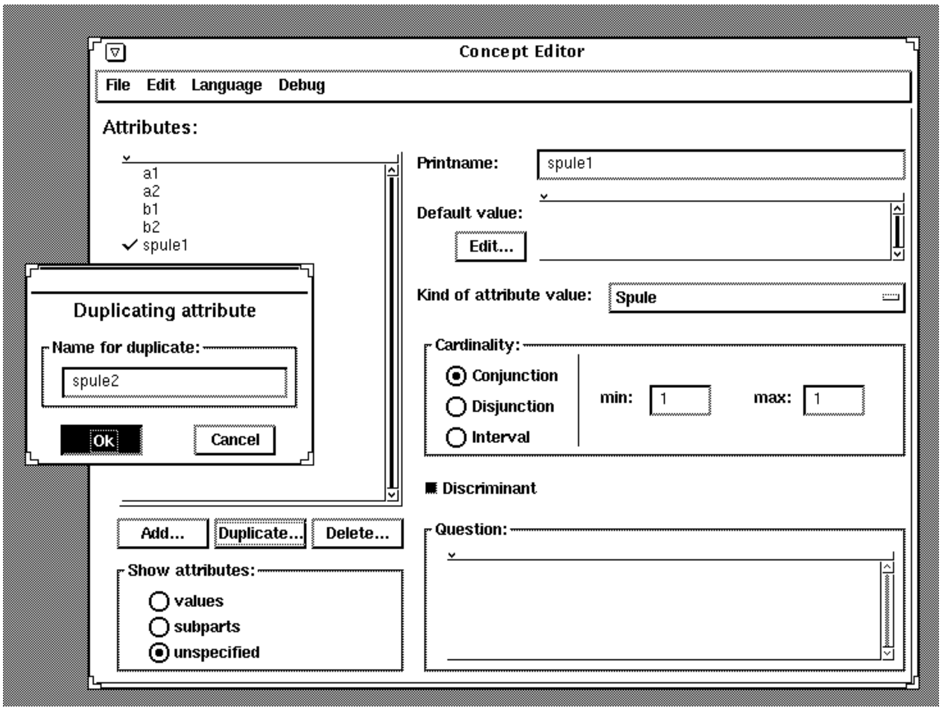
ABBILDUNG 40 Der Editor zur Abstraktion von Regeln



abstraktion wird in Abbildung 40, “Der Editor zur Abstraktion von Regeln” vorgestellt. Hier ist ein Dialogfenster geöffnet, mit dem man sich die Regel des Übertragers nochmals ansehen kann.

Wenn wir nun den Transformator modellieren, geben wir wie bei der Spule zunächst die sechs Attribute (a1, a2, b1, b2, spule1, spule2) an, von denen die ersten vier als Typ wieder “Voltage”, die letzten beiden wieder ein Unterkonzept Spule besitzen. (Abbildung 41, “Definition der Attribute für den Transformator”)

ABBILDUNG 41 Definition der Attribute für den Transformator



Danach werden die inneren Verbindungen innerhalb des Transformators festgelegt (Menüeintrag “Connections”, dargestellt in Abbildung 42, “Der Editor für die inneren Verbindungen im Transformator”). Schließlich werden auch wieder die Übertragungsregeln (analog zur Definiton der Spule) eingegeben. Als letztes werden dann noch die Attribute und ihre Werte selbst abstrahiert, wie in Abbildung 43, “Abstraktionsregel-Editor für die Abstraktion vom Transformator zum Übertrager” dargestellt.

