Diplomarbeit

Erklärungsbasiertes Lernen von Kontrollinformationen aus Fehlschlägen bei der Planung

Thomas Roth-Berghofer

Juli 1996

Betreuung: Prof. Dr. Michael M. Richter
Dipl. Inform. Frank Weberskirch

Arbeitsgruppe Künstliche Intelligenz
— Expertensysteme —
Prof. Dr. Michael M. Richter

Universität Kaiserslautern
Fachbereich Informatik

Inhaltsverzeichnis

Kapitel 1

Einleitung

Das Lernen ist für den Menschen ein wichtiger Teil des Entwicklungsprozesses und erlaubt ihm, aus positiven und negativen Erfahrungen Konsequenzen für sein weiteres Verhalten abzuleiten, insbesondere für seine Entscheidungsfindung. Diese Art des Lernens, das Lernen aus Erfahrung, kann jedoch nur stattfinden, wenn diese Erfahrungen erklärt werden können. Auf diesem Ansatz aufbauend, werden seit einigen Jahren Verfahren zur Übertragung erklärungsbasierter Lernprozesse auf Computersysteme untersucht.

1.1 Lernen in der Planung

Erklärungsbasierte Lernverfahren [Winston *et al.*, 1983; DeJong, 1981; DeJong & Mooney, 1986; Mitchell *et al.*, 1986] stammen aus dem Bereich der Klassifikation und des Konzeptlernens. Im Gegensatz zu induktiven Lernverfahren, die Regularitäten in großen Beispielmengen erkennen, sind beim *erklärungsbasierten Lernen* (EBL) sowohl Zielkonzept als auch Hintergrundtheorie explizit vorgegeben und vollständig bekannt. Anhand einzelner Trainingsbeispiele werden Konzeptbeschreibungen gesucht, die ein effizientes Wiedererkennen von Zielkonzeptinstanzen gewährleisten. [Keller, 1988] betrachtete verschiedene Systeme in Bezug auf gelernte Konzepte und die effiziente Verwendbarkeit des erzeugten Kontrollwissens. Der Sinn dieser Systeme liegt nun nicht darin, mehr über das Zielkonzept zu lernen, sondern darin, die Zielkonzeptbeschreibung umzuformulieren, sie zu operationalisieren. Ein entsprechendes Operationalisierungskriterium muß dazu im Raum synonymer Konzeptbeschreibungen diejenigen bevorzugen, die effizienter zu testen sind.

Erklärungsbasiertes Lernen ist ein Teilbereich deduktiven Lernens, wobei ein deduktiver Lernschritt ein Deduktionsschritt im klassischen (monotonen) Sinne ist. Damit liegt ein wichtiger Vorteil gegenüber induktiven (meist nichtmonotonen) Lernverfahren, daß das durch erklärungsbasierte Methoden Erlernte stets richtig ist und nicht revidiert werden muß.[1]

[Minton, 1988] übertrug erstmals erklärungsbasierte Methoden auf den Bereich des Planens. Mit PRODIGY/EBL, einem total-ordnenden Planungssystem, entwickelte Minton eine Vielzahl von Zielkonzepten, die durch die Hintergrundtheorie des total-ordnenden Planens vollständig erklärt werden. Die damit erzeugten Regeln erlauben PRODIGY/EBL eine sinnvolle Einschränkung des durchsuchten Raums der Weltzustände.

Im Gegensatz zu total-ordnenden Planern operieren partiell-ordnende Verfahren auf Plan-

[1] Zu verschiedenen Lernverfahren siehe z. B. [Richter, 1992].

zuständen, wodurch die Zielkonzepte von [Minton, 1988] nicht direkt anwendbar sind. Kambhampati entwickelte mit SNLP+EBL und UCPOP+EBL [Kambhampati *et al.*, 1995a] ein EBL-Verfahren für partiell-ordnende Planer wie das hier verwendete CAPlan [Weberskirch, 1994; Weberskirch, 1995]. Er konzentrierte sich auf das *Lernen aus Fehlschlägen*, da dieses Zielkonzept aus einer Nützlichkeitsanalyse (vgl. *utility problem*, [Minton, 1988]) als erfolgreichstes Konzept hervorging.

Was kann man nun aus Fehlschlägen lernen? Ein Fehlschlag, insbesondere in der Aktionsplanung, ist durch das Vorliegen bestimmter Bedingungen gekennzeichnet, die nicht gemeinsam erfüllt werden können (siehe Abschnitt 4.2). Bei einem Fehlschlag liegt eine unmittelbare Inkonsistenz im Plan vor, die aber nicht mehr verschwindet, da Pläne bezüglich ihrer Constraints im Lauf des Planungsprozesses monoton wachsen. Diese inkonsistenten Bedingungen müssen identifiziert werden, um einen erneuten Fehlschlag zu vermeiden, wenn die gleiche Situation wieder eintritt. Die Identifikation der Fehlschlagsbedingungen bedeutet eine Suche nach einer *Erklärung* auf die Frage: *Warum* schlug eine Aktion fehl? Dabei darf die Erklärung nicht im umgangssprachlichen Sinne verstanden werden, sondern ist hier ein *Beweis*, der in der verwendeten Hintergrundtheorie geführt wird. In dieser Arbeit ist diese Theorie das Verfahren der partiell-ordnenden Planung, anhand derer Fehlschläge erklärt und daraus Regeln zur Vermeidung dieser Fehlschläge abgeleitet werden.

1.2 Zielsetzung der Arbeit

Das Ziel der Arbeit war die Erweiterung des Planungssystems CAPlan um die Fähigkeit des Lernens aus Fehlschlägen. Da CAPlan modular aufgebaut ist, konnten Kontrollkomponenten zur Regelerzeugung und Regelabarbeitung realisiert werden, die die bestehenden Module ergänzen.

Die Ziele waren im einzelnen:

- Entwicklung einer Komponente für das Lernen von Zurückweisungsregeln zur Steuerung der Operatorauswahl, basierend auf SNLP+EBL von [Kambhampati *et al.*, 1995a].
- Entwicklung eines Konzeptes für die Verarbeitung gelernter Zurückweisungsregeln.
- Anbindung eines Prolog-Regelinterpreters an das Planungssystem CAPlan zur schnellen Regelabarbeitung.
- Realisierung von Prädikaten, die Prolog als Regelauswertungskomponente Auskunft über den aktuellen Planzustand (z. B. über die Existenz eines Causal Links oder eine bestimmte Ordnung von Planschritten), und über die Problemstellung geben (z. B. welche Aussagen in der Start-Situation gelten).
- Erste Auswertungen über Nutzen der gelernten Regeln und Effektivität der Regelanwendung.

Der partiell-ordnende Planer SNLP+EBL von [Kambhampati *et al.*, 1995a] operiert auf einem Suchbaum, der von Plänen aufgespannt wird. Zum Zwecke der Planungsentscheidungen und den Erwerb von Kontrollwissen kann SNLP+EBL die Pläne beliebig betrachten. Im Gegensatz dazu speichert CAPlan nicht die Pläne, sondern Informationen über erfolgreiche und fehlgeschlagene Entscheidungen, sowie Abhängigkeiten zwischen ihnen mit Hilfe der Abhängigkeitsverwaltung Redux. Eine entsprechende Anpassung mußte vorgenommen werden.

1.3 Übersicht über die Arbeit

Die vorliegende Arbeit gliedert sich wie folgt:

- Kapitel 2 stellt den SNLP-Ansatz und seine Ausprägung im System CAPlan vor. Anhand des SNLP-Grundalgorithmus werden die Stellen im Verfahren verdeutlicht, an denen erklärungsbasiertes Lernen ansetzt. Auch ein Überblick über die Abhängigkeitsverwaltung Redux wird gegeben.

- In Kapitel 3 wird erklärungsbasiertes Lernen in seinen Grundzügen dargelegt. Ein Beispiel führt in die Thematik der erklärungsbasierten Generalisierung ein, aus der sich dann erklärungsbasiertes Lernen entwickelte. Darauf aufbauend wird die nötige Terminologie erläutert, bevor das Lernen aus Fehlschlägen im System Prodigy/EBL betrachtet wird.

- Kapitel 4 bildet das Kernstück dieser Arbeit. Detailliert wird das Lernen aus Fehlschlägen entwickelt. Von der Generierung initialer Erklärungen und ihre Regression wird ein Bogen zur Erzeugung der Kontrollregeln und ihre Generalisierung gespannt. Auf Probleme und Aspekte, die bei der Realisierung in CAPlan/EBL von Bedeutung waren, wird dabei an den entsprechenden Stellen hingewiesen.

- Kapitel 5 widmet sich der Realisierung von erklärungsbasiertem Lernen in CAPlan/EBL. Im Mittelpunkt steht die Erweiterung um eine Kontrollkomponente, die das Lernen in den Backtrackingmechanismus integriert, sowie eine Kontrollkomponente, die die Regelverarbeitung in den Konsistenztest von Operatoren einbezieht. Die Benutzerschnittstelle zur Regelbearbeitung und Verwaltung der Regelbasen wird ebenfalls betrachtet. Erste Tests bezüglich der Nützlichkeit der gelernten Zurückweisungsregeln schließen sich an.

- Kapitel 6 faßt die Ergebnisse zusammen und gewährt einen Ausblick auf geplante Erweiterungen

Die vorgestellte Erweiterung von CAPlan wurde, wie das Gesamtsystem selbst, in Smalltalk-80 in der Version VisualWorks 2.0, auf den Rechnern der AG Richter realisiert. Das Verständnis der Implementierung setzt daher gründliche Kenntnisse dieser Sprache und des objektorientierten Ansatzes voraus. Empfohlene Literatur dafür ist z.B. [Goldberg & Robinson, 1983]. Ebenso wichtig sind Kenntnisse im Bereich der Justification Based Truth Maintenance Systeme [Doyle, 1979], da ein solches System zur Abhängigkeitsverwaltung im Systemkern von CAPlan verwendet wird.

Das System CAPlan/EBL basiert, wie bereits erwähnt, auf [Weberskirch, 1994; Weberskirch, 1995]. Dort wird der SNLP-Ansatz ausführlich erläutert. Auch die Verknüpfung des SNLP-Verfahrens mit der Abhängigkeitsverwaltung, die eine interaktive Planung erst erlaubt, ist dort detailliert zu finden.

Als Interpreter zur Auswertung der Kontrollregeln fand SWI-Prolog, Version 1.8, Anwendung [Wielemaker, 1993]. Grundlegende Kenntnisse in Prolog sind für diesen Teil der

Arbeit vonnöten. Eine gute Einführung in die Prolog-Programmierung für die künstliche Intelligenz bietet z.B. [Bratko, 1990].

Das Verbindungsstück zwischen Smalltalk und Prolog bildet die Kommunikationsschnittstelle von [Niehaus & Lorscheid, 1995], deren Implementierung diese Arbeit erst ermöglichte. Die Kommunikationsschnittstelle wurde bereits erfolgreich im Begriffshierarchie-orientierten Konfigurationssystem IDAX der AG Richter [Schirp, 1995] eingesetzt und dient dort zur Auswertung von Präferenzregeln.

Kapitel 2

Der Planungsassistent CAPLAN

CAPLAN [Weberskirch, 1994; Weberskirch, 1995] basiert auf dem Algorithmus zur *systematischen, nichtlinearen Planung (*SNLP*)* von [McAllester & Rosenblitt, 1991], einem domänenunabhängigen Planungsverfahren mit partiell geordneter Planrepräsentation. Die Realisierung orientierte sich an der Version von [Barrett & Weld, 1992].

Die Verwaltung von Entscheidungsabhängigkeiten in CAPLAN geschieht durch REDUX [Petrie, 1991], einer Erweiterung eines *Justification-based Truth Maintenance Systems* [Doyle, 1979], das dem Planungsassistenten eine flexible Einbindung des Benutzers in den Planungsprozeß ermöglicht. Desweiteren wurde der Grundalgorithmus um Konzepte erweitert, die eine vereinfachte Domänenmodellierung erlauben und die Effektivität erhöhen [Weberskirch, 1994].

In den folgenden Abschnitten wird das Planungsverfahren, die Abhängigkeitsverwaltung REDUX und die CAPLAN-Systemarchitektur vorgestellt, soweit sie im Rahmen dieser Arbeit von Interesse sind.

2.1 Der SNLP-Ansatz

Der SNLP-Algorithmus operiert auf partiell geordneten Plänen und eröffnet dadurch einige Freiheitsgrade bei deren Modifikation. Zudem wird durch diese Planrepräsentation der Einsatz mächtiger Backtrackingverfahren erleichtert [Weberskirch, 1994; Weberskirch, 1995]. Zuvor soll jedoch definiert werden, wie ein Planungsproblem aussieht, das eine Domänenbeschreibung mit einschließt.

2.1.1 Planungsproblem und Domänendefinition

SNLP ist ein domänenunabhängiges Planungsverfahren. Aus diesem Grund muß die Domänenbeschreibung Teil der Problemstellung sein. Durch die Verwendung einer Form der *Closed World Assumption* (siehe z. B. [Richter, 1992]) genügt es, mit Hilfe von Prädikaten, die in der Domänenbeschreibung deklariert wurden, den Startzustand zu beschreiben. Dabei sind *Zustände der Planungswelt* für STRIPS [Fikes & Nilsson, 1971], einen zustandsbasierten Planer, Mengen prädikatenlogischer Formeln, d. h. im einfachsten Fall funktionsfreie Atomformeln.

Ein STRIPS-*Planungsproblem* ist nun definiert als ein Tripel (DOM,Σ,Ω):

- DOM: Domänendefinition, bestehend aus einer Menge von Prädikatssymbolen für die Definition von Zuständen der Planungswelt, sowie einer Menge von Operatorschemata, die in der Planungswelt erlaubte Aktionen beschreiben.
- Σ: Menge von Formeln zur Beschreibung des Startzustandes. Mit dieser Menge wird ein künstlicher Planschritt namens $START$ assoziiert, der die entsprechenden Formeln als Effekte zur Verfügung stellt.
- Ω: Menge von Formeln zur Beschreibung des Zielzustandes. Hiermit wird der künstliche Planschritt $FINISH$ identifiziert, dessen Vorbedingungen die zu erreichenden Ziele sind.

Die *Lösung* eines Planungsproblems besteht dann aus einer Folge von Operatoren aus DOM, so daß die sequentielle Anwendung dieser Operatoren die initiale Situation Σ in eine Situation überführt, in der die Zielprädikate aus Ω gelten.

2.1.2 Planrepräsentation

Bevor wir zu einer genauen Definition eines Planes kommen, sollen seine Bestandteile und seine Rahmenbedingungen vorgestellt werden.

Ein *Plan* besteht aus einer Menge von Planschritten und einer darauf definierten (partiellen) Ordnung. Diese Ordnung auf den Planschritten ist durch eine Menge von geordneten Paaren $s \prec t$, also Schritt s vor Schritt t, bestimmt. Eine transitive Erweiterung der Ordnung $\prec$ sei durch $\prec_{trans}$ bezeichnet und es gilt: $(s \prec t) \vdash (s \prec_{trans} t)$ und $((s \prec u) \wedge (u \prec_{trans} t)) \vdash (s \prec_{trans} t)$.

Jedem Planschritt ist eindeutig ein STRIPS-*Operator* zugeordnet. STRIPS-Operatoren sind definiert durch eine Menge von *Vorbedingungen*, die erfüllt sein müssen, um einen Operator ausführbar zu machen, sowie eine Menge von *Effekten*, die angeben, welche Prädikate nach der Ausführung gelten und welche nicht mehr gültig sind. Zur Definition der Operatorschemata sind nur funktionsfreie Atomformeln zugelassen. Quantifizierungen sind nicht erlaubt.

Da die Operatordefinitionen Variablen zulassen, gehören zu einem Plan auch Variablenbindungs-Constraints. Sie erzwingen für zwei Variablen x und y dieselbe Bindung $(Same(x, y))$ oder untersagen eine Bindung an den gleichen Wert $(NotSame(x, y))$.

Der Grund für die Einführung eines Schrittes wird beim SNLP-Ansatz über sogenannte *Causal Links*[1] buchhalterisch erfaßt:

Definition: Sei s ein Planschritt mit dem Effekt p und $t \neq s$ ein Planschritt mit der Vorbedingung p, die durch den Effekt von s erfüllt wird. Dies wird *Causal Link* genannt und durch $s \xrightarrow{p} t$ notiert.

Zusätzlich zu jedem Causal Link $s \xrightarrow{p} t$ wird in Plan auch eine Ordnung $s \prec t$ zwischen den Schritten aufgenommen.

[1]Causal Links sind eine Spezialität des SNLP-Ansatzes. Die Bestimmung der Gültigkeit eines Prädikates in einem Plan kann auch errechnet werden, wie das System TWEAK [Chapman, 1987] zeigt. Ein modales Wahrheitskriterium (modal truth criterion, MTC) bestimmt dort die notwendige oder mögliche Wahrheit eines betrachteten Prädikates. Auch für CAPLAN steht das MTC in einer Implementierung zur Verfügung [Roth-Berghofer, 1995]. Damit ist es möglich, an einer beliebigen Stelle während eines Planungsvorgangs Ein-/Ausgabesituationen von Planschritten zu berechnen, konkreter, die notwendigerweise und die möglicherweise geltenden Aussagen.

Die Gültigkeit einer Vorbedingung zur Ausführungszeit kann durch die Einführung eines Causal Links alleine noch nicht garantiert werden. Es kann zwischen einem Causal Link und einem parallel dazu liegenden Schritt zu *Interaktionen* kommen, die bei SNLP *Threat* (Bedrohung) genannt werden.

Definition: Seien p und q miteinander unifizierbare Prädikate, s, t, v paarweise verschiedene Planschritte und v bezüglich s und t nicht angeordnet. Dann ist v ein *positiver Threat (negativer Threat)* zu einem Causal Link $s \xrightarrow{p} t$, wenn q ($\neg q$) ein Effekt von v ist.

Während ein *negativer Threat* bei Nichtbeachtung zu inkorrekten Plänen führt, hat das Ignorieren von *positiven Threats* „nur" die Zerstörung der Systematik des Suchprozesses zur Folge. Da die Systematik der Suche Grundbestandteil des SNLP-Ansatzes ist, werden beide Arten von Threats berücksichtigt.[2]

Für die *Auflösung der Interaktionen* stehen grundsätzlich drei Möglichkeiten zur Verfügung (vgl. [Chapman, 1987]). Zum einen kann der bedrohende Schritt durch Einführung einer neuen Ordnung, einer *Protection*, vor den einen Schritt des Causal Link geordnet werden ($v \prec s$, *Demotion*) oder hinter den anderen ($t \prec v$, *Promotion*). Zum anderen können die Prädikate p und q, wenn sie noch ungebundene Variablen enthalten, durch Einführung von Bindungs-Constraints ($Same/NotSame$) verschieden gemacht werden, ohne die Schritte dabei anzuordnen (*Separation*).