ABBILDUNG 42 Der Editor für die inneren Verbindungen im Transformator

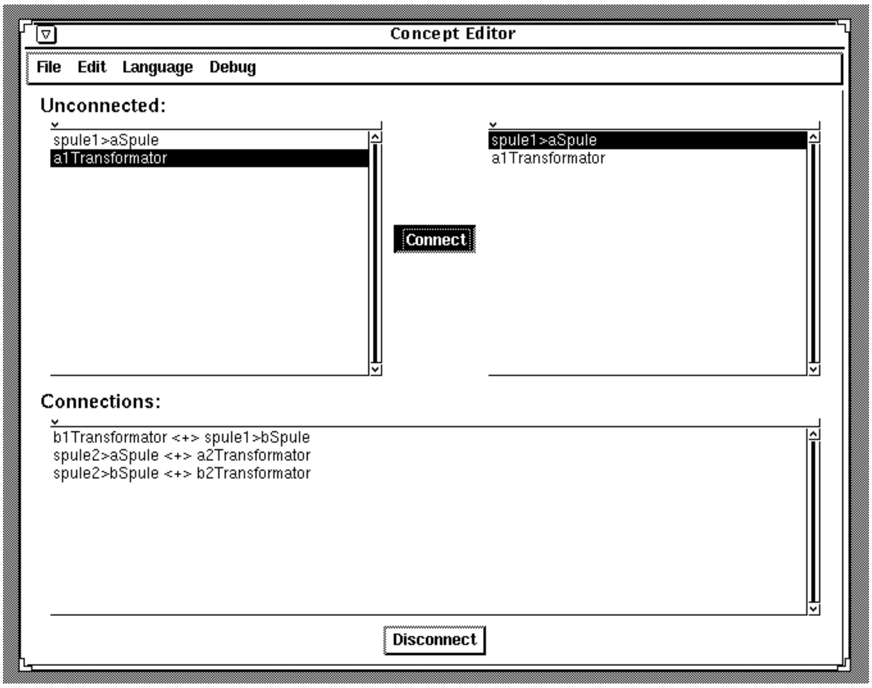
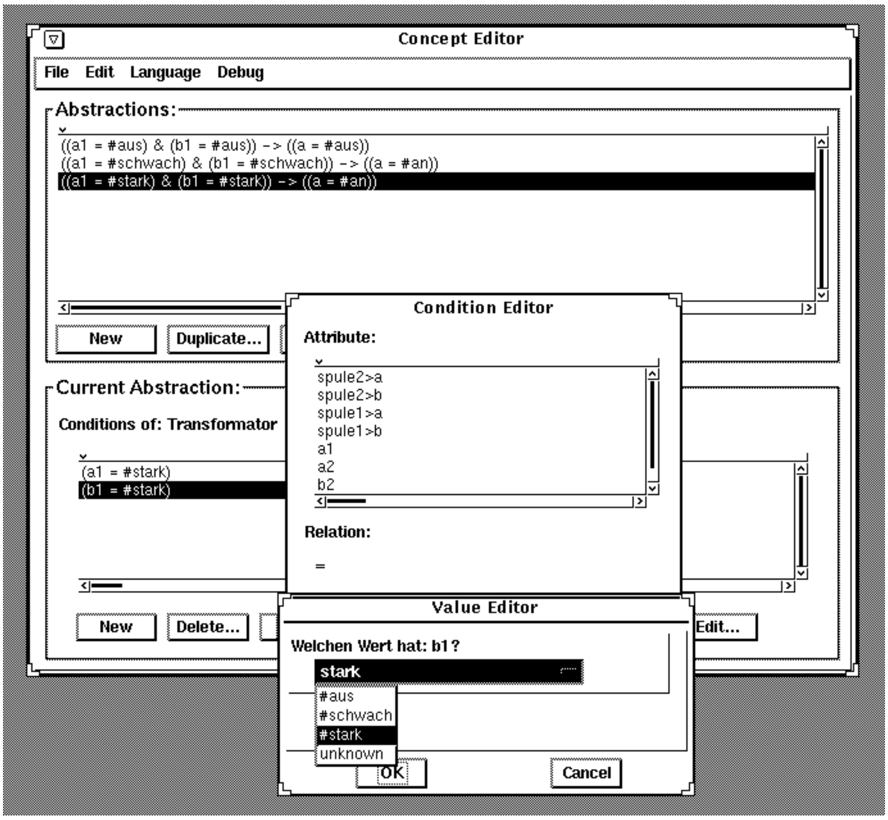


ABBILDUNG 43 Abstraktionsregel-Editor für die Abstraktion vom Transformator zum Übertrager



ANHANG 3

Literatur

[Aamodt91]
A. Aamodt: A Knowledge-Intensive, Integrated Approach to Problem Solving and Sustained Learning, PhD Thesis University of Trondheim 1991

[Aamodt93]
A. Aamodt: Tutorial Case-Based-Reasoning - An Overview, European Workshop on Case-Based-Reasoning 1993

[Althoff et al. 93]
K.-D. Althoff, R. Bergmann, F. Maurer, S. Wess and M. Manago, E. Auriol, N. Conruyt, R. Traphöner, M. Bräuer, S. Dittrich: Integrating Inductive and Case-Based Technologies for Classification and Diagnostic Reasoning, Proceedings of the ECML-93 Workshop on „Integrated Learning Architectures“

[BareissPorterWier87]
Ray Bareiss, Bruce Porter, Creg Wier: PROTOS - An Exemplar-Based Learning Apprentice, Proceedings 2nd Knowledge acquisition for knowledge based systems workshop, 1987

[Bareiss89]
Ray Bareiss: PROTOS: A unified Approach to Concept Representation, Classification and Learning, PhD Thesis, Vanderbillt University, Nashville 1988

[Bergmann92]
R. Bergmann: Knowledge acquisition by generating skeletal plans. Contemporary Knowledge Engineering and Cognition, Springer-Verlag Heidelberg 1992, pages 125-133.