Wir haben nun alle Komponenten eines Planes vorgestellt und können jetzt eine genaue Definition angeben.

Definition: Sei $\mathcal{P} =< \mathcal{S}, \mathcal{O}, \mathcal{B}, \mathcal{L} >$, mit

- $\mathcal{S}$: Menge von Planschritten
- $\mathcal{O}$: Menge von Ordnungsbeziehungen zwischen den Planschritten
- $\mathcal{B}$: Menge von Variablenbindungs-Constraints über den freien Variablen in $\mathcal{S}$
- $\mathcal{L}$: Menge der Causal Links im Plan, mit $\forall s \xrightarrow{p} t \in \mathcal{L} : \exists s \prec t \in \mathcal{O}$.

Dann ist $\mathcal{P}$ ein *Plan*.

Ziel des Planungsansatzes SNLP ist das Finden eines vollständigen und konsistenten Plans. Für einen *vollständigen* Plan muß gelten, daß die Vorbedingungen aller vorkommenden Planschritte zur Ausführungszeit notwendigerweise erfüllt sind. Das wird dadurch erreicht, daß für jede Vorbedingung p ein Causal Link $s \xrightarrow{p} t$ existiert und dieser Causal Link gegen alle Threats geschützt ist. Weiter muß gelten, daß der Plan *konsistent* ist.

Definition: Ein Plan $\mathcal{P}$ heißt *konsistent*, wenn folgende Bedingungen erfüllt sind:

- $\mathcal{P}$ ist *ordnungskonsistent*: Jeder Planschritt ist hinter $START$ und vor $FINISH$ angeordnet, und es existieren keine Zyklen, d. h. es gibt keine Schritte $s_1, s_2 \in \mathcal{S}$ für die gilt: $s_1 \prec_{trans} s_2 \in \mathcal{O}$ und $s_2 \prec s_1 \in \mathcal{O}$.
- $\mathcal{P}$ ist *bindungskonsistent*: Für jede Variable existiert mindestens eine Bindungsmöglichkeit, die konsistent mit allen Variablenbindungs-Constraints ist.

Auf diesen Definitionen aufbauend wird im folgenden der Planungsprozeß vorgestellt.

[2] Untersuchungen zeigten, daß die strikte Beachtung der Systematik nicht unbedingt ein effizientes Planungsverfahren zur Folge hat [Kambhampati, 1993; Kambhampati *et al.*, 1995b; Knoblock & Yang, 1994]

2.1.3 Planungsprozeß

Abbildung 2.1 zeigt eine einfache Version des SNLP-Algorithmus von [Barrett & Weld, 1992]. Dabei sei der Plan $\mathcal{P} = (\mathcal{S}, \mathcal{O}, \mathcal{B}, \mathcal{L})$ wie im vorigen Abschnitt definiert. $\mathcal{G}$ bezeichne die Menge der Vorbedingungen von Schritten (Goals, Ziele) in P und $\mathcal{T}$ die Menge der Threats. Eine Vorbedingung p am Schritt t sei bezeichnet durch $p@t$. Ein Effekt p eines Schrittes s sei durch $s \mapsto p$ notiert. Die Menge $varBindings(p, q)$ sei die Menge der Variablenbindungs-Constraints vom Typ $Same$, die notwendig sind, damit der Effekt q von Schritt s mit dem Prädikat p übereinstimmt. Die Menge $sepBindings(p, q)$ enthalte die Variablenbindungs-Constraints vom Typ $Same$ und $NotSame$, die notwendig sind, damit p und q nicht übereinstimmen.

SNLP$((\mathcal{S}, \mathcal{O}, \mathcal{B}, \mathcal{L}), \mathcal{G}, \mathcal{T})$

1. **Terminierung** falls $\mathcal{G} = \emptyset$ und $\mathcal{T} = \emptyset$
2. **Threat-Behandlung** falls $\mathcal{T} \neq \emptyset$
   - **Threat-Auswahl:** thr$\in \mathcal{T}$ Threat zwischen $u \mapsto q$ und $s \xrightarrow{p} t$
   - **Threat-Auflösung:** *Backtracking-Punkt*
     Auflösung durch Einführung einer zusätzlichen Ordnung (Promotion, $\mathcal{O}' := \mathcal{O} \cup \{t \prec u\}$, oder Demotion, $\mathcal{O}' := \mathcal{O} \cup \{u \prec s\}$), oder durch Einführung zusätzlicher Constraints, die p und q verschieden machen (Separation, $\mathcal{B}' := \mathcal{B} \cup sepBindings(p, q)$).
   - **Threat-Aktualisierung:** $\mathcal{T}'$:={„aktive Threats aus $\mathcal{T}$“}
   - **Rekursiver Aufruf:** $SNLP((\mathcal{S}, \mathcal{O}', \mathcal{B}', \mathcal{L}), \mathcal{G}, \mathcal{T}')$
3. **Zielbehandlung** falls $\mathcal{G} \neq \emptyset$
   - **Zielauswahl:** $q@t \in \mathcal{G}$
   - **Operatorauswahl:** *Backtracking-Punkt*
     Wähle Schritt $s \mapsto p$ (neuer Schritt oder existierender Schritt aus $\mathcal{S}$ mit Effekt p, der mit q unifiziert werden kann.)
   - **Aktualisiere Plan:**
     $\mathcal{S}' := \mathcal{S} \cup \{s\}$
     $\mathcal{O}' := \mathcal{O} \cup \{s \prec t\}$
     $\mathcal{B}' := \mathcal{B} \cup varBindings(p, q)$
     $\mathcal{L}' := \mathcal{O} \cup \{s \xrightarrow{p} t\}$
   - **Threat-Erkennung:**
     $\mathcal{T}'$:= {„aktive Threats für Plan $(\mathcal{S}', \mathcal{O}', \mathcal{B}')$“}
   - **Rekursiver Aufruf:** $SNLP((\mathcal{S}', \mathcal{O}', \mathcal{B}', \mathcal{L}'), \mathcal{G}', \mathcal{T}')$

Abbildung 2.1: Der SNLP-Grundalgorithmus [Barrett & Weld, 1992]

Der Algorithmus erzeugt sukzessive zu einem vorgegebenen Planungsproblem einen vollständigen, konsistenten Plan. Schritt für Schritt werden offene Vorbedingungen erfüllt, indem Causal Links durch Einführung neuer Schritte (Step Addition, Establishment) oder durch Ausnutzung existierender Schritte (Phantomisierung, Simple Establishment) erzeugt und auftretende Threats aufgelöst werden. Jeder dieser Schritte wird von [Kambhampati *et al.*, 1995a] *Planverfeinerungsentscheidung* genannt, *Entscheidung* deswegen, weil i. A. eine Auswahl aus einer Menge potentieller Alternativen zu treffen ist (vgl. Abschnitt 2.2), und *Verfeinerung*, weil der Plan durch Hinzufügen von Constraints *verfeinert* wird.

Der Algorithmus terminiert erfolgreich, wenn für alle Ziele ein geschützter Causal Link und somit ein vollständiger Plan gefunden wurde. Die Möglichkeit einer Teilzielrekursion, die beim Grundalgorithmus von SNLP auftreten kann, wurde durch eine spezielle Erweiterung von CAPLAN vermieden [Weberskirch, 1994]. Beim Algorithmus, wie er ursprünglich von [McAllester & Rosenblitt, 1991] definiert wurde, existiert das Problem der Nicht-Termination nicht, da dort auch dann Backtracking durchgeführt wird, wenn die Kosten des Planes, die über die Kosten von Planschritten definiert sind, einen gegebenen Maximalwert überschreiten. Der Maximalwert (Suchtiefe) wird dann während des Planungsprozesses iterativ erhöht.

Die Threat-Auflösung wurde in CAPLAN in Form eines *Protection Goal* realisiert, das erzeugt wird, sobald eine Bedrohung für einen Causal Link erkannt wird. Somit ergibt sich zusammen mit den zu erfüllenden offenen Vorbedingungen eine einheitliche Sicht auf eine Menge von Zielen, die durch geeignete Operatoren zu erfüllen sind. Die beiden Backtracking- bzw. Entscheidungspunkte werden somit zu einem einzigen und zum Ansatzpunkt für die Lernkomponente (Abschnitt 5.5.2).

Der Algorithmus arbeitet mit *chronologischem Backtracking*. Jede Möglichkeit, einen Causal Link zu erzeugen oder gegen Bedrohungen zu schützen, ist dabei ein Backtracking-Punkt. Kann an einer Stelle im Planungsprozeß kein geeigneter Causal Link erzeugt oder ein bedrohter Causal Link nicht geschützt werden, wird Backtracking durchgeführt.

2.2 Die Abhängigkeitsverwaltung REDUX

REDUX [Petrie, 1991] ist eine Erweiterung eines *Justification-Based Truth Maintenance Systems (JTMS)*, das eine Darstellung kausaler Abhängigkeiten zwischen Planungsentscheidungen erlaubt. In den folgenden Abschnitten wird von grundlgegenden Kenntnissen im Gebiet der Truth Maintenance Systeme ausgegangen (vgl. [Richter, 1992]). Auch werden nur die für diese Arbeit relevanten Punkte herausgegriffen. Einen genaueren Überblick gewährt [Weberskirch, 1994].

Die Hauptbegriffe für REDUX sind *Ziel* und *Operator*. Zur Erreichung von Zielen dienen Operatoren, deren Anwendung auf Ziele (Zielreduktion, siehe Abbildung 2.2) neue Ziele herleiten können. Mit dem initialen Ziel als Wurzel kann aus einer Folge von Operatoranwendungen eine Baumhierarchie von Zielen entstehen.

Die Zielreduktion hat hinsichtlich der zu lösenden Aufgabe unter Umständen auch Zuweisungen (*assignments*) zur Folge. Die Menge der momentan gültigen Zuweisungen beschreibt die aktuelle Teillösung für das gegebene Problem. Dabei ist eine Zuweisung nur dann gültig, wenn der zugehörige Operator gültig ist. Bei der Aktionsplanung sind konkrete Aktionen in der Planungswelt oder die Anordnung von Aktionen als Zuweisungen eines Operators zu verstehen. Diese Aktionen (Planschritte) und ihre Anordnung repräsentieren einen Plan.

Meist gibt es mehr als einen Operator, der auf ein Ziel anwendbar[3] ist, eine sog. *Konfliktmenge* (Abbildung 2.2). Dies führt zum bereits erwähnten Begriff der *Entscheidung*, der Auswahl eines Operators aus einer Konfliktmenge zur Reduzierung eines Ziels, für die eine Begründung

[3] Eine Abgrenzung der unterschiedlichen Verwendungen des Begriffs *Anwendbarkeit* findet sich in [Weberskirch, 1994].

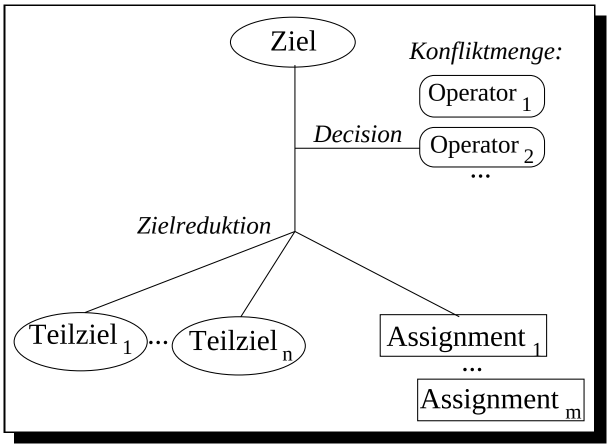

Abbildung 2.2: Zielreduktion in REDUX

definiert werden kann.

Ein Ziel, auf das ein Operator angewandt werden konnte, heißt *reduziert*. Sind alle Ziele reduziert, ist die gestellte Aufgabe gelöst und die mit den getroffenen Entscheidungen (Operatoranwendungen) assoziierten Zuweisungen beschreiben die Lösung.

Im allgemeinen gelingt es nicht, sofort eine Lösung zu finden. Meist werden im Laufe des Planungsprozesses lokal sinnvolle Entscheidungen getroffen, die nicht für die Gesamtlösung geeignet sind. Zum einen können Constraints über gültigen Zuweisungen verletzt werden, zum anderen kann für ein nicht reduziertes Ziel kein anwendbarer Operator gefunden werden, weil die Konfliktmenge leer ist oder nur zurückgewiesene Operatoren enthält. Es kommt zu *Backtracking* und dem Rückzug von Entscheidungen. Werden geeignete Methoden zur Inkonsistenzanalyse definiert, ist *dependency-directed Backtracking* möglich.

In engem Zusammenhang dazu steht die Möglichkeit, zwischen der Notwendigkeit zum Rückzug einer Entscheidung (*rejection*) und dem eigentlichen Rückzug (*retraction*) zu unterscheiden. Die Rückzugsnotwendigkeit führt auf jeden Fall zum Rückzug einer Entscheidung, aber eine nicht weiter bestehende Rückzugsnotwendigkeit hebt nicht automatisch den Rückzug auf. REDUX sieht hier vor, die Möglichkeit einer solchen lokalen Optimierung zunächst dem Problemlöser zu melden und die Entscheidung, wie darauf reagiert werden soll, dort zu belassen.

Der Rückzug einer Entscheidung muß in jedem Fall propagiert werden und verlangt daher Mechanismen, solche Änderungen an abhängige Assignments und Teilziele weiterzuleiten. Geeignete Mechanismen fanden sich in *Truth Maintenance Systemen*, speziell in einem JTMS [Doyle, 1979]. Für jedes Ziel und jede Operatoranwendung (Entscheidung) werden einige Standardabhängigkeitsstrukturen erzeugt, mit deren Hilfe sich die REDUX-Konzepte recht natürlich modellieren lassen.

Alle beschriebenen Abhängigkeitsstrukturen sind kleine TMS-Netze. Sie bestehen aus TMS-Knoten, die mittels ihres Labelings (`IN`/`OUT`) die Gültigkeit eines bestimmten Aspekts ausdrücken. Die Abhängigkeiten zwischen den Knoten werden durch TMS-Rechtfertigungen (*justifications*) beschrieben. Ein TMS-Knoten kann dabei mehrere Rechtfertigungen haben. Der Labeling-Mechanismus des JTMS hat die Aufgabe, für alle TMS-Knoten unter Berücksichtigung aller TMS-Rechtfertigungen ein konsistentes Labeling zu berechnen, ist also, von der eigentlichen REDUX-Funktionalität losgelöst, ein Mittel zu dessen Realisierung.

In CAPLAN speichert REDUX Abhängigkeiten zwischen Teilzielen, sowie Teillösungen, bereits erkundete Teile des Suchraumes und die damit verbundenen gültigen und ungültig gewordenen Operationen auf dem Plan. Dabei stellt das Abhängigkeitsnetz den aktuellen Zustand des Planungsprozesses dar. REDUX verhindert durch diese Abhängigkeitsverwaltung, daß CAPLAN zweimal denselben inkonsistenten Planzustand bearbeitet. Das SNLP-Verfahren sorgt durch seine systematische Suche für einen Suchbaum. CAPLAN läßt jedoch auch andere Komponenten zur Planungssteuerung zu, für die diese Systematik nicht gelten muß.

Es bleibt festzuhalten, daß für diese Arbeit vor allem die Assignments wichtig sind. Mit ihnen stehen die für die Operatorauswahl gültigen Zuweisungen zur Verfügung. Sie werden benötigt, um Inkonsistenzen im Plan festzuhalten.

2.3 Systemarchitektur von CAPLAN

Das CAPLAN-Grundsystem besteht, grob gesehen, aus Benutzerschnittstelle und Systemkern, (Abbildung 2.3). Über die Benutzerschnittstelle erfolgt die Domänen- und Problembe-

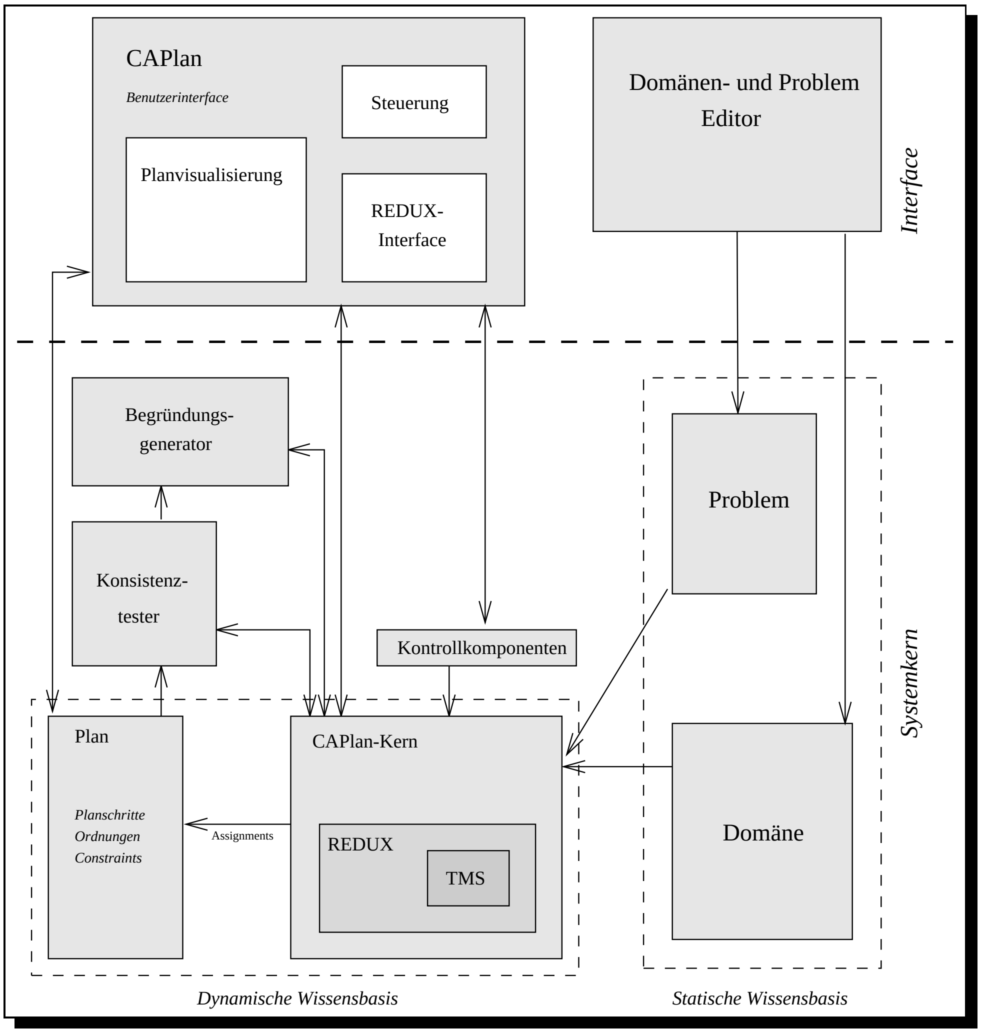

Abbildung 2.3: Das CAPLAN-Grundsystem

schreibung, sowie die Auswahl von Kontrollkomponenten und die Steuerung des Planungsprozesses. Dabei kann der Benutzer ein ganzes Spektrum von Eingriffsmöglichkeiten nutzen, welches von der vollautomatischen Planung bis hin zur Auswahl oder Zurückweisung einzelner Planungsentscheidungen reicht. Anhand einer visuellen Darstellung des aktuellen Planes und Listen der anstehenden Ziele und Threats kann der Benutzer den Verlauf des Planungsprozesses Schritt für Schritt verfolgen.

Der Systemkern läßt sich noch einmal unterteilen in die statische und die dynamische Wissensbasis. Die statische Wissensbasis beinhaltet die Domänen- und Problemspezifikation, die während des Problemlösevorgangs unverändert bleibt, wohingegen die dynamische Wissensbasis den aktuellen (partiellen) Planzustand mit seinen Variablenbindungen und Ordnungen auf den Planschritten, sowie die damit assoziierten TMS-Knoten verwaltet.

Der *Konsistenztester* prüft aktuelle Zuweisungen auf ihre Konsistenz. Zusammen mit dem *Begründungsgenerator* werden gegebenenfalls TMS-Rechtfertigungen erzeugt und an REDUX weitergereicht. Über das REDUX-Interface steht dem Benutzer die Möglichkeit offen, gezielt einzelne TMS-Knoten zu betrachten und zu verändern. Der Benutzer kann damit direkt in den Planungsprozeß eingreifen. Auch hierbei sorgt der Konsistenztester dafür, daß der Plan konsistent bleibt, indem gegebenenfalls bereits gefällte Entscheidungen zurückgezogen werden.

Bei den *Kontrollkomponenten* soll die Unterscheidung in solche, die den Suchprozeß steuern, und jene, die die Form des Backtracking bestimmen, nicht unerwähnt bleiben.

- Die Kontrollkomponenten zur Steuerung der Suche setzen an den Entscheidungspunkten Operatorauswahl und Threatauflösung an. Hier kann eine Regelverarbeitung zur Eingrenzung der Wahlmöglichkeiten zum Einsatz kommen.
- Für das Backtracking sind bisher zwei Arten in CAPLAN implementiert: zum einen chronologisches Backtracking als Standard-Verfahren und zum anderen eine einfache Variante des abhängigkeitsgerichteten Backtrackings (Backjumping). Die Backtracking-Komponente sorgt dafür, daß Entscheidungen, die für einen Fehlschlag verantwortlich sind, zurückgezogen werden.

Im Hinblick auf erklärungsbasiertes Lernen sind somit als Eingriffspunkte die Kontrollkomponente zur Steuerung der Suche für die Regelanwendung und die Backtracking-Komponente für die Regelgenerierung von Interesse.

Kapitel 3

Erklärungsbasiertes Lernen

Steven Minton untersuchte in [Minton, 1988] als erster die Möglichkeiten der Effizienzsteigerung für das total-ordnende Planungssystem PRODIGY mittels des ursprünglich aus dem Bereich des Konzeptlernens stammenden Technik des *erklärungsbasierten Lernens (EBL)*. Seine Terminologie basiert auf [Mitchell *et al.*, 1986]. Anhand eines Beispiels wird zunächst verdeutlicht, was *erklärungsbasierte Generalisierung (EBG)* ausmacht, aus der sich dann EBL entwickelte. Mintons Terminologie und sein Operationalisierungskriterium werden dann in Zusammenhang mit der Implementierung im System PRODIGY/EBL vorgestellt.

3.1 Erklärungsbasierte Generalisierung

Die Fähigkeit der Verallgemeinerung anhand von Beispielen gilt gemeinhin als Grundlage jeglichen Lernens. Generalisierung wird hierbei als Suche im Raum der Konzeptbeschreibungen aufgefaßt, welche durch Trainingsdaten, Annahmen und Domänenwissen eingegrenzt werden soll. Die Analyse eines einzelnen Trainingsbeispiels unter Hinzunahme von Domänenwissen wurde von [DeJong, 1981] *erklärungsbasierte Generalisierung (EBG)* genannt und besteht aus zwei Schritten: Erklärungsfindung und anschließende Generalisierung.

Während induktive, ähnlichkeitsbasierte Methoden keine Rechtfertigungen für die erzeugten Generalisierungen liefern können, ist gerade dies bei erklärungsbasierten Methoden der Fall. Mittels Domänenwissen wird eine *Erklärung* dafür erbracht, warum ein betrachtetes Trainingsbeispiel Instanz einer Konzeptbeschreibung ist. Genau darin liegt der Vorteil erklärungsbasierten Lernens: Eine Erklärung ist ein *deduktiver Beweis* für die Zugehörigkeit eines Beispiels zu einem Zielkonzept (vgl. [Richter, 1992]). Somit gilt eine gewonnene Erklärung innerhalb der verwendeten Hintergrundtheorie uneingeschränkt und man muß keinerlei Revision des Erlernten befürchten.

Beispiel. Anhand eines Beispiels aus [Winston *et al.*, 1983] über Lernen aus funktionalen Definitionen physischer Beschreibungen soll nun die Methode des erklärungsbasierten Generalisierens vorgestellt werden. Das Ziel ist bei dem Beispiel, die strukturelle Definition einer Tasse zu erlernen.

Abbildung 3.1 zeigt eine Beschreibung für das Konzept, welches gelernt werden soll. Das Zielkonzept ist eine *Tasse* (cup), die eine Klasse von Objekten definiert, die *offene Gefäße* (open vessel) und *anhebbar* (liftable) sind, und eine *stabile Lage* (stable) einnehmen. Die Domänentheorie enthält Regeln, die diese Eigenschaften auf einfachere strukturelle Eigen-

Gegeben:

- Zielkonzept:
  Eine Klasse von Objekten x, für die gilt:
  $Cup(x) \Leftrightarrow Liftable(x) \wedge Stable(x) \wedge OpenVessel(x)$

- Trainingsbeispiel:
  $Color(obj1, red)$
  $PartOf(obj1, concavity1)$
  $PartOf(obj1, handle1)$
  $Is(obj1, light)$
  $IsA(handle1, handle)$
  $\ldots$

- Domänentheorie:
  $Is(x, light) \wedge PartOf(x, y) \wedge IsA(y, handle) \rightarrow Liftable(x)$
  $PartOf(x, y) \wedge IsA(y, bottom) \wedge Is(y, flat) \rightarrow Stable(x)$
  $PartOf(x, y) \wedge IsA(y, concavity) \wedge Is(y, upwardPointing) \rightarrow OpenVessel(x)$
  $\ldots$

- Operationalisierungskriterium:
  Die Konzeptbeschreibung muß mittels struktureller Merkmale erfolgen, die bei der Beispieldefinition verwendet wurden (z. B. *light*, *handle*, *flat* usw.).

Gesucht:

- Eine Verallgemeinerung des Trainingsbeispiels, das dem Operationalisierungskriterium genügt.

Abbildung 3.1: Beispiel für erklärungsbasiertes Lernen nach [Winston *et al.*, 1983]

schaften abbildet, z. B. einen flachen Boden (flat) und einen Griff (handle). Das Operationalisierungskriterium verlangt hier, daß die gefundene Konzeptbeschreibung zum Wiedererkennen einer Tasse geeignet ist, die Beschreibung also die gleichen Begriffe wie das Trainingsbeispiel verwendet.[1]

Die Erklärungsstruktur für ein Trainingsbeispiel, welches eine bestimmte Tasse, $obj1$, beschreibt, ist in Abbildung 3.2 dargestellt. Die Erklärungsstruktur ist der Beweisbaum, der sich aus der Anwendung der Regeln aus der Domäne ergibt, wobei im Beweisbaum nur instanziierte Regeln vorkommen.

Die Erklärung rechtfertigt, warum $obj1$ zur Klasse *cup* gehört, indem es beweist, warum $obj1$ ein anhebbares, offenes Gefäß ist und zudem hingestellt werden kann, ohne umzufallen. Das EBG-Verfahren unterscheidet bei der Erzeugung der Erklärungsstruktur zwischen relevanten Eigenschaften, z. B. Gewicht, und unwichtigen Eigenschaften, z. B. Farbe. Die Relevanz einer Eigenschaft bezieht sich hierbei auf die Bedeutung bezüglich des Zielkonzeptes. Dies wird durch das Operationalisierungskriterium erreicht, welches vorgibt, mit welchen einfachen und damit leicht auszuwertenden Prädikaten ein gelerntes Konzept zu beschreiben ist. In diesem Beispiel sind dies $PartOf$, IsA und Is. Zur Bestimmung der hinreichenden schwächsten

[1] Der Begriff *Wiedererkennen* ist hier im eigentlichen Sinne des Wortes zu verstehen, vgl. [Mitchell *et al.*, 1986].

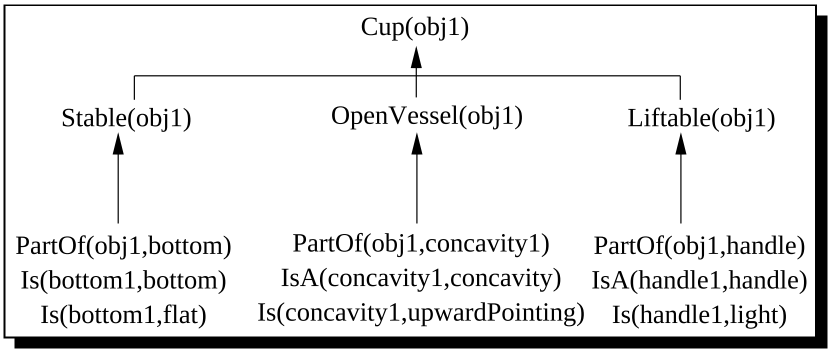

Abbildung 3.2: Erklärungsstruktur für das Konzept *Cup*

Vorbedingungen, unter denen eine Regel angewendet werden kann, wird das Zielkonzept einer Rückwärtspropagierung durch die Erklärungsstruktur (*Regression*) unterzogen, wiederum unter Beachtung des Operationalisierungskriteriums.

Eine Generalisierung der Erklärung aus Abbildung 3.2 ergibt:

$$PartOf(x, xc) \wedge IsA(xc, concavity) \wedge Is(xc, upwardPointing) \wedge$$
$$PartOf(x, xb) \wedge IsA(xb, bottom) \wedge Is(xb, flat) \wedge$$
$$PartOf(x, xh) \wedge IsA(xh, handle) \wedge Is(x, light)$$
$$\rightarrow Cup(x)$$

Die verwendeten Begriffe werden im nächsten Abschnitt stärker formalisiert.

3.2 Terminologie des EBL-Verfahrens

Verschiedene Arbeiten haben zu einer standardisierten Ausdrucksweise auf dem Gebiet des erklärungsbasierten Lernens geführt [DeJong & Mooney, 1986; Keller, 1988; Mitchell *et al.*, 1986; Rosenbloom & Laird, 1986]. Abbildung 3.3 zeigt ein abstraktes Schema, welches Ein- und Ausgabe erklärungsbasierten Lernens definiert. EBL startet mit einem abstrakten *Zielkonzept* und einem *Trainingsbeispiel*. Unter Benutzung einer *Domänentheorie* als einer Menge von Axiomen, die die Domäne beschreiben, können wir erklären, warum ein Trainingsbeispiel eine Instanz des Zielkonzepts ist. Da eine Erklärung, wie bereits erwähnt, ein Beweis dafür ist, daß ein Trainingsbeispiel ein vorgegebenes Zielkonzept erfüllt, läßt sich die Terminologie leicht auf die Gebiete des Theorembeweisens bzw. Problemlösens übertragen (vgl. [DeJong & Mooney, 1986]). Minton wählte die Terminologie des Theorembeweisens, da hier bereits eine stärkere Standardisierung der Ausdrucksweise vorlag.

EBL-Verfahren. Das EBL-Verfahren erzeugt nun durch Suche nach der schwächsten Vorbedingung, für die eine Erklärung gültig ist, eine *gelernte Beschreibung*, die sowohl eine Generalisierung des Trainingsbeispiels als auch eine Spezialisierung des Zielkonzepts ist. Die gelernte Beschreibung muß einem *Operationalisierungskriterium* genügen, einem Test, der sicherstellen soll, daß mittels dieser Beschreibung das Zielkonzept effizient wiedererkannt wird. Erst durch dieses Kriterium wird erklärungsbasiertes Lernen sinnvoll, da es die Anwendbarkeit der gelernten Beschreibung garantiert.

Die meisten EBL-Systeme basieren auf der Annahme einer vollständigen und korrekten Domänentheorie, d.h. für alle positiven Instanzen (und nur für positive Instanzen!) des

Gegeben:

- Zielkonzept: Ein Konzept, das gelernt werden soll.
- Trainingsbeispiel: Ein Beispiel des Zielkonzepts.
- Domänentheorie: Eine Menge von Regeln und Fakten, die zur Erklärung dafür dienen, warum das Trainingsbeispiel das Zielkonzept erfüllt.
- Operationalisierungskriterium: Ein Prädikat über Beschreibungen. Es spezifiziert die Art und Weise, in der eine gelernte Beschreibung ausgedrückt werden muß.

Gesucht:

- Eine Beschreibung, die sowohl eine Verallgemeinerung des Trainingsbeispiels als auch eine Spezialisierung des Zielkonzepts darstellt und das Operationalisierungskriterium erfüllt.

Abbildung 3.3: EBL-Spezifikation nach [Mitchell *et al.*, 1986]

Zielkonzepts kann mit Hilfe der Domänentheorie ihre Zugeörigkeit zum Zielkonzept *bewiesen* werden.[2] Der Zweck dieser Systeme liegt also nicht darin, *mehr* über das Zielkonzept zu erlernen, sondern darin, die Beschreibung des Zielkonzeptes umzuformulieren, die Beschreibung zu operationalisieren.[3] Obwohl die exakte Definition der Anwendbarkeit von EBL-System zu EBL-System variiert, bleibt effiziente Wiedererkennung die Basis jedes Operationalisierungskriteriums. Denn wäre Effizienz kein Thema, könnte das Zielkonzept, so wie es in der Domänentheorie definiert wurde, ohne Veränderung verwendet werden.

EBL-Programm. Ein einfaches EBL-Programm arbeitet auf folgende Weise: Nach Eingabe eines Trainingsbeispiels erzeugt das System eine gelernte Beschreibung, eine Verallgemeinerung des Beispiels. Wird das nächste Trainingsbeispiel nicht von dieser Beschreibung abgedeckt, erzeugt das Programm eine neue gelernte Beschreibung. Wird ein drittes Trainingsbeispiel weder von der ersten Beschreibung noch von der zweiten abgedeckt, wird eine weitere gelernte Beschreibung erzeugt, etc. Auf diese Weise formuliert das Programm das Zielkonzept immer wieder um. Es erzeugt eine Menge disjunktiv verknüpfter gelernter Beschreibungen. Dieses *Kontrollwissen* in Form von Regeln muß komplett überprüft werden, bevor feststeht, daß ein Beispiel keine Instanz einer Konzeptbeschreibung ist.

3.3 Kosten

Das Operationalisierungskriterium soll sicherstellen, daß die gelernten Beschreibungen effizient getestet werden können. Dabei werden meist die kumulativen Kosten solcher Tests unterschlagen. Auch wenn der Test einer einzelnen gelernten Beschreibung weniger kosten

[2] Eine Diskussion der Probleme schwächerer Domänentheorien findet sich in [Mitchell *et al.*, 1986].

[3] [Keller, 1988] unterteilt den Raum der Konzeptbeschreibungen in einen nichtoperationalen und einen operationalen Anteil mit vielen synonymen Konzepten. Das Operationalisierungskriterium ist in diesem Zusammenhang als Präferenzkriterium zu vestehen, das eine operationale Beschreibung einer anderen vorzieht.

mag als ein Test des Zielkonzeptes, kann es wesentlich teurer sein, die ganze Menge der Disjunktionen zu testen als das Zielkonzept selbst.

Desweiteren vernachlässigt das einfache Modell die Beziehungen zwischen dem Zielkonzept und dem Problemlöser, dessen Leistung gesteigert werden soll. Liegt die Aufgabe des Problemlösers darin, das Zielkonzept einfach nur wiederzuerkennen, dann reicht es aus, das ursprüngliche Zielkonzept durch die Menge der Disjunktionen zu ersetzen. Im System Prodigy/EBL, das verschiedene Zielkonzepte kennt, sind die Beziehungen zwischen dem Problemlöser und den Zielkonzepten sehr komplex. Dort hängen die Lerneffekte von der Verwendungsart der gelernten Beschreibungen ab. Fehlt zudem ein Operationalisierungskriterium, führt Lernen zu einem Leistungsverlust des Gesamtsystems. Oft arbeitet das ursprüngliche System ohne Kontrollwissen effizienter, da ein Test des Kontrollwissens zuviel Zeit in Anspruch nimmt.

Hauptfaktoren für die Leistungsminderung sind:

- geringe Anwendungshäufigkeit des Kontrollwissens,
- geringer Nutzen gelernter Beschreibungen und
- hohe Match-Kosten.

[Minton, 1988] analysierte in diesem Zusammenhang das *Utility-Problem*, die Frage nach dem Nutzen von Kontrollwissen, konnte aber kein allgemeines Verfahren angeben, um die dabei anfallenden Kosten entscheidend zu senken.

3.4 Erklärungsbasiertes Lernen im System Prodigy/EBL

Dieser Abschnitt erläutert kurz die Kernpunkte des STRIPS-artigen, total-ordnenden Planungssystems Prodigy/EBL und stellt das Zielkonzept *Lernen aus Fehlschlägen* vor.

3.4.1 Allgemeines

Prodigy/EBL verwendet erklärungsbasiertes Lernen in der Form der *erklärungsbasierten Spezialisierung* (EBS) von Zielkonzepten. EBS nimmt eine Zielkonzeptbeschreibung und spezialisiert sie zu einer gelernten Beschreibung. Der Spezialisierungsprozeß erweitert rekursiv die Zielkonzeptbeschreibung, indem durch Domänenaxiome definierte Substitutionen ausgeführt werden. Das Trainingsbeispiel wird dabei zur Auswahl der geeigneten Axiome benutzt. Minton versteht diesen Prozeß als Beweisverfahren, da er äquivalent zur Erweiterung eines Beweises über die Erfüllung des Zielkonzeptes ist.

Zusätzlich zur erklärungsbasierten Spezialisierung verwendet Prodigy/EBL weitere Techniken zur Effizienzsteigerung:

- Deklarative Definition der Zielkonzepte
- Kompression von Regeln
- Nutzenberechnung der Regelanwendungen

Das System kennt *mehrere Arten von Zielkonzepten*. Es kann sowohl erklären, warum ein Operator erfolgreich angewandt werden konnte bzw. fehlschlug, als auch, warum zwei Ziele interferieren. Durch die deklarative Definition der Zielkonzepte kann jederzeit ein neues Zielkonzept zum System hinzugefügt werden.