[BergmannPews93]
Rasph Bergmann und Gerd Pews: Explanation-Based Similarity for Case Retrieval and Adaptation and its Application to Diagnosis and Planning Tasks

[BergmannPewsWilke94]
Ralph Bergmann, Gerd Pews, Wolfgang Wilke: A Unifying Approach for Integrating Domain Knowledge into Case-based Reasoning for Diagnosis and Planning

Tasks. Selected papers from the first European Workshop on Case-based Reasoning (EWCBR-93), Springer (in press)

[BergmannWilke94]
Ralph Bergmann und Wolfgang Wilke: Lernen von Abstraktionshierarchien zur Optimierung der Auswahl von maschinell abstrahierten Plänen. 8. Workshop „Planen und Konfigurieren“ (PuK-94), Universitaet Kaiserslautern 1994

[Beste94]
Enrik Beste: Automatisches Generieren von Eingabemasken unter CoMo-Kit, Projektarbeit, Universität Kaiserslautern.

[Carbonell86]
J. G. Carbonell: Derivational Analogy: A Theory of Reconstructive Problem Solving and Expertise Acquisition, Machine Learning, Vol. 2, 1986

[Feigenbaum80]
E. Feigenbaum: Knowledge Engineering - The Applied Side of Artificial Intelligence. Stanford Heuristic Programming Project Report, Stanford University, 1980

[Goldberg83]
Adele Goldberg: SMALLTALK-80 The interactive Programming Environment. Addison-Wesley, 1983

[GoldbergRobson83]
A. Goldberg and D. Robson: SMALLTALK-80 The Language and its Implementation. Addison-Wesley, 1983

[Kolodner93]
Janet L. Kolodner: Case-Based Reasoning, Morgan Kaufmann, 1993

[ManagoAlthoff et al. 93]
M. Manago, K. D. Althoff, E. Auriol, R. Traphöner, S. Wess, N. Conruyt, F. Maurer: Induction and Reasoning from Cases, Proceedings EWCBR-93

[Maurer93]
Frank Maurer: Hypermediabasiertes Knowledge Engineering für verteilte wissensbasierte Systeme, infix 1993

[PewsWeilerWess92]
Pews, G. and Weiler, F. and Wess, S.: Bestimmung der Ähnlichkeit in der fallbasierten Diagnose mit simulationsfähigen Maschinenmodellen, Workshop: Ähnlichkeit von Fällen beim fallbasierten Schließen (SEKI WORKING PAPER SWP-92-11), pages 101-106, University of Kaiserslautern 1992

[PewsWeiler92]
G. Pews und F. Weiler: MoCAS - Ein Ansatz zur falladaptiven Diagnostik, Projektarbeit, Universität Kaiserslautern, Fachbereich Informatik 1992

[Pews Wess93]
Gerd Pews and Stefan Wess: Integration fallbasierter und modellbasierter Ansätze zur Diagnose technischer Systeme, Proceedings CBR-Workshop Freiburg 1993, pages 41-49

[PewsWess93]
Gerd Pews und Stefan Wess: Combining Case-Based and Model-Based Approaches for Diagnostic Applications in technical Domains, Proceedings EWCBR-93

[PfeiferRichter93]
Pfeifer, Richter (Hrsg.): Diagnose von technischen Systemen, Deutscher Universitäts Verlag Wiesbaden 1993

[PorterBareiss89]
Bruce Porter and Ray Bareiss: Knowledge Acquisition and Heuristic Classification in Weak-Theory Domains, Artificial Intelligence 45, 1989

[Rehbold91]
R. Rehbold: Integration modellbasierten Wissens in technische Diagnostik-Expertensysteme, PhD Thesis, Fachbereich Informatik, Universität Kaiserslautern, 1991

[Richter89]
Michael M. Richter: Prinzipien der künstlichen Intelligenz. Teubner, Stuttgart 1989

[Schuch92]
Andreas Schuch: iMAKE - Inkrementelle Modellierung und Simulation technischer Geräte zur Generierung einer Wissensbasis für MOLTKE3.0. Diplomarbeit, Universität Kaiserslautern, Fachbereich Informatik, 1992

[Schweiger91]
Ulrich Schweiger: Eine erklärungsorientierte Methode zur Integration fall- und modellbasierten Schließens, Dimplomarbeit, Universität Karlsruhe, 1991

[Tversky77]
A. Tversky: Features of Similarity. Psychological Review 84, S.327-352

[VelosoCarbonell93]
Towards scaling up machine learning: A case study with derivational analogy in PRODIGY. Machine Learning Methods for Planning, pages 233-272, Morgan Kaufmann 1993

[Wess91]
Stefan Wess: PATDEX/2: Ein System zum adaptiven, fallfokussierenden Lernen in technischen Diagnosesituationen

[Wess93]
Stefan Wess: PATDEX - Inkrementelle und wissensbasierte Verbesserung von Ähnlichkeitsurteilen in der fallbasierten Diagnostik

[Wilke94]
Wolfgang Wilke: Entwurf, Implementierung und experimentelle Bewertung von Auswahlverfahren für abstrakte Pläne in dem fallbasierten Planungssystem PARIS, Diplomarbeit, Universität Kaiserslautern, 1994