Desweiteren versucht PRODIGY/EBL, Regeln mit hohen Match-Kosten zu vereinfachen, sie zu *komprimieren*. Dies entspricht der Einführung von Vereinfachungsaxiomen in die ursprüngliche Erklärung und kann als Suche im Raum alternativer Erklärungen aufgefaßt werden.

Schließlich berechnet das System das Verhältnis der Match-Kosten zu den eingesparten Suchkosten. Beim Lernen einer Regel wird abgeschätzt, welche Suchersparnis sie erbringt. Bei weiteren Programmläufen wird dann die Einschätzung der Ersparnis immer wieder aktualisiert.

3.4.2 Lernen aus Fehlschlägen

Das zentrale Thema dieser Arbeit ist der Erwerb von Kontrollwissen durch erklärungsbasiertes Lernen aus Fehlschlägen in der Planung. In Abschnitt 1.1 wurde bereits vorgestellt, was der Begriff *Fehlschlag* in der Planung bedeutet. Bezogen auf die Hintergrundtheorie des total-ordnenden Planens sind dies nicht erfüllte Ziele und Operatoren, die aufgrund inkonsistenter Bindungen nicht anwendbar sind. Dieses Zielkonzept soll anhand von PRODIGY/EBL kurz vorgestellt werden, bevor im nächsten Kapitel die Version für partiell-ordnende Planer im Detail beschrieben wird.

Eine Planungsentscheidung (vgl. auch Abschnitt 2.1.3) kann in PRODIGY/EBL die Wahl eines Knotens[4], eines Ziels, eines Operators oder einer Variablenbindung sein. Die Entscheidung schlägt fehl, wenn sie nur in Sackgassen führt, d. h. jede mögliche Alternative inkonsistent ist. Sobald PRODIGY/EBL auf einen derartigen Fehler trifft, analysiert das System den betroffenen Teilplan und erzeugt daraus eine Zurückweisungsregel für die entsprechende Planungsentscheidung. Die vier Zielkonzepte hierfür lauten *node-fails*, *goal-fails*, *op-fails* und *bindings-fail*. Sie stehen für die o.g. vier Entscheidungspunkte in PRODIGY/EBL an denen ein Fehlschlag eintreten kann.

Die Zielkonzepte bauen aufeinander auf, wie Abbildung 3.4 ausschnittweise zeigt.[5] Die verschiedenen Schemata zeigen einen kleinen Ausschnitt der deklarativen Definitionen der Zielkonzepte in PRODIGY/EBL. Schema-F1 besagt etwa, daß ein Fehlschlag an einem Knoten durch einen Fehlschlag eines Zieles an diesem Knoten impliziert wird. Schema-F2 sagt, daß dieser Fehlschlag wiederum durch einen fehlgeschlagenen Operator unter Berücksichtigung aller relevanten Bindungsmöglichkeiten herbeigeführt wurde. Weitere Schemata, die hier nicht aufgeführt sind, beschreiben Zyklen im Goalstack und zwischen Zuständen, Aspekte, die im Hinblick auf SNLP weniger wichtig sind. Ebenfalls nicht gezeigt, sind die Schemata für Zurückweisungen von Zielen, Operatoren und Bindungen durch Kontrollregeln.

---

[4] Damit sind Knoten im Suchbaum gemeint. Tiefensuche ist in PRODIGY/EBL das Standardsuchverfahren.

[5] Die Darstellung ist bereits in [Minton, 1988] aus Gründen der besseren Lesbarkeit vereinfacht.

Schema-F1: Ein Knoten schlägt fehl, wenn ein Ziel an diesem Knoten fehlschlägt.
`(node-fails` *`node`*`)`
`if ((atomic-formula` *`goal`*`) and (is-goal` *`node goal`*`) and`
`(goal-fails` *`goal node`*`))`

Schema-F2: Ein Ziel schlägt fehl, wenn kein Operator dieses Ziel erfüllt.
`(goal-fails` *`goal node)`*
`if (forall` *`op`* `such-that (is-op` *`op`*`) (op-fails` *`op goal node`*`))`

Schema-F3: Ein Operator schlägt fehl, wenn er für das Ziel nicht relevant ist.
`(op-fails` *`op goal node`*`)`
`if (forall` *`effect`* `such-that (is-effect` *`op effect`*`)`
`(does-not-match` *`effect goal`*`))`

Schema-F4: Ein Operator schlägt fehl, wenn er zwar relevant für das Ziel ist, aber alle Bindungsmöglichkeiten fehlschlagen.
`(op-fails` *`op goal node`*`)`
`if ((forall` *`bindings`*`) such-that (is-relevant-bindings` *`bindings`*
`op goal``) (bindings-fail` *`bindings op goal`*`))`

Schema-F5: Eine Menge von Bindungen schlägt fehl, wenn der Operator damit nicht angewendet werden kann und die Bildung von Unterzielen fehlschlägt.
`(bindings-fail` *`bindings op goal node`*`)`
`if ((not-applicable` *`bindings op node`*`) and`
`(child-node-after-subgoaling` *`childnode bindings op node`*`) and`
`(node-fails` *`childnode`*`))`

Schema-F6: Eine Mengen von Bindungen schlägt fehl, wenn die Anwendung des damit instanziierten Operators fehlschlägt.
`(bindings-fail` *`bindings op goal node`*`)`
`if ((is-applicable` *`bindings op node`*`) and`
`(child-node-after-subgoaling` *`childnode bindings op node`*`) and`
`(node-fails` *`childnode`*`))`

Abbildung 3.4: Zielkonzepte für das Lernen aus Fehlschlägen [Minton, 1988]

Kapitel 4

Lernen aus Fehlschlägen für partiell-ordnende Planer

[Kambhampati *et al.*, 1995a] übertrug erklärungsbasiertes Lernen aus der Welt der total-ordnenden Planer auf die Welt der partiell-ordnenden Verfahren. Ihre Erweiterung des SNLP-Algorithmus, SNLP+EBL, soll hier vorgestellt und in Bezug gesetzt werden zu den Definitionen in [Minton, 1988].

Nach einer kurzen Motivation wird zunächst gezeigt, wie initiale Erklärungen gebildet werden. Die erzeugten Constraint-Mengen werden dann einer Regression über Planungsentscheidungen unterzogen, um festzustellen, welche Bedingungen im Plan gegolten haben, bevor ein Operator angewandt wurde. Das Ergebnis wird im Suchbaum propagiert und zu einer Regel zusammengesetzt. Die Generalisierung der Regel erlaubt dann eine breitere Anwendbarkeit derselben. Der vorletzte Abschnitt geht auf das Lernen aus Situationen ein, in denen der Planer eine vorgegebene Suchtiefe überschreitet. Die dort vorgestellte Vorgehensweise ist domänenabhängig. Zuletzt werden verschiedene Erweiterungen des EBL-Ansatzes von [Kambhampati *et al.*, 1995a] vorgestellt.

4.1 Motivation

Das Ziel erklärungsbasierten Lernens ist, wie bereits in Kapitel 1 gesagt, die Erzeugung von Regeln, die den zu durchsuchenden Raum einschränken, hier eine Eingrenzung der Menge der möglichen Planzustände.

Suchkontrollregeln sollen dem zugrundeliegenden Planungsalgorithmus zusätzliche Informationen zur Verfügung stellen, damit dieser bessere Entscheidungen treffen kann. Diese Entscheidungspunkte sind, wie in Kapitel 2 bereits vorgestellt, offene Vorbedingungen (Einführung neuer Schritte und Phantomisierungen) und Threats. Ziel- und Threat-Auswahl läßt sich nicht mit EBL-Verfahren erlernen, da SNLP kein Backtracking über diese Entscheidungen durchführt.[1] Da das Lernen aus Fehlschlägen den größten Nutzen hat (vgl. [Minton, 1988]), beschränkt sich SNLP+EBL darauf. Sobald SNLP+EBL einen inkonsistenten Plan erzeugt, versucht der Planer eine Erklärung zu finden und daraus eine Suchkontrollregel zu erzeugen. Mit Hilfe dieser Regeln sollen im späteren Planungsverlauf ähnliche Fehler vermie-

[1]Nichtlernende Verfahren, die in diesem Zusammenhang erwähnt werden sollten, sind die verzögerte Threat-Behandlung von Peot und Smith [Peot & Smith, 1993], welches in CAPLAN zur Verfügung steht, sowie die statische Domänenanalyse von Etzioni mittels sogenannter *Problemlösungsgraphen* (PSG) [Etzioni, 1993].

den werden.

4.2 Fehlgeschlagene Pläne und initiale Erklärungen

Lernen aus Fehlschlägen beginnt im einfachsten Fall an der Stelle, an der der Planer durch Auswahl falscher Alternativen, z. B. beim Entscheidungspunkt Operatorauswahl, einen inkonsistenten Plan erzeugt. Allgemeiner determinieren drei Situationen einen fehlgeschlagenen Plan:

- Analytische Fehler,
- Suchtiefenüberschreitungen und
- Anwendungen von Suchkontrollregeln.

In jedem dieser Fälle begründet SNLP+EBL den Fehlschlag, indem eine Menge von Constraints bestimmt wird, die zusammengenommen inkonsistent sind. Diese Menge von Constraints ist die *initiale Fehlererklärung* eines partiellen Plans.

Die Fehlersituationen sollen im folgenden näher erläutert werden.

4.2.1 Analytische Fehler

Unter einem analytischen Fehler versteht man zum einen Bindungs- oder Ordnungsinkonsistenzen in einem Plan, zum anderen fehlende Möglichkeiten zur Erfüllung einer offenen Vorbedingung (Establishment-Fehler).

Ordnungsinkonsistenz. Zwei in einem Zyklus angeordnete Planschritte zeigen eine fehlerhafte Situation an. Beispiel: Sind die Schritte s_1 und s_2 so angeordnet, daß $(s_1 \prec_{trans} s_2) \in \mathcal{O}$ und $(s_2 \prec s_1) \in \mathcal{O}$ gilt, so repräsentiert dies eine Inkonsistenz im partiellen Plan. Die initiale Fehlererklärung ist die Konjunktion der Constraints: $(s_1 \prec_{trans} s_2) \wedge (s_2 \prec s_1)$.

Bindungsinkonsistenz. Bindungsinkonsistenzen treten auf, wenn für mindestens eine Variable keine konsistente Bindungsmöglichkeit gefunden werden kann. Beispiel: Es gebe eine Variable x im partiellen Plan, für die sowohl $Same(x, A)$, als auch $NotSame(x, A)$ gelten soll. Die initiale Fehlererklärung ist hierbei ebenfalls die Konjunktion der Constraints: $(Same(x, A)) \wedge (NotSame(x, A))$.

Fehlende Establishment-Möglichkeit. Trifft der Planer auf eine offene Vorbedingung p am Schritt s im partiellen Plan $\mathcal{P}$, für die er keinen geeigneten Operator und keine Phantomisierungsmöglichkeit findet, dann ist dieser Fehlertypus gegeben. Hier müssen zwei Tatsachen beachtet werden. Entweder existiert kein Operator in der Domäne DOM, der den gesuchten Effekt hat, oder p ist nicht wahr im Startzustand Σ. Erstere Tatsache ist unabhängig vom bearbeiteten Problem. Das zweite Faktum kann jedoch in einer anderen Problemstellung gegeben sein. Aus diesem Grund muß letzteres in die initiale Fehlererklärung mitaufgenommen werden: $p@s \wedge \neg initiallyTrue(p)$. $initiallyTrue(p)$ gibt hierbei an, ob p Teil des Startzustandes Σ des Problems ist.

Wie man sieht, liefert jeder der analytischen Fehler eine Menge von Constraints oder eine Eigenschaft der Start-Situation des Problems als Grund für einen Fehlschlag.

4.2.2 Tiefenüberschreitungsfehler

Dieser Fehler wird angezeigt, wenn eine vorgegebene Suchtiefe überschritten wird. Im Gegensatz zu den analytischen Fehlern haben sie keine direkte Fehlererklärung. Die Feststellung der Suchtiefenüberschreitung allein genügt nicht für eine Fehlererklärung, es muß vielmehr eine Untermenge aus der Menge der Plan-Constraints herausgesucht werden, die möglicherweise in ihrer Gesamtheit inkonsistent sind. [Kambhampati *et al.*, 1995a] stellt hierfür einen domänenaxiombasierten Ansatz vor, der in Abschnitt 4.6 erläutert wird.

4.2.3 Anwendung einer Zurückweisungsregel

Sollte eine zuvor gelernte Regel angewendet worden sein, wird dadurch ein Teil des Suchbaums zurückgewiesen. Somit kennzeichnet dies ebenfalls einen fehlgeschlagenen Plan. Die Fehlererklärung, aufgrund derer diese Regel erzeugt wurde, dient hier als neue Fehlererklärung.

4.3 Regression

Um aus einer initialen Fehlererklärung Nutzen zu ziehen, muß sie im Suchbaum hinaufpropagiert werden. Diese Rückwärtspropagierung über die angewandten Operatoren heißt *Regression* und wird in diesem Abschnitt ausführlich beschrieben.

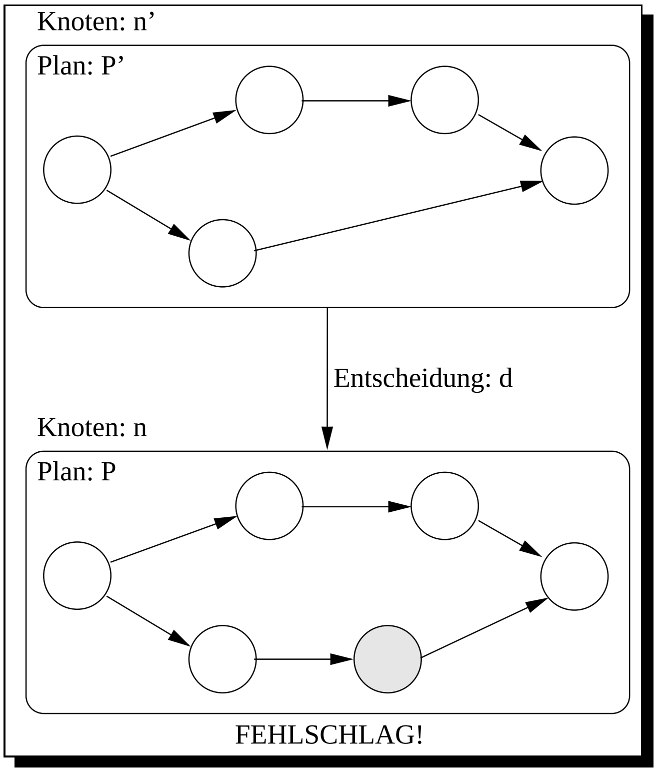

Abbildung 4.1: Erfolglose Planvervollständigung

Betrachten wir den Ausschnitt aus einem Suchbaum in Abbildung 4.1. Der Plan $\mathcal{P}$ sei am Knoten n fehlgeschlagen, nachdem der Plan $\mathcal{P}'$ durch die Entscheidung d in $\mathcal{P}$ überführt worden ist. Hier würden wir uns wünschen, daß eben diese Entscheidung d nicht getroffen worden wäre. Um also diese Entscheidung zu vermeiden, benötigen wir eine minimale Menge von Constraints $\mathcal{E}'$ des Planes $\mathcal{P}'$, mit deren Hilfe wir den Fehlschlag für $\mathcal{P}$ hätten vorhersehen können. Da wir nun wissen, daß $\mathcal{P}$ fehlschlug, weil die Constraints $\mathcal{E}$ in $\mathcal{P}$ enthalten sind, muß $\mathcal{E}'$ bereits vor der Entscheidung d gegolten haben.

Die folgenden Unterabschnitte zeigen detailliert, wie diese minimale Menge von Bedingungen erzeugt wird. Zuerst betrachten wir die Regression eines einzelnen Constraints, dann die Verallgemeinerung dieses Prozesses auf Mengen von Constraints und wenden uns schließlich konkret der Rückwärtspropagierung über Planungsentscheidungen in SNLP zu.

4.3.1 Regression eines Constraints

Für zustandsbasierte Planer gestaltet sich Regression problemlos, da eine Entscheidung der Anwendung eines Domänenoperators gleichkommt und somit eine Regression über Entscheidungen im Grunde eine Regression über Operatoren ist. Die Regression dieses Typs wurde besonders gut bei STRIPS-Operatoren untersucht [Nilsson, 1980]. Partiell-ordnende Planer dagegen transformieren durch eine Entscheidung einen partiellen Plan in einen anderen. Jedoch läßt sich auch hier Regression leicht formalisieren, sobald man sich vor Augen hält, daß eine Planungsentscheidung durch SNLP ein STRIPS-Operator mit Vorbedingungen und Effekten ist, welche wiederum aus Constraints auf dem partiellen Plan bestehen.

Das Ergebnis der *Regression eines Constraints c über eine Entscheidung d*, die *Regrediente*, ist die Menge aller Constraints, die ein partieller Plan vor dem Fällen der Entscheidung enthalten muß, damit genau dieser Constraint c nach der Entscheidung d enthalten ist.

Formal läßt sich die Regression eines Constraint c über eine Entscheidung d mit Effekten $eff(d)$ folgendermaßen definieren:

$$\begin{array}{lcll} Regress(c,d) & = & True\text{, falls } c \in eff(d) & \text{Fall 1} \\ & = & c\text{, sonst} & \text{Fall 2} \end{array}$$

Somit ist das Ergebnis der Regression eines Constraints c, der durch die Entscheidung d in einen partiellen Plan eingefügt wird, $True$, falls c ein Effekt von d ist (Fall 1). Die Regrediente ist c, falls d diesen Constraint nicht hinzufügte, d. h. c bereits vorhanden war (Fall 2). In Fall 1 heißt das, daß der Constraint c von der Entscheidung d verursacht wurde, und das Ergebnis $True$ sagt aus, daß die Regression von c über d keine weitere Aussage mehr bringt; die Erklärung sagt nichts mehr über den Fehler aus, und c ist damit vor Anwendung von d uninteressant. Lernen führt im zweiten Fall statt: Hier sagt das Ergebnis der Regression, daß c bereits vor Anwendung von d problematisch war, aber dies zu dem Zeitpunkt noch nicht erkannt werden konnte.

Transitive Constraints müssen gesondert behandelt werden. Constraints heißen *transitiv*, wenn zwei Constraints $c_1 \wedge c_2$ einen dritten Constraint c_3 implizieren ($c_1 \wedge c_2 \vdash c_3$). Ordnungs- und Bindungsconstraints sind transitiv, denn es gilt: $((s_1 \prec s_2) \wedge (s_2 \prec s_3)) \vdash (s_1 \prec s_3)$.

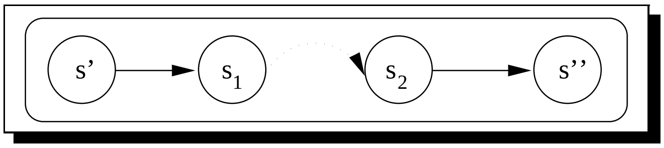

Abbildung 4.2: Transitive Constraints

Sobald eine Planungsentscheidung einen transitiven Constraint zum Plan hinzufügt, werden implizit alle Constraints, die transitiv dazugehören, ebenfalls hinzugefügt. In Abbildung 4.2 werden die Schritte s' und s'' durch die Einführung der Ordnung $s_1 \prec s_2$ ebenfalls angeordnet. Die Regression eines transitiven Constraints c über eine Entscheidung d muß also den Fall in Betracht ziehen, in dem der Plan vor dem Fällen der Entscheidung d den Constraint c' enthält, und d den Constraint c'' hinzufügt, so daß c' und c'' transitiv c nach sich ziehen:

$$Regress(c, d) \quad = \quad c', \text{ falls } c'' \in eff(d) \wedge (c'' \wedge c') \vdash c \quad \text{Fall 3}$$

Unter Umständen gibt es mehrere Mengen von c', die c mit c'' nach sich ziehen. Hier ergibt sich als Regressionsresultat eine disjunktive Verknüpfung der Constraintmengen. In Abbildung 4.2 ist die Regrediente von $s' \prec s''$ über die Ordnungsentscheidung $s_1 \prec s_2$ folgende Disjunktion: $(s' \prec s'') \vee [(s' \prec s_1) \wedge (s_2 \prec s'')]$ (Fall 3).

4.3.2 Regression von Fehlererklärungen

Die Regression von Fehlererklärungen ist nach der Erläuterung der Regression einzelner Constraints einfach. Da eine Fehlererklärung eine Menge von Constraints eines partiellen Plans ist, besteht eine Regression der Fehlererklärung über eine Entscheidung aus einer Regression jedes Constraints dieser Menge. Die Einzelresultate werden konjunktiv verknüpft.

Sei $\mathcal{E} = c_1 \wedge c_2 \wedge \ldots \wedge c_i$, dann gilt:

$$Regress(\mathcal{E}, d) = Regress(c_1, d) \wedge Regress(c_2, d) \wedge \ldots \wedge Regress(c_i, d)$$

Die Regression von $\mathcal{E}$ führt zum Teil zu disjunktiv verknüpften Ergebnissen $\mathcal{E}' \vee \mathcal{E}'' \vee \ldots$ Da wir wissen wollen, welcher Teil des aktuellen Plans für das Scheitern verantwortlich ist, *benutzen wir nur den Teil der Fehlererklärung, der im Plan auch tatsächlich enthalten ist.*

Dies kann an Abbildung 4.2 veranschaulicht werden. Dort wurde $(s' \prec s'')$ über die Entscheidung regrediert, die $(s_1 \prec s_2)$ ordnen wollte, mit dem Ergebnis: $(s' \prec s'') \vee [(s' \prec s_1) \wedge (s_2 \prec s'')]$. Es wird aber nur $[(s' \prec s_1) \wedge (s_2 \prec s'')]$ berücksichtigt, da $(s' \prec s_1)$ und $(s_2 \prec s'')$ im partiellen Plan enthalten waren, bevor die Ordnungsentscheidung getroffen wurde.

4.3.3 Regression über Entscheidungen von SNLP

Nach der Erläuterung des allgemeinen Prinzips der Regression soll es in diesem Abschnitt an Planverfeinerungsentscheidungen, wie sie in SNLP verwendet werden (vgl. Abschnitt 2.1.3), konkretisiert werden. Die nachfolgenden Abbildungen enthalten jeweils oben eine exakte Definition einer Planverfeinerungsentscheidung, bestehend aus Vorbedingungen, die die Anwendbarkeit der Entscheidung definieren, sowie Effekten, die die Auswirkungen der Entscheidung auf den Plan festhalten. Darunter ist in einer Tabelle für jeden Constraint, der in den Effekten der Entscheidung vorkommt, die Regrediente mit Begründung aufgelistet.

Demotion. Die Demotion-Entscheidung löst einen Threat durch Anordnen auf. Das Resultat der Regression von $s_1 \prec s_2$ ist $True$, da dieser Constraint durch die Demotion-Entscheidung hinzugefügt wird (Fall 1). Nach Fall 2 bleiben alle Ordnungen unverändert, die weder mit s_1 noch mit s_2 zusammenhängen. Sind s_1 und s_2 betroffen, werden alle Schritte, die vor s_1 liegen, vor allen Schritten angeordnet, die hinter s_2 liegen. Mit Fall 3 ergibt sich aus $[(s' \prec s_1) \wedge (s_2 \prec s'')] \wedge (s_1 \prec s_2)$ die Regrediente $(s' \prec s'')$. Das Gesamtergebnis ist die disjunktive Verknüpfung der Einzelresultate: $[(s' \prec s_1) \wedge (s_2 \prec s'')] \vee (s' \prec s'')$.

Promotion. Die Promotion-Entscheidung verläuft analog zur Demotion-Entscheidung. Die eingefügte Ordnung $(s_3 \prec s_1)$ führt zur Regrediente: $[(s'' \prec s_3) \wedge (s_1 \prec s')] \vee (s'' \prec s')$.

Step Addition (Establishment). Ziel dieser Entscheidung ist die Erfüllung der offenen Vorbedingung p' eines Schrittes s_d durch Einfügen eines Schrittes s_n mit Effekt p''. Dadurch werden die Mengen der Schritte, der Causal Links, der Ordnungen, Bindungen, Vorbedingungen und Effekte vergrößert. Die Regression einer offenen Vorbedingung $q'@s'$ ergibt $True$, wenn s' der neu eingeführte Schritt ist. Gleiches gilt für Operatoreffekte. Ist q' ein Effekt von s', dann existierte $s' \mapsto q'$ vor der Einführung des Schrittes nicht im Plan. Ein Causal Link $s' \xrightarrow{q} s''$ wird ebenfalls zu $True$ regrediert, wenn er Folge der StepAddition-Entscheidung ist. Ansonsten bleibt er unverändert. Die Menge der Ordnungen wird hier sogar um zwei Elemente erweitert. Zum einen wird die Ordnung eingeführt, die den neuen Schritt vor den Schritt anordnet, dessen Vorbedingungen erfüllt werden soll; zum anderen wird eine spezielle Ordnung hinzugefügt, die sicherstellt, daß ein neuer Schritt immer nach dem initialen Schritt angeordnet ist (vgl. Ordnungskonsistenz, Abschnitt 2.1.2). Dies spiegelt sich im Prädikat $isStart(s)$ wieder, das erfüllt ist, wenn s der Start-Schritt ist. In Abschnitt 4.5 wird dieses Prädikat für die Generalisierung benötigt, um die besondere Bedeutung des Start-Schrittes hervorzuheben. Zur Menge der Bindungen werden diejenigen Constraints hinzugefügt, die sicherstellen, daß p' und p'' notwendigerweise übereinstimmen, und die Constraints der Operatordefinition. Nimmt man den Operator $PutOn(x, y, z)$ aus der Blocksworld-Domäne von CAPLAN, so müssen folgende Constraints erfüllt sein: $IsOfType(Block, x)$,$NotSame(x, y)$,$NotSame(x, z)$ und $NotSame(y, z)$. Sie alle müssen zur Menge der Bindungen hinzugefügt werden. Werden die Constraints über die StepAddition-Entscheidung regrediert, ergibt sich als Regrediente $True$, da diese Constraints vor Einführung des Operators nicht Bestandteil des Plans waren.

Phantomisierung (Simple Establishment). Eine Phantomisierungsentscheidung fügt im Gegensatz zur StepAddition-Entscheidung keinen neuen Schritt ein, sondern nutzt eine bereits existierend Quelle s_n mit Effekt p'' für die Erfüllung der Vorbedingung p' aus. Die Mengen der Causal Links, Ordnungen und Bindungen wird wie bei der StepAddition-Entscheidung erweitert. Es kommen jedoch keine neuen Effekte und Vorbedingungen hinzu.

4.4 Propagierung von Fehlererklärungen und Regelerzeugung

Im letzten Abschnitt wurde definiert, wie eine Fehlererklärung über eine Entscheidung rückwärts propagiert wird. Nun soll geklärt werden, wie die regredierten Erklärungen zusammengesetzt und im Suchbaum propagiert werden müssen, um zu Erklärungen für höhergelegene Knoten im Suchbaum zu gelangen. Der Abschnitt schließt mit der Erzeugung von Kontrollregeln aus den gewonnenen Erklärungen.

4.4.1 Erklärungsverknüpfung

Betrachten wir den Ausschnitt aus einem Suchbaum in Abbildung 4.6, in dem ein Knoten m einen Sohn n hat, welcher wiederum zu zwei fehlgeschlagenen Knoten n_1 und n_2 führt. Da beide Knoten fehlgeschlagen sind, sollte der Planer in Zukunft n erst gar nicht erzeugen, er sollte demnach die dazu führende Entscheidung d nicht treffen. Zu diesem Zweck müssen

Entscheidung: Demote($s_1 \mapsto \neg p''$, $s_2 \xrightarrow{p'} s_3$)
Vorbedingungen: $s_2 \xrightarrow{p'} s_3 \in \mathcal{L}$
$s_1 \mapsto \neg p'' \in \mathcal{E}$
$mgu(p', p'') \in \mathcal{B}$
$(s_1 \prec s_2), (s_3 \prec s_1) \notin \mathcal{O}$
Effekte: $\mathcal{O} \leftarrow \mathcal{O} + (s_1 \prec s_2)$

Constraint	Regrediente	Begründung
$(s' \prec s'')$	$True$	falls $(s' \approx s_1) \wedge (s'' \approx s_2)$ (Fall 1)
	$(s' \prec s'')$	(Fall 2)
	$[(s' \prec s_1) \wedge (s_2 \prec s'')]$	(Fall 3)

Abbildung 4.3: Ergebnis der Regression über die Demotion-Entscheidung

Entscheidung: StepAddition($s_n \mapsto p''$,$p'@s_d$)
Vorbedingungen: $p'@s_d \in \mathcal{C}$
$s_n \xrightarrow{p'} s_d \notin \mathcal{L}$
$p'' \in$ Effekte von s_n
Effekte: $\mathcal{S} \leftarrow \mathcal{S} + s_n$
$\mathcal{L} \leftarrow \mathcal{L} + s_n \xrightarrow{p'} s_d$
$\mathcal{O} \leftarrow \mathcal{O} + (s_n \prec s_d) + \{(s_0 \prec s_n) | isStart(s_0)\}$
$\mathcal{B} \leftarrow \mathcal{B} + mgu(p', p'') +$ zugehörige Bindungen von s_n
$\mathcal{C} \leftarrow \mathcal{C} + \{p@s_n | p \in$ Vorbedingungen von $s_n\}$

Constraint	Regrediente	Begründung
$q'@s'$	$True$ $q'@s'$	falls $s' \approx s_n$ sonst
$s' \mapsto q'$	$True$ $s' \mapsto q'$	falls $s' \approx s_n$ sonst
$s' \xrightarrow{q} s''$	$True$ $s' \xrightarrow{q} s''$	falls $s' \approx s_n$ sonst
$(s' \prec s'')$	$True$ $[(s' \prec s_n) \wedge (s_d \prec s'')] \vee$ $(s' \prec s'')$	falls $s' \approx s_n \wedge s'' \approx s_d$ falls $s' \not\approx s_n \wedge s'' \not\approx s_d$ sonst
$(s \prec s_n)$	$isStart(s) \vee$ $(s \prec s_n)$	falls $s = s_0$ sonst
$x \approx y$	$True$ $(x \approx u) \wedge (y \approx v)$ $x \approx y$	falls $x \approx y \in mgu(p', p'') \cup bindings(s_n)$ falls $u \approx v \in mgu(p', p'') \cup bindings(s_n)$ sonst
$x \not\approx y$	$True$ $x \not\approx y$	falls $x \not\approx y \in bindings(s_n)$ sonst

Abbildung 4.4: Ergebnis der Regression über die SteppAddition-Entscheidung

Entscheidung:	Phantomize($s_n \mapsto p''$,$p'@s_d$)
Vorbedingungen:	$p'@s_d \in \mathcal{C}$
	$s_n \xrightarrow{p'} s_d \notin \mathcal{L}$
	$p'' \in$ Effekte von s_n
Effekte:	$\mathcal{L} \leftarrow \mathcal{L} + s_n \xrightarrow{p'} s_d$
	$\mathcal{O} \leftarrow \mathcal{O} + (s_n \prec s_d)$
	$\mathcal{B} \leftarrow \mathcal{B} + mgu(p', p'')$

Constraint	Regrediente	Begründung
$q'@s'$	$True$ $q'@s'$	falls $s' \approx s_n$ sonst
$s' \mapsto q'$	$True$ $s' \mapsto q'$	falls $s' \approx s_n$ sonst
$s' \xrightarrow{q} s''$	$True$ $s' \xrightarrow{q} s''$	falls $s' \approx s_n$ sonst
$(s' \prec s'')$	$True$ $[(s' \prec s_n) \wedge (s_d \prec s'')]$ $(s' \prec s'')$	falls $s' \approx s_n \wedge s'' \approx s_d$ falls $s' \not\approx s_n \wedge s'' \not\approx s_d$ sonst
$(s \prec s_n)$	$isStart(s) \vee$ $(s \prec s_n)$	falls $s = s_0$ sonst
$x \approx y$	$True$ $(x \approx u) \wedge (y \approx v)$ $x \approx y$	falls $x \approx y \in mgu(p', p'')$ falls $u \approx v \in mgu(p', p'')$ sonst

Abbildung 4.5: Ergebnis der Regression über die Phantom-Entscheidung

wir die Constraints im partiellen Plan P_m am Knoten m kennen, die für den Fehlschlag verantwortlich sind. Für die Fehlererklärung am Knoten m regredieren wir die Fehlererklärung am Knoten n über die Entscheidung d, wie bereits in Abschnitt 4.3.3 beschrieben. Dazu wiederum müssen wir zuerst die Fehlererklärung für den Knoten n bestimmen.

Wir benötigen im folgenden den Begriff *Flaw* $\mathcal{F}$ [Kambhampati *et al.*, 1995a], der für ein offenes Ziel oder einen Threat und somit für eine Unvollständigkeit des aktuellen Plans steht. In Sinne von CAPLAN und REDUX ist dies ein offenes (nicht reduziertes) Ziel (vgl. Abschnitt 2.1.3).

Seien $\mathcal{E}_1$ bzw. $\mathcal{E}_2$ die Fehlererklärungen für die Knoten n_1 bzw. n_2. Desweiteren seien diese Knoten vom Knoten n generiert worden, um den Flaw $\mathcal{F}$ durch die an dieser Stelle einzig möglichen Entscheidungen d_1 bzw. d_2 zu bearbeiten. Die Regredienten von $\mathcal{E}_1$ bzw. $\mathcal{E}_2$ über d_1 bzw. d_2 seien $\mathcal{E}'_1$ bzw. $\mathcal{E}'_2$.

Solange also F am Knoten n existiert, werden die Entscheidungen d_1 bzw. d_2 gefällt und führen zu einem Fehlschlag. Deshalb ergibt sich als Fehlererklärung für den Knoten n:

$$\mathcal{E}(n) = \text{„}\textit{Constraints, die F beschreiben}\text{“} \wedge Regress(\mathcal{E}_1, d_1) \wedge Regress(\mathcal{E}_2, d_2)$$

Als allgemeine *Propagierungsregel* für eine Fehlererklärung am Knoten n mit Planunvollständigkeit $\mathcal{F}$ und m Suchpfaden, die mit den Entscheidungen $d_1, \ldots, d_m$, den Suchknoten $n_1, \ldots, n_m$ und den Fehlererklärungen $\mathcal{E}_1, \ldots, \mathcal{E}_m$ korrespondieren:

$$\mathcal{E}(n) = \text{„}\textit{Constraints, die F beschreiben}\text{“} \bigwedge_{n_i} Regress(\mathcal{E}_i, d_i)$$

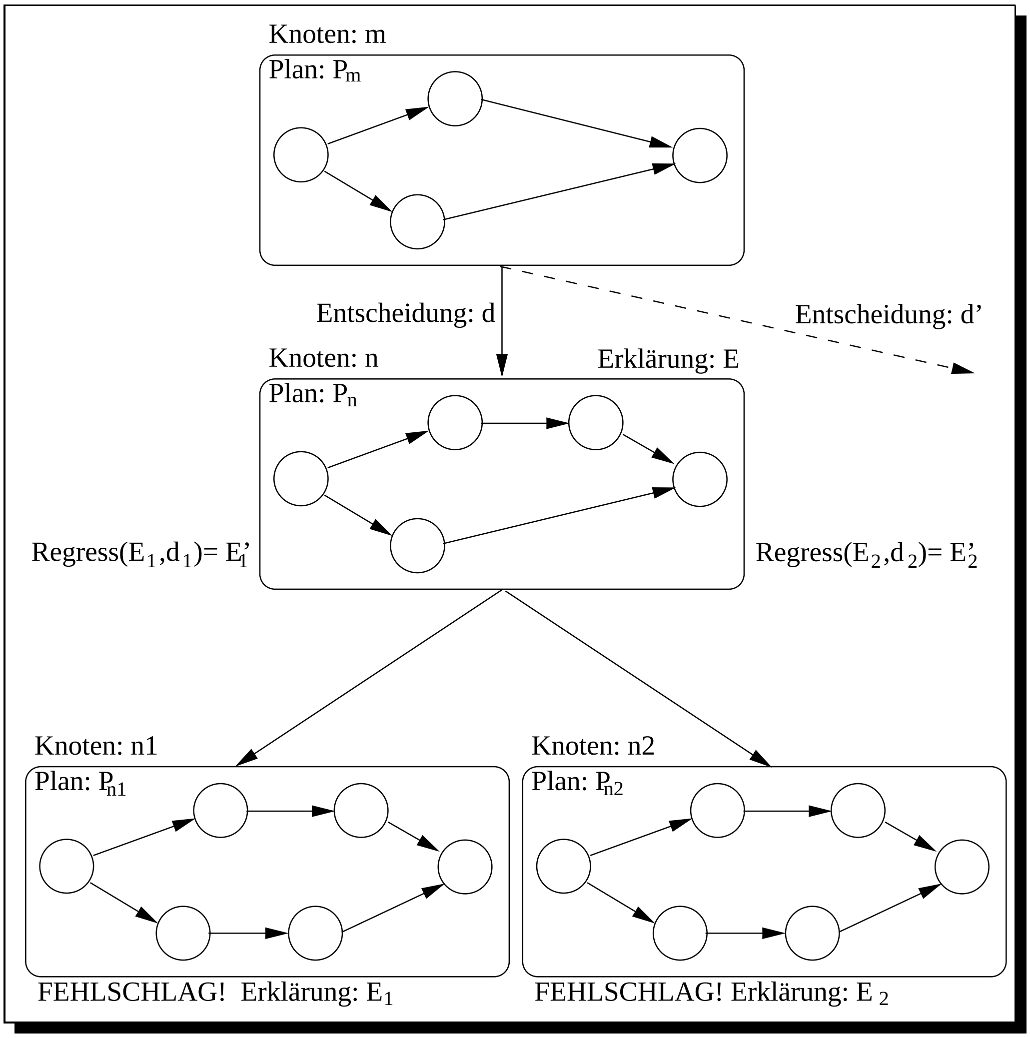

Abbildung 4.6: Beispiel für die Propagierung einer Fehlererklärung

Bemerkenswert ist, daß hier die Fehlererklärungen der Äste mit der Planunvollständigkeit $\mathcal{F}$ verknüpft werden und nicht mit den Vorbedingungen der Entscheidungen, die diese Unvollständigkeiten zu beseitigen versuchten. Dies ist deswegen sinnvoll, da, solange $\mathcal{F}$ im Plan existiert, ansonsten die Entscheidung getroffen wird und damit zu einem Fehlschlag führt.

Diese Methode führt auch zu einer allgemeineren Fehlererklärung als eine Vereinigung der Vorbedingungen der Entscheidung mit den Fehlererklärungen der Äste, da die Vorbedingungen der jeweiligen Entscheidungen Obermengen der Beschreibung der Planunvollständigkeit sind.

In Abbildung 4.6 kann die Fehlererklärung $\mathcal{E}$, die am Knoten n berechnet wurde, über die Entscheidung d regrediert werden und resultiert in der Menge der Constraints, unter welchen die Entscheidung d notwendigerweise zu einem Fehlschlag von m führt.

4.4.2 Vermeiden überspezifizierter Fehlererklärungen

Manchmal erzeugt der oben beschriebene Propagierungsprozeß zu spezifische Fehlererklärungen. Dies läßt sich gut an folgendem Beispiel in Abbildung 4.7 verdeutlichen. Beide

Nachfolger von Knoten B können die offene Vorbedingung $Q_2@O_1$ nicht erfüllen, da kein Domänenoperator mit entsprechendem Effekt zur Verfügung steht. Auch in der Startsituation findet sich kein geeignetes Prädikat. Die initialen Erklärungen $\mathcal{E}_1$ und $\mathcal{E}_2$ werden nun

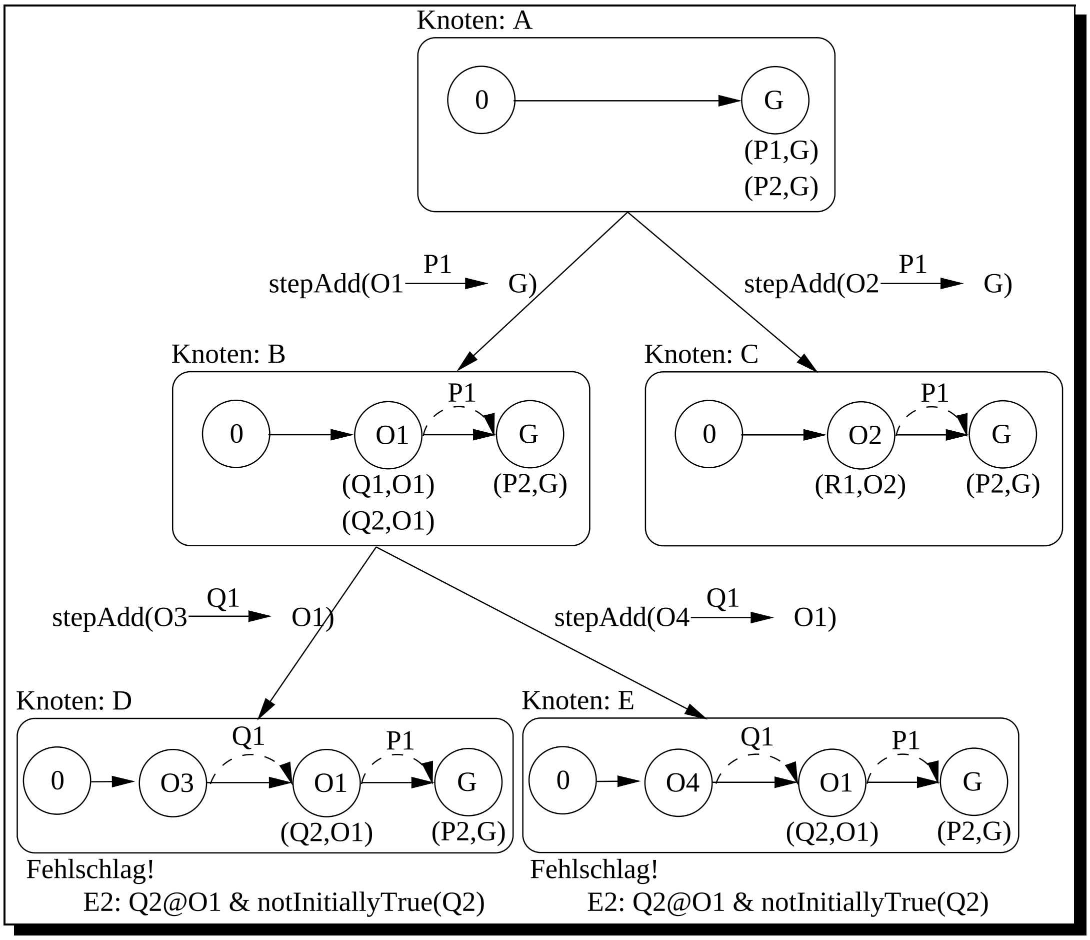

Abbildung 4.7: Beispiel für überspezifizierte Erklärungen

erzeugt, wie in Abschnitt 4.2 erläutert. Mit der Propagierungsregel ergibt sich nun:

$$
\begin{aligned}
\mathcal{E}(B) &= Q_1@O_1 \\
&\wedge \quad Regress(\mathcal{E}_1, stepAdd(O_3 \mapsto Q_1, Q_1@O_1)) \\
&\wedge \quad Regress(\mathcal{E}_2, stepAdd(O_4 \mapsto Q_1, Q_1@O_1)) \\
&\quad\;\; (\text{mit } \mathcal{E}_1 = \mathcal{E}_2 = Q_2@O_1 \wedge notInitiallyTrue(Q_2)) \\
&= Q_1@O_1 \wedge Q_2@O_1 \wedge notInitiallyTrue(Q_2)
\end{aligned}
$$

Auf dieses Weise wird am Knoten B der Constraint $Q_1@O_1$ in die Erklärung aufgenommen. Da der Plan am Knoten B immer fehlschlägt, solange $Q_2@O_1$ im Plan enthalten ist und Q_2 nicht in der initialen Situation zur Verfügung steht, ist $Q_1@O_1$ in der Fehlererklärung überflüssig. Die Erklärung wird *überspezifiziert*. Wird dieser Constraint weggelassen, dann ist die Fehlererklärung am Knoten B gleich der Fehlererklärung am Knoten D, da sich die Fehlererklärung bei der Regression über den StepAddition-Operator, der vom Knoten B zum Knoten D führt, nicht ändert.

Die Propagierungsmethode muß aus diesem Grund dahingehend abgeändert werden, daß sie diesem Sachverhalt auch formal gerecht wird. Ändert die Regression die Fehlererklärung am Knoten n nicht, so ist die vollständige Fehlererklärung am Vaterknoten von n die gleiche wie am Knoten n selbst.

$$\begin{array}{lcll} \mathcal{E}(n) & = & \mathcal{E}_1 & \text{wenn } Regress(\mathcal{E}_1, d_1) = \mathcal{E}_1 \\ & = & \textit{Constraints, die } \mathcal{F} \textit{ beschreiben} & \\ & & \wedge Regress(\mathcal{E}_1, d_1) & \\ & & \wedge Regress(\mathcal{E}_2, d_2) & \text{sonst} \end{array}$$

Für die Berechnung der Fehlererklärung am Knoten n mit Planunvollständigkeit $\mathcal{F}$ und m Suchpfaden zur Lösung von $\mathcal{F}$, sowie den entsprechenden Entscheidungen $d_1, d_2 \dots d_m$, den daraus resultierenden Suchknoten $n_1, n_2 \dots d_n$ und den Fehlererklärungen $\mathcal{E}_1, \mathcal{E}_2 \dots \mathcal{E}_m$ gilt:

$$\begin{array}{lcll} \mathcal{E}(n) & = & \mathcal{E}_i & \text{wenn } Regress(\mathcal{E}_i, d_i) = \mathcal{E}(n_i) \\ & = & \textit{Constraints, die } \mathcal{F} \textit{ beschreiben} & \\ & & \bigwedge_{\forall n_i} Regress(\mathcal{E}_i, d_i) & \text{sonst} \end{array}$$

Die einzelnen Erklärungen der Knoten n_i werden nicht zusammengefaßt, wenn die Regression über die Entscheidung d_i nicht die Erklärung $\mathcal{E}_i$ verändert, da dies impliziert, daß $\mathcal{E}_i$ bereits am Knoten n im partiellen Plan existiert. Das wiederum bedeutet, daß der partielle Plan am Knoten n bereits inkonsistent ist und $\mathcal{E}_i$ eine korrekte Fehlererklärung für n ist. Daraus folgt, daß n nicht weiter zu einer Lösung verfeinert werden kann und so kein Gewinn damit erzielt wird, weitere Verfeinerungen von n zu verfolgen.

4.4.3 Korrektheit gelernter Regeln

SNLP+EBL generiert Suchkontrollregeln, während das System Erklärungen im Suchbaum hinaufpropagiert. Da Fehlererklärungen betrachtet werden, lernt das System nur Zurückweisungsregeln. Die einfachste Regel besteht darin, eine partiellen Plan zu verwerfen, nachdem er erzeugt wurde. Angenommen SNLP+EBL erreiche den Knoten n und generiere die Fehlererklärung $\mathcal{E}$. Eine Art der Kontrollregel, eine *Beschneideregel*, könnte folgende Form haben:

Wenn $\mathcal{E}$ im partiellen Plan gilt,
dann weise den Plan **zurück**.

Eine sehr ähnliche Klasse von Regeln weist Entscheidungen zurück, bevor sie getroffen werden. Sei $\mathcal{E}'$ das Resultat einer Regression der Erklärung $\mathcal{E}$ über die Entscheidung d. Daraus läßt sich folgende Zurückweisungsregel herleiten:

Wenn $\mathcal{E}'$ im partiellen Plan gilt,
dann weise die Entscheidung d **zurück**.

CAPLAN lernt nur Zurückweisungsregeln für Entscheidungen (vgl. Kapitel 5). Dies hängt mit der Abhängigkeitsverwaltung zusammen, die keine kompletten Pläne, sondern Beziehungen zwischen Entscheidungen verwaltet.

Korrektheit einer Zurückweisungsregel. Sei $\mathcal{P}$ ein partieller Plan, der durch eine Zurückweisungsregel abgelehnt wird, oder sei die Entscheidung d, die zu $\mathcal{P}$ führen würde, von dieser Regel zurückgewiesen worden. Dann heißt die Zurückweisungsregel *korrekt*, wenn für jeden solchen partiellen Plan $\mathcal{P}$ gilt, daß $\mathcal{P}$ nicht zu einer Lösung des Planungsproblems verfeinert werden kann.

Behauptung: Wenn SNLP+EBL eine Zurückweisungsregel lernt, dann ist sie im beschriebenen Sinne korrekt, d. h. $\mathcal{P}$ kann an dieser Stelle nicht mehr zu einer Lösung führen.

Beweisidee: Eine korrekte Regel muß garantieren, daß sie keine Lösung des Suchraums wegschneidet. Für initiale Erklärungen und daraus erzeugte Regeln ist die Korrektheit unmittelbar ersichtlich. Denn schlägt eine Entscheidung fehl, wird eine initiale Erklärung generiert, die aus einer inkonsistenten Constraint-Menge besteht. Solange diese Constraints Bestandteil des Plans sind, kann der Plan nicht zu einer Lösung vervollständigt werden.
Für Knoten im Innern des Suchbaums gestaltet sich der Beweis schwieriger, da hier sichergestellt sein muß, daß die Erklärung *alle* Fehlschläge der Nachfolgerknoten umfaßt. Diesen Nachweis führt [Kambhampati *et al.*, 1995a] über eine Klassifizierung der Knoten. Die Threat-Auflösungsknoten vereinen nur Entscheidungen des Typs Demotion und Promotion unter sich. Establishment-Knoten haben nur Phantomisierungs- und Step-Addition-Entscheidungen als Nachfolger.
Threat-Auflösungsknoten haben nur die beiden genannten Entscheidungsmöglichkeiten. Auch für die Einführung eines neuen Schrittes in den Plan existieren nur endlich viele Alternativen, da die Anzahl der Operatoren einer Domäne feststeht. Nur die Anzahl der Phantomisierungsmöglichkeiten ist variabel und hängt von der Anzahl der Planschritte ab. Die Phantomisierungsalternativen können nochmals bezüglich der Quelle für die Phantomisierung unterschieden werden. Die Phantomisierung durch den Startzustand wird durch Einführung des Prädikates $notInitiallyTrue()$ gekennzeichnet und reicht bei geänderter Problemstellung für die korrekte Anwendbarkeit einer Regel aus. Die Phantomisierung durch einen anderen Planschritt muß nicht in Bezug auf die Korrektheit betrachtet werde, da der Planer alle Step-Addition-Möglichkeiten und Erklärungen für einen Fehlschlag in diesen Ästen des Suchbaums die Erklärungen für den fehlgeschlagene Phantomisierungen subsumieren.
(Für einen vollständigen Beweis siehe [Kambhampati *et al.*, 1995a])

4.4.4 Regelerzeugung

Wenden wir uns nun der eigentlichen Konstruktion von Regeln zu. Abbildung 4.8 zeigt einen Knoten des Suchbaums mit zwei Schritten, sprich zwei Operatoren. Zwischen dem initialen Schritt 0 und s_2 existiert ein Causal Link $0 \stackrel{On(a,b)}{\longrightarrow} s_2$, der durch s_1 bedroht wird. Versucht der Planer nun, mittels einer Demotion-Entscheidung s_1 vor 0 anzuordnen, dann wäre der nachfolgende Plan inkonsistent und müßte zurückgewiesen werden. Es ergibt sich nach der Generierung einer initialen Erklärung für den Knoten n eine Regel, die den Nachfolgerknoten zurückweist:

Wenn $(0 < s_1) \wedge (s_1 < 0)$ im Plan gilt,
dann weise den Plan **zurück**.

Diese Regel sagt also, daß ein Plan, der eine zyklische Ordnung enthält, zurückgewiesen wird.

Der Planer regrediert die Erklärung über die Demotion-Entscheidung, um die Fehlererklärung für den Suchpfad zum Knoten n zu erzeugen, und bildet folgende Zurückweisungsregel für

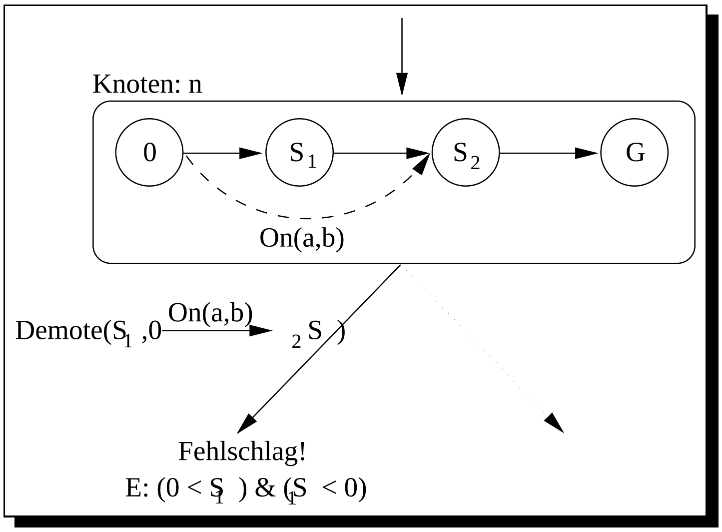

Abbildung 4.8: Eine fehlgeschlagene Demotion-Entscheidung

die Entscheidung:

Wenn $(0 < s_1)$ im Plan gilt,
dann weise die Entscheidung $Demote(s_1, 0 \stackrel{On(a,b)}{\longrightarrow} s_2)$ **zurück**.

Der Nutzen der beiden gezeigten Regeln ist nicht sehr hoch, da der Planer die Konsistenz sowieso geprüft hätte. Besonders die Kosten, den Bedingungsteil der Regeln mit dem aktuellen Plan zu vergleichen, können hier leicht die Suchersparnis übertreffen.

Regeln, die von Zwischenknoten eines Suchbaumes erlernt werden, sind oft sehr viel nützlicher.

4.5 Generalisierung von Regeln

Betrachten wir noch einmal die Zurückweisungsregel zur *demotion*-Entscheidung aus Abschnitt 4.4.4. Darin wird die Erzeugung eines Causal Link mit dem Prädikat $On(a, b)$, unter Beteiligung der Schritte $0, s_1, s_2$ verworfen.

Was geschieht aber, wenn statt der Objekte a, und b die Objekte c und d vorliegen?

Da die Regel für diesen Fall zu speziell ist, kann sie nicht angewandt werden und ist in dieser Form nur in wenigen Planungssituationen von Nutzen. Die Regel muß verallgemeinert werden, indem Konstanten durch Variablen ersetzt werden.

Die Ersetzung aller Konstanten einer Regel durch Variable erhält aber nicht unbedingt die Korrektheit, da u. U. bestimmte Konstanten (Objekte, Schritte) in der Regel verbleiben müssen, damit ein Fehler im Plan auftritt.

Folgendes Blocksworld-Problem soll dies verdeutlichen:

Im Startzustand des Problems gelte *Clear(Table)* und es sei das Ziel $\neg$*Clear(Table)* zu erreichen. SNLP+EBL schlägt nun fehl, da immer *Clear(Table)* gilt, und erzeugt die Fehlererklärung

$$\neg Clear(Table)@G \wedge \texttt{initiallyTrue}(Clear(Table))$$

sowie die Kontrollregel

Wenn $\neg Clear(Table)@G \wedge \texttt{initiallyTrue}(Clear(Table))$
dann weise den Plan **zurück**.

Es ist sofort ersichtlich, daß der spezifische Schritt G durch eine Variable ersetzt werden kann, da der Planer solange fehlschlagen wird, wie die Vorbedingung $\neg Clear(Table)$ als Vorbedingung eines beliebigen Schrittes im Plan enthalten ist. Die Konstante *Table* kann wiederum nicht zu einer Variable x generalisiert werden, da dann $Clear(x)$ mit beispielweise $Clear(a)$ unifiziert werden kann, wobei a ein Block sei. Hier würde der Planer dann irrtümlich einen Plan mit der Vorbedingung $Clear(a)@G$ zurückweisen. Die Vollständigkeit des Planers ginge verloren.

Auch die Namen der Schritte in den Fehlererklärungen müssen generalisiert werden. Da die Erklärungen Schritte durch Effekte und Bedingungen, sowie Relationen zwischen den Schritten qualifizieren, können die Schritte in den meisten Fällen durch Variablen ersetzt werden. Abhängig von der Situation gilt nur der Startschritt als Ausnahme und muß dann gesondert behandelt werden.

Im folgenden wird die Version des Generalisierungsprozesses von [Kambhampati *et al.*, 1995a] vorgestellt. Im Gegensatz zu Mintons Verfahren (erklärungsbasierte Spezialisierung der Zielkonzepte, [Minton, 1988]) ist dieses einschrittig. Es muß für das Zielkonzept nicht erst einen Beweisbaum erzeugen, da dieser bereits Teil des Suchbaums von SNLP+EBL ist, um dann daraus mittels neuer Operatorschema-Instanzen die schwächsten Vorbedingungen für ein variablisiertes Zielkonzept zu berechnen.

Puton(x von y auf z)	
Vorbedingungen:	Clear(z), Clear(x), On(x,y)
Effekte:	$\neg$On(x,y), Clear(y),
	$\neg$Clear(z), On(x,z)
Bindungen:	NotSame(z,*Table*), NotSame(x,z)

Abbildung 4.9: Namen-sensitive Version des PutOn-Operators

Puton(x von y auf z)	
Vorbedingungen:	Clear(z), Clear(x), $\neg$IsTable(z)
Effekte:	$\neg$On(x,y), Clear(y),
	$\neg$Clear(z), On(x,z)
Bindungen:	NotSame(x,z)

Abbildung 4.10: Namen-insensitive Version des PutOn-Operators

[Kambhampati *et al.*, 1995a] unterteilt die Domänentheorien in zwei Kategorien:

- Namen-sensitive und
- Namen-insensitive Theorien.

Namen-sensitive Theorien enthalten Operatoren, die auf Objekte mit Namen zugreifen, d. h. bereits in den Operatorschemata werden ausgezeichnete Objekte verwendet. *Namen-insensitive* Theorien verwenden Prädikate, um ein Objekt zu qualifizieren.

Namen-insensitive Theorien erleichtern die Variablisierung von Regeln erheblich. In solcherart definierten Theorien kann ein Operator jederzeit ein Objekt durch dessen Eigenschaften qualifizieren. Als Beispiel für die beiden Definitionsmöglichkeiten diene der *PutOn*-Operator in den Abbildungen 4.9 und 4.10. Die erste Abbildung zeigt *Table* als spezielle Konstante, während die zweite Definition das Objekt x über die Vorbedingung $\neg$*IsTable(x)* qualifiziert.

Diese Definitionen zeigen auch, daß sich eine sensitive Domänenbeschreibung leicht in eine äquivalente insensitive Theorie tranformieren läßt, indem die Vorbedingungen der Operatoren durch geeignete einstellige Prädikate, die die entsprechende Eigenschaft bezeichnen sollen, erweitert werden. Da diese einfache Transformationsmöglichkeit gegeben ist und CA-PLAN namen-insensitive Domänenbeschreibungen verwendet, wird im folgenden nur auf die diesbezüglichen Besonderheiten der Generalisierung eingegangen.

Der Vorteil namen-insensitiver Theorien liegt darin, daß jedes spezifische Objekt in einer berechneten Fehlererklärung vollständig durch die Eigenschaften des Objektes qualifiziert wird. Dadurch kann in der Generalisierungsphase jedes Objekt variablisiert werden.

Wird in obigem Beispielproblem mit dem Ziel $\neg$*Clear(Table)@G* der namen-insensitive Operator aus Abbildung 4.10 verwendet, dann kommt der Planer zum Schluß, daß der Fehler aufgrund der fehlenden Establishment-Möglichkeit für $\neg$*IsTable(Table)* auftritt. Die Fehlererklärung hierfür lautet:

$$\neg \mathit{IsTable(Table)@Puton(x,y,Table)} \wedge \mathtt{initiallyTrue}\mathit{(IsTable(Table))}$$

Bei der Regression über die Entscheidung, die den *Puton*-Operator zum Plan hinzufügte, entsteht daraus zusammen mit der Zielbeschreibung folgende Fehlererklärung:

$$\neg \mathit{Clear(Table)@G} \wedge \mathtt{initiallyTrue}\mathit{(IsTable(Table))}$$

An dieser Stelle kann nun gefahrlos jede Konstante durch Variablen ersetzt werden.

$$\neg \mathit{Clear(x)@G} \wedge \mathtt{initiallyTrue}\mathit{(IsTable(x))}$$

Diese Erklärung ist korrekt, da kein Objekt vom Typ *Block* mit x unifiziert werden kann.

Die bisherigen Überlegungen galten nur Objektnamen. Für Schritte erkennt man sofort, daß Operatoren vollständig durch ihre Vorbedingungen qualifiziert werden. Die einzige Planungsentscheidung, die einen Schritt beim Namen nennt, ist die StepAddition-Entscheidung, die einen neuen Schritt in den Plan einfügt. In Abbildung 4.4 wurde das einstellige Prädikat $isStart()$ eingefügt, um auch diese Entscheidung namen-insensitiv zu machen.

Wenn die Ordnung $0 \prec s_n$ über einen Establishment-Operator, der einen Schritt s_n in den Plan einfügt, regrediert wird, erhält man das Ergebnis $isStart(0)$. Dieses wird Teil der endgültigen Fehlererklärung, da keine Planungsentscheidung einen Effekt bezüglich des Start-Schrittes hat. Durch das Prädikat $isStart()$ wird die besondere Rolle des Start-Schrittes in der endgültigen Erklärung herausgestellt. Hiermit wird garantiert, daß bei vollständiger Variablisierung der Schrittnamen auch die besondere Rolle des Start-Schrittes berücksichtigt wird.

4.6 Suchtiefenüberschreitungsfehler

In Abschnitt 4.2 wurden Suchtiefenüberschreitungen bereits kurz angesprochen. An dieser Stelle soll eine Möglichkeit vorgestellt werden, die in einem solchen Fall erklärt, warum diese Überschreitung einen Fehler darstellt. Im Rahmen der hier vorliegenden Arbeit wurde dieses Verfahren jedoch nicht realisiert.

Der Grund für die Überschreitung der Suchtiefe liegt oft darin, daß der Planer lokal sinnvolle Entscheidungen trifft, diese aber, global gesehen, keinen Beitrag zur Annäherung an den Zielzustand zu leisten vermögen. Manchmal können wir jedoch auf strengere Konsistenzprüfungen zurückgreifen, die auf der Domänentheorie und der Metatheorie des Planers basieren. Damit ist es möglich, bei gegebener Suchtiefe Fehler im Plan aufzudecken, die dem Konsistenztester des Planers zu diesem Zeitpunkt noch verborgen sind.

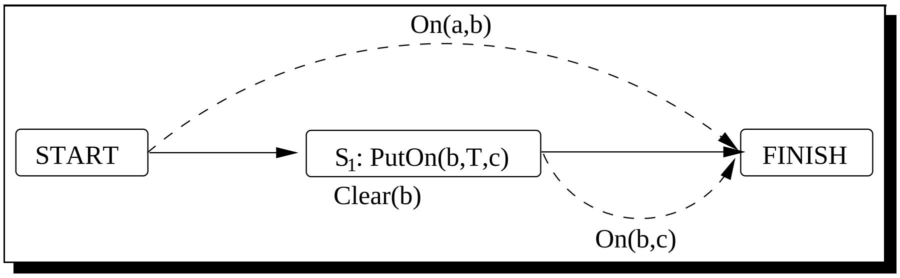

Abbildung 4.11: Beispiel für ein Domänenaxiom und ein Axiom der Metatheorie von SNLP

Ein Beispiel soll dies illustrieren. Sei dazu das Axiom der Blocksworld gegeben, daß kein Block zugleich frei sein kann, während ein anderer Block obenauf liegt. Desweiteren sei die Metatheorie von SNLP mit der Aussage gegeben, daß ein Causal Link $s \xrightarrow{P} t$, sobald er etabliert wurde, die Bedingung P zwischen den Schritten s und t in jedem Fall schützt. Im Plan der Abbildung 4.11 kann deshalb keine Lösung gefunden werden. Darauf aufbauend wird eine Menge von *notwendigen persistenten Bedingungen* ($np - conditions$) für einen Schritt s' im Plan $\mathcal{P}$ definiert (vgl. [Kambhampati *et al.*, 1995a]). Diese Menge enthält alle durch Causal Links geschützten Prädikate, die von s' bedroht werden:

$$np - conditions(s') = \{c \mid s_1 \xrightarrow{c} s_2 \wedge s_1 \prec s' \wedge s' \prec s_2\}$$

Mit den *np-conditions* eines Schrittes wissen wir, daß ein partieller Plan, der diese Bedingungen enthält, nicht vervollständigt werden kann, solange die Vereinigungsmenge aus Vorbedingungen von s', $preconditions(s')$, und $np - conditions(s')$ inkonsistent ist in Bezug auf die Domänenaxiome.[2] Der lokale Konsistenztest erkennt diese Inkosistenz nicht. Dies wiederum kann zu unendlichen Schleifen führen.[3] Im Beispielplan der Abbildung 4.11 versucht SNLP möglicherweise $Clear(b)$ am Schritt s_1 zu etablieren, indem es einen Schritt $s_3 : PutOn(x, b, z)$ einfügt und dann versucht, $On(x, b)$ zu erfüllen,indem es a von b wieder herunternimmt. Überschreitet SNLP dabei die Suchtiefe, dann benutzt SNLP+EBL einen Konsistenztest, der auf den notwendigen persistenten Bedingungen beruht, um den impliziten Fehler zu entdecken und zu erklären.

[2] Das gleiche gilt zwar auch für die Effekte eines Schrittes, doch werden diese Inkonsistenzen bereits durch den Threat-Erkennungsmechanismus entdeckt und aufgelöst.

[3] In CAPLAN wurde das Problem der Teilzielrekursion durch explizite Verwaltung der Teilzielhierarchie behoben. Siehe dazu [Weberskirch, 1994].

Die Domänenaxiome müssen in ihrer Repräsentation eingeschränkt werden, um den Konsistenztest anwendbar zu halten: Jedes Domänenaxiom ist dabei eine Konjunktion von Literalen mit einer Menge von Bindungs-Constraints. Beispielhaft seien folgende Axiome für die Blocksworld genannt. Hierbei formuliert das erste Axiom die oben erwähnte Tatsache, daß kein Block auf y liegen darf, wenn y gleichzeitig frei sein soll, mit der Ausnahme, daß y der Tisch ist.

$$\begin{gathered} On(x,y) \wedge Clear(y) \wedge \neg IsTable(y) \\ On(x,y) \wedge On(x,z) \wedge NotSame(y,z) \\ On(x,y) \wedge On(z,y) \wedge NotSame(x,z) \wedge \neg IsTable(y) \end{gathered}$$

Ein partieller Plan ist somit inkonsistent, sobald er einen Schritt s enthält, für den ein Domänenaxiom mit einer Teilmenge von $np - conditions(s) \cup preconditions(s)$ unifizierbar ist. Mit dieser Theorie ergibt sich als initiale Erklärung für das obige Beispiel:

$$START \stackrel{On(a,b)}{\longrightarrow} FINISH \wedge (START \prec s_1) \wedge (s_1 \prec FINISH) \wedge Clear(b)@s_1 \wedge \neg IsTable(b)$$

Diese Erklärung kann nun wie die in Abschnitt 4.2 vorgestellten initialen Fehlererklärungen zur Regelerzeugung über die Planungsentscheidungen regrediert werden.

4.7 Erweiterte Ansätze

Im folgenden werden zwei Planer kurz vorgestellt, in denen erklärungsbasiertes Lernen ebenfalls eingesetzt wird. Zum einen ist das UCPOP+EBL, ein Planer mit einer mächtigeren Domänenbeschreibungssprache als SNLP, und zum anderen DerSNLP+EBL, ein fallbasierter Planer.

4.7.1 Planer mit erweiterter Domänenbeschreibungssprache

UCPOP von [Penberthy & Weld, 1992] ist wie SNLP ein partiell-ordnender Planer, der einen partiellen Plan durch Hinzufügen von Constraints (Verfeinerung) vervollständigt. Die Menge der Vorbedingungen eines Operators kann *disjunktive*, sowie *all-* und *existenz-quantifizierte Bedingungen* enthalten. Die Effekte dürfen zwar ebenfalls bedingt sein, aber nur Allquantifizierungen enhalten. Durch diese Möglichkeit kann UCPOP viele Domänen kompakter spezifizieren und mehr Merkmale eines Operators explizit machen. Der Suchraum wird dadurch jedoch entsprechend größer.

Durch die ausdrucksstärkeren Vor- und Nachbedingungen der Operatoren muß das Verfahren, das SNLP zur Lösung eines Zieles verwendet, für UCPOP erweitert werden (vgl. [Kambhampati *et al.*, 1995a]).

4.7.2 Erklärungen in Replay-Verfahren

[Ihrig & Kambhampati, 1996] beschreibt den fallbasierten Planer DerSNLP+EBL. Das Grundsystem DerSNLP (derivational replay in SNLP) ist ein partiell-ordnender Planer, der um eine fallbasierte Komponente erweitert wurde. DerSNLP speichert Planherleitungen (plan derivations) und bereitet die Lösung eines neuen Falles dadurch vor, daß das

System die Entscheidungen, die es in einem ähnlichen Fall getroffen hat, im Kontext des neuen Problems wiederholt. Diese Vorgehensweise entspricht dem Verfahren *Derivational Analogy* (siehe [Carbonell, 1986]). Der Planer versucht anschließend den gefundenen Fall so zu vervollständigen, daß er das gestellte Problem löst.

Erklärungsbasiertes Lernen wird in DerSNLP+EBL zur Verbesserung der Fallauswahl eingesetzt. Treten im Zuge der Vervollständigung eines aus der Fallbasis geholten Planes Fehler auf, die zu Backtracking bis in den Fall hinein führen, erzeugt die Erklärungskomponente eine Regel, die bei einem späteren Zugriff auf den Fall wie ein Filter wirkt. Auf diese Weise werden zusätzlich zur grundsätzlichen Fallauswahl von DerSNLP Bedingungen gewonnen, die zu einer weiteren Verbesserung der Fallauswahl führen.

Kapitel 5

Erklärungsbasiertes Lernen in CAPLAN/EBL

Zur Einordnung in das Gesamtsystem werden zunächst die im Rahmen dieser Arbeit entstandenen zusätzlichen Module beschrieben. Das EBL-Verfahren von [Kambhampati *et al.*, 1995a] bedurfte verschiedener Modifikationen, bevor es in CAPLAN realisiert werden konnte. Die Unterschiede zwischen SNLP+EBL und CAPLAN mit seiner EBL-Erweiterung (CAPLAN/EBL) werden im zweiten Abschnitt vorgestellt. Die Erzeugung von Fehlererklärungen und Zurückweisungsregeln schließt sich an. Die Generalisierung von Regeln muß ebenfalls genauer betrachtet werden, da [Kambhampati *et al.*, 1995a] keine Typen kennt. Das Kapitel schließt mit einer Darstellung der Regelverarbeitung.

5.1 Erweiterungen des CAPLAN-Grundsystems

Bevor wir auf das eigentliche Lernen in CAPLAN/EBL und die Regelerzeugung eingehen, sollen die Punkte im CAPLAN-Grundsystem vorgestellt werden, an denen es um neue Komponenten erweitert wurde (Abbildung 5.1):

- Die statische Wissenbasis wurde um Regelbasen erweitert. Zur besseren Verwaltung wurden die Regeln aufgeteilt in die Typen von Entscheidungen, die in CAPLAN zur Verfügung stehen: Phantomisierung, Establishment, Promotion bzw. Demotion und Separation (Abbildung 5.2). So kann auf einfache Weise die Regelauswertung auf einzelne Regeltypen beschränkt werden. Auch die Anwendungshäufigkeit der Regeln ist duch diese Aufteilung besser vergleichbar.

- Zum Domänen- und Problem-Editor kommt nun ein *Regel-Editor* hinzu (Abbildung 5.2). Er dient der Verwaltung der verschiedenen Regelbasen. Der Benutzer kann neue Zurückweisungsregeln einführen, bestehende Regeln modifizieren, löschen, speichern und auch wieder laden.

- Das Herz der EBL-Implementierung ist die zusätzliche Kontrollkomponente für chronologisches Backtracking mit Erklärungsgenerierung *chrontBTWithExpl.* Auf der Basis chronologischen Backtrackings wird hier bei jedem Backtracking-Schritt der EBL-Mechanismus angestoßen. Der *Regelgenerator* erzeugt dann eine entsprechende Zurückweisungsregel anhand der gefundenen Erklärung und speichert sie in der jeweili-

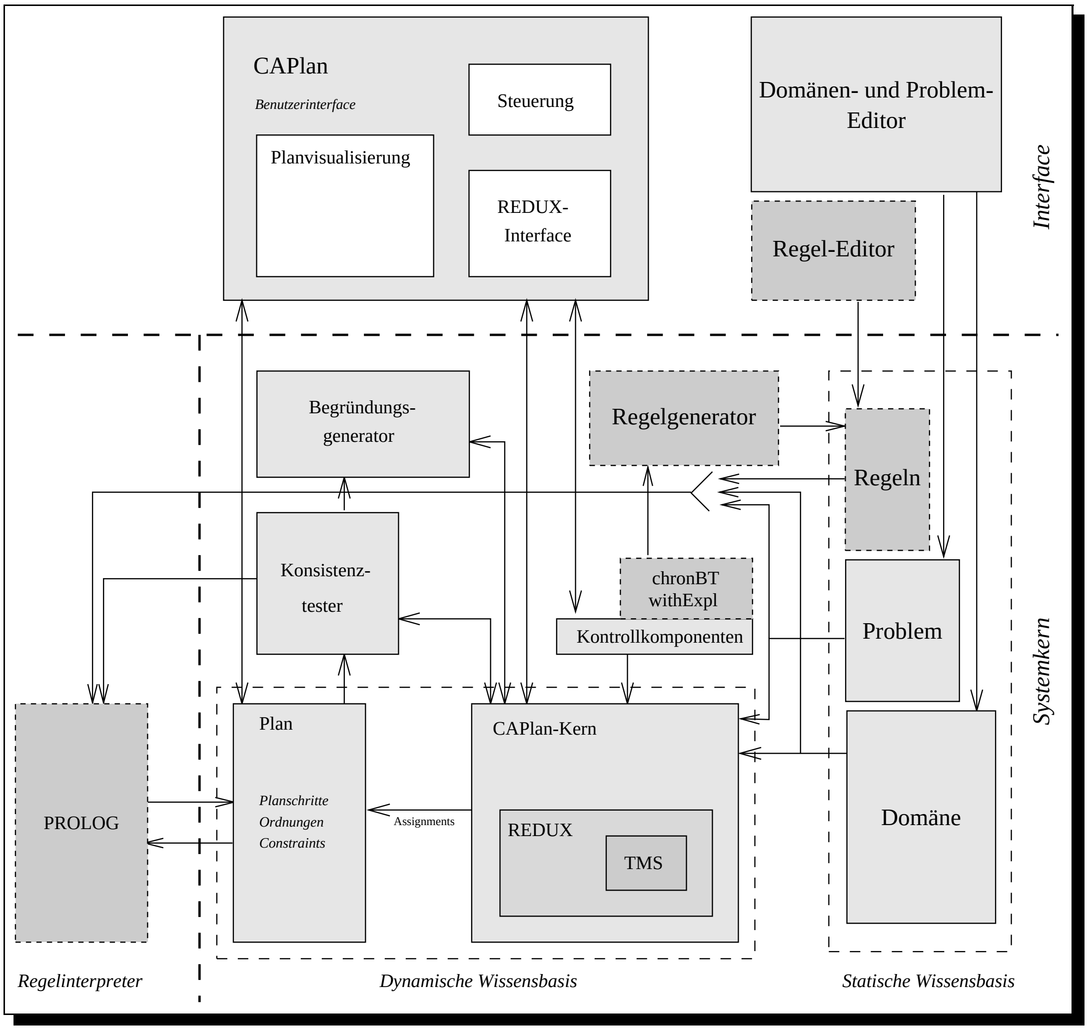

Abbildung 5.1: CAPLAN/EBL

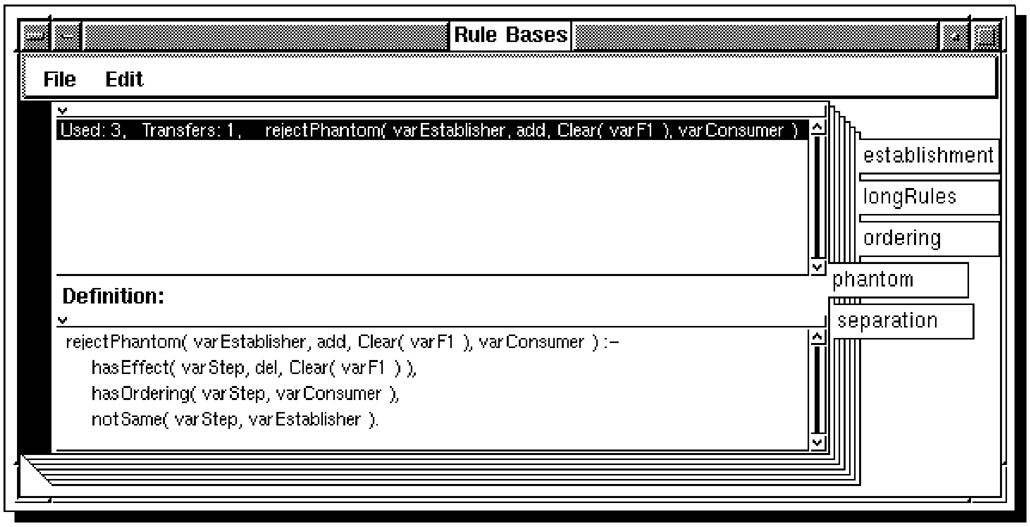

Abbildung 5.2: Der Regeleditor mit einer gelernten Regel für die Blocksworld

gen Regelbasis. Über das Benutzerinterface läßt sich die Lernkomponente beliebig ein- und ausschalten.

- Für die Regelabarbeitung wurde eine existierende Anbindung von SMALLTALK an PROLOG verwendet [Niehaus & Lorscheid, 1995]. PROLOG kann Regeln schnell abarbeiten und sorgt für entsprechend notwendiges Backtracking bei der Verwendung der Regeln. Die Variablenbelegungen können jedoch nur von CAPLAN kommen, so daß zusätzlich ein Protokoll zur Abfrage von Planzuständen entwickelt wurde.

Die Regelauswertung wird vom Konsistenztester bei der Operatorauswahl für die Bearbeitung eines Zieles angestoßen. Bisher ist die Regelauswertung auf Operatoren beschränkt, die vom Konsistenztester als konsistent angesehen werden. Somit werden lokal inkonsistente Operatoren von der Regelverwendung ausgeschlossen.

5.2 Unterschiede zwischen SNLP+EBL und CAPLAN/EBL

Die Hauptunterschiede zwischen SNLP+EBL und CAPLAN/EBL sind in der Struktur des Suchbaums, in der Definition von Threats, in der erweiterten Wahl von Establishern, sowie in typisierten Variablen und in der Abhängigkeitsverwaltung REDUX zu sehen. Diese Punkte sollen im folgenden im einzelnen diskutiert werden.

Struktur des Suchbaums. Der Hauptunterschied zwischen beiden Systemen liegt in der Handhabung von Plänen. SNLP+EBL speichert im Suchbaum an jedem Knoten den zu diesem Zeitpunkt aktuellen Plan (siehe z. B. Abbildung 4.1). Im Laufe eines Planungsprozesses sammelt sich also eine recht große Zahl von Plänen an. Jedoch können somit bei der Regression die jeweiligen Pläne beliebig begutachtet werden. In CAPLAN/EBL wird der Suchbaum zwar ebenfalls von den Entscheidungen aufgespannt, jedoch werden die bearbeiteten Ziele als Knoten gespeichert. Der aktuelle Pfad in diesem Baum findet sich dabei auf dem Goal-Stack wieder und repräsentiert den *einen* aktuellen Plan, den CAPLAN betrachtet. Anhand des Goal-Stack wird die Regression durchgeführt.

Abbildung 5.3 zeigt einen Ausschnitt aus einem Suchbaum und einen aktuellen Suchpfad. Direkt daneben ist der zugehörige Goal-Stack dargestellt. Kann ein Ziel nicht weiter bearbeitet werden, wird ein *Goal-Block-Task* von REDUX ausgelöst, der zu Backtracking führt. Bei der Implementierung wurde chronologisches Backtracking eingesetzt. Durch den Rückzug des blockierten Zieles wird das Lernen aus Fehlschlägen ausgelöst. Während in SNLP+EBL ein fehlgeschlagener Operator Auslöser für die Erzeugung einer initialen Erklärung ist, wird in CAPLAN/EBL erst bei einer Zielblockade mit dem Lernen begonnen. Da für die Bearbeitung eines Ziels nur einmal Variablenbindungen für die möglichen Operatoren vorgenommen werden, die höchstens bei einer Zielblockade aufgehoben werden, ist es vollkommen ausreichend, erst zu diesem Zeitpunkt initiale Erklärungen aufzubauen. Auf diese Weise werden u. U. viele unnötige initiale Erklärungen, die *am Pfad des vollständigen Planes liegen*, erst gar nicht erzeugt.

Definition von Threats. Ein weiterer Unterschied liegt in der Definition von Threats. Für [Kambhampati *et al.*, 1995a] liegt erst dann eine Bedrohung vor, wenn das Prädikat p eines Causal Links $s \xrightarrow{p} t$ notwendigerweise übereinstimmt mit einem Effekt q eines bezüglich s und t nicht angeordneten Schrittes v. Liegt eine notwendige Übereinstimmung vor, dann sind

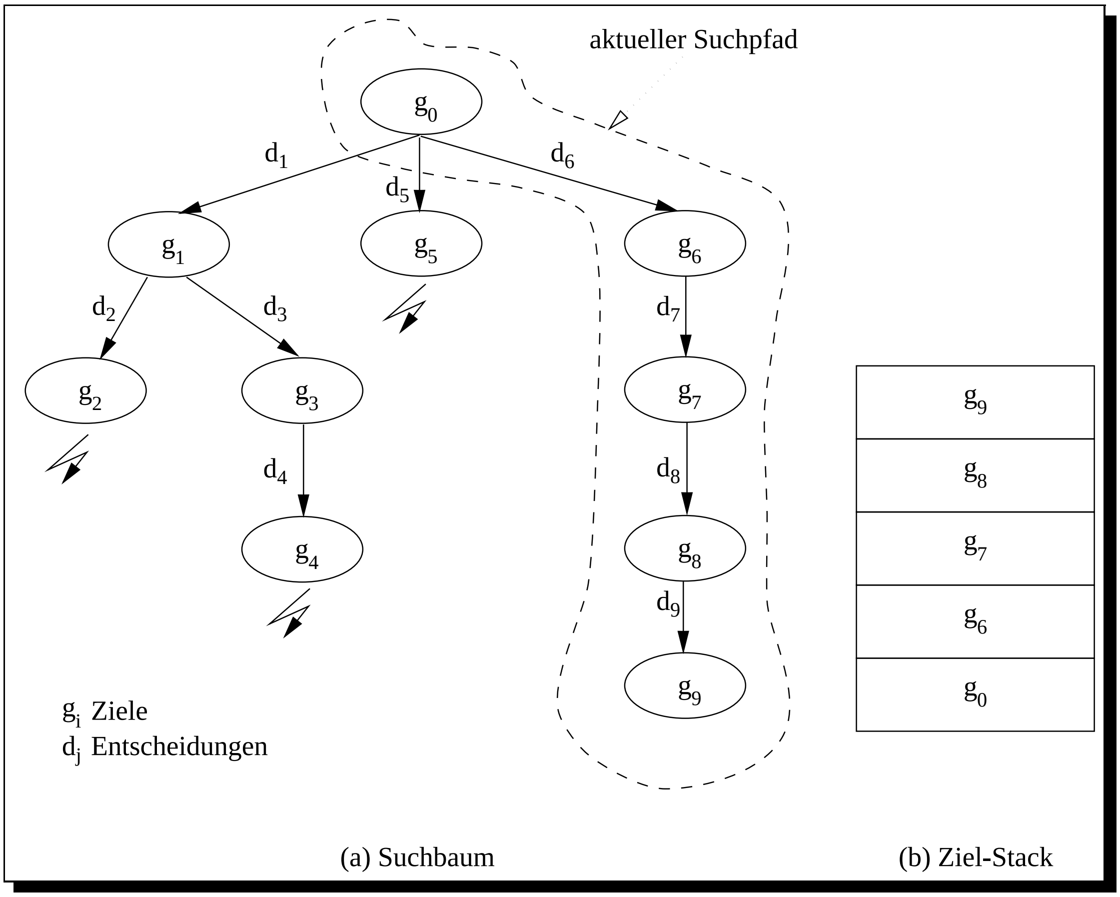

Abbildung 5.3: Suchbaum in CAPLAN/EBL mit zugehörigem Ziel-Stack

alle Variablen von p und q gebunden. Für CAPLAN existiert ein (potentieller) Threat bereits bei möglicher Übereinstimmung. Eine derartige Bedrohung kann hier durch zusätzliche Constraints abgewendet werden, die p und q verschieden machen (vgl. Separation, Abschnitt 2.1.3, siehe auch Fußnote Kapitel 4 auf Seite 21). Die Einbindung von Zurückweisungsregeln für die Separation-Entscheidung zeigte jedoch bisher keine befriedigenden Resultate.

Zusätzliche Establishing-Möglichkeiten. In der Wahl von Establishern besteht bei CAPLAN durch seine erweiterte Domänenbeschreibung eine weitere Abweichung. Die Spezifikation einer Domäne läßt die Definition von *Invarianten* zu. Invarianten sind Prädikate, die in der Domäne (und somit während jedes Problemlösevorgangs) durchweg gültig sind. In der Blocksworld ist dies z. B. für $Clear(Table)$ immer gegeben. Dadurch kann bei der Phantomisierung nicht nur auf Prädikate aus der initialen Situation und aus Effekten bereits eingeführter Schritte zurückgegriffen werden, sondern auch auf Invarianten.

Abhängigkeitsverwaltung REDUX. Im Gegensatz zu SNLP+EBL speichert CAPLAN nicht alle Pläne, die das System bereits erzeugt hat, sondern bearbeitet nur einen einzigen aktuellen Plan, der einem Pfad im Suchbaum entspricht. REDUX ersetzt die Speicherung aller erzeugten Pläne durch die Verwaltung der Abhängigkeiten zwischen Zielen und darauf angewandten Operatoren (und verhindert schon dadurch den zweimaligen Besuch des gleichen Planzustandes). Deshalb kann der Lernprozeß nur *während* eines Planungsvorgangs durchgeführt werden. Eine nachträgliche Betrachtung der einzelnen Pläne ist nicht möglich.

Typisierte Variablen. Ein letzter wichtiger Unterschied zwischen den Systemen ist die Verwendung von *Typen* in CAPlan. Die Domänenspezifikation erlaubt die Definition einer Typenhierarchie für die Variablen in den Operatorschemata. Bei der Generalisierung der Regeln mußte daher sichergestellt werden, daß entsprechende Typen-Constraints in den Bedingungsteil der Regeln aufgenommen wurden (vgl. Abschnitt 5.4). Abbildung 5.4 zeigt einen Ausschnitt aus der Typenhierarchie der Drehteil-Domäne.

- ProcessingArea
  - WorkpieceSide
    - WorkpieceSide1
    - WorkpieceSide2
- ...
- Tool
  - ExternRotatoryTool
    - LeftRotatoryTool
    - RightRotatoryTool
  - FeatureTool
    - GrooveTool
    - InterneRotatoryTool
    - RoundOffTool
    - ...
- WorkpieceType
  - Disk
  - LongSpindle
  - Spindle

Abbildung 5.4: Ausschnitt aus der Typenhierarchie für die Drehteil-Domäne

5.3 Erzeugung von Fehlererklärungen

Der Auslöser für die Erzeugung von Fehlererklärungen ist ein blockiertes Ziel. Sobald auf ein Ziel kein Operator mehr angewandt werden kann, wird Backtracking eingeleitet, d. h. der Operator, der zu diesem Ziel führte, wird zurückgezogen. Für die Operatoren in der Konfliktmenge des nicht erfüllten Ziels werden an dieser Stelle initiale Erklärungen erzeugt.

Abbildung 5.5 stellt diese Situation dar. Der Operator *selectedOp* gehört zum Vorgängerziel g_i auf dem Goal-Stack. Er wird vom CAPlan-Grundsystem zurückgenommen und von

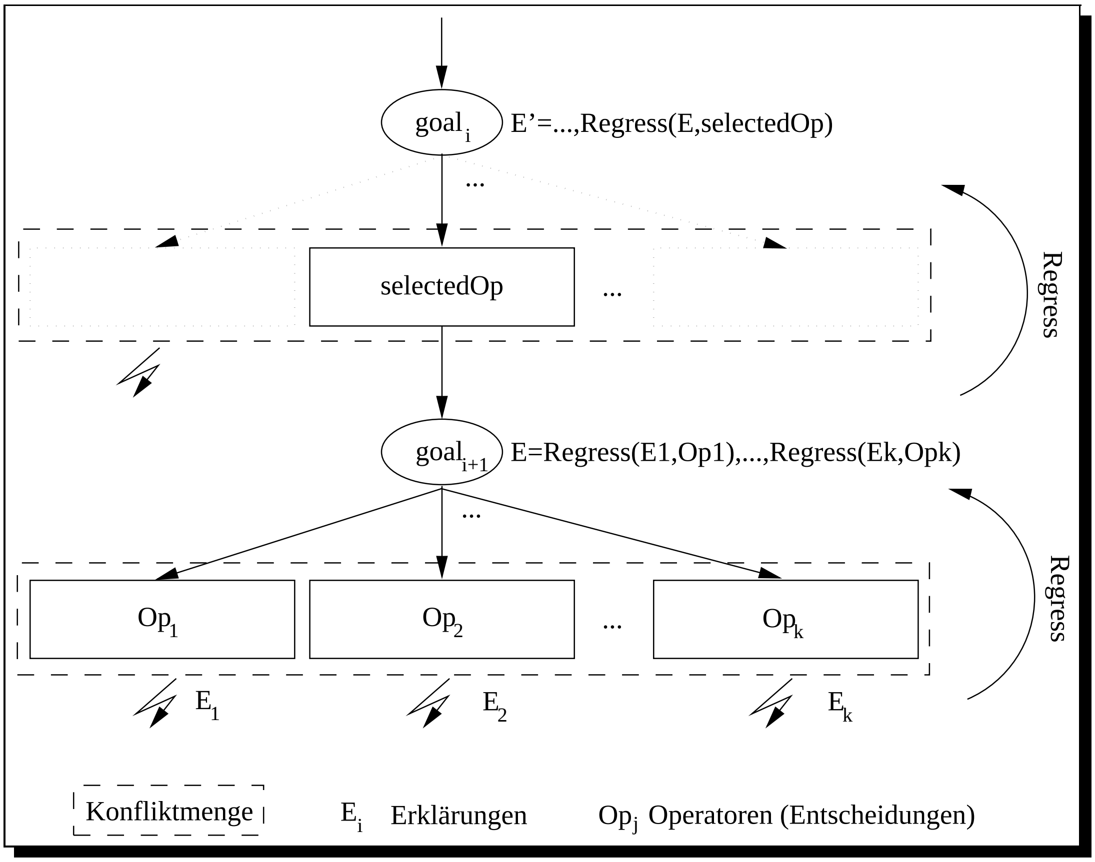

Abbildung 5.5: Initiale Erklärungen, Regression und Propagierung in CAPLAN/EBL

REDUX mit einer Zurückweisungsrechtfertigung versehen. Das blockierte Ziel g_{i+1} hat als Konfliktmenge die fehlgeschlagenen Operatoren op_1 bis op_k. Für jeden von diesen Operatoren muß eine initiale Fehlererklärung erzeugt werden. Als initiale Fehlererklärung wird exakt die vom Konsistenztester als inkonsistent erkannte Menge von Elementen des aktuellen Plans verwendet, die im Basissystem CAPLAN automatisch als Rückzugsbegründung eingesetzt wird (vgl. [Weberskirch, 1994]). Die so erhaltenen Fehlererklärungen $\mathcal{E}_1$ bis $\mathcal{E}_k$ werden danach über die jeweiligen Operatoren regrediert und resultieren in der Fehlschlagserklärung $\mathcal{E}$ am Ziel g_{i+1}. Die Regrediente wird dann am Zielknoten gespeichert und über den zurückgewiesenen Operator *selectedOp* zusammen mit den Constraints, die das offene Ziel oder einen Threat beschreiben, zum Ziel g_i hinaufpropagiert.

5.4 Regelerzeugung und Generalisierung

Aus der für den Fehlschlag der Demotion-Entscheidung S1$\prec$S4 erzeugten Erklärung aus Abbildung 5.6

$$\begin{aligned}\mathcal{E}' &= \text{Threat(START-[+Clear(b)]-S2,S4-[-Clear(X)])},\\ &\quad \text{(START}\prec\text{S1), (S4}\prec\text{S3), (S3}\prec\text{S2), Same(X,b)}\end{aligned}$$

wird nun im Laufe von zwei Phasen eine verallgemeinerte Regel generiert.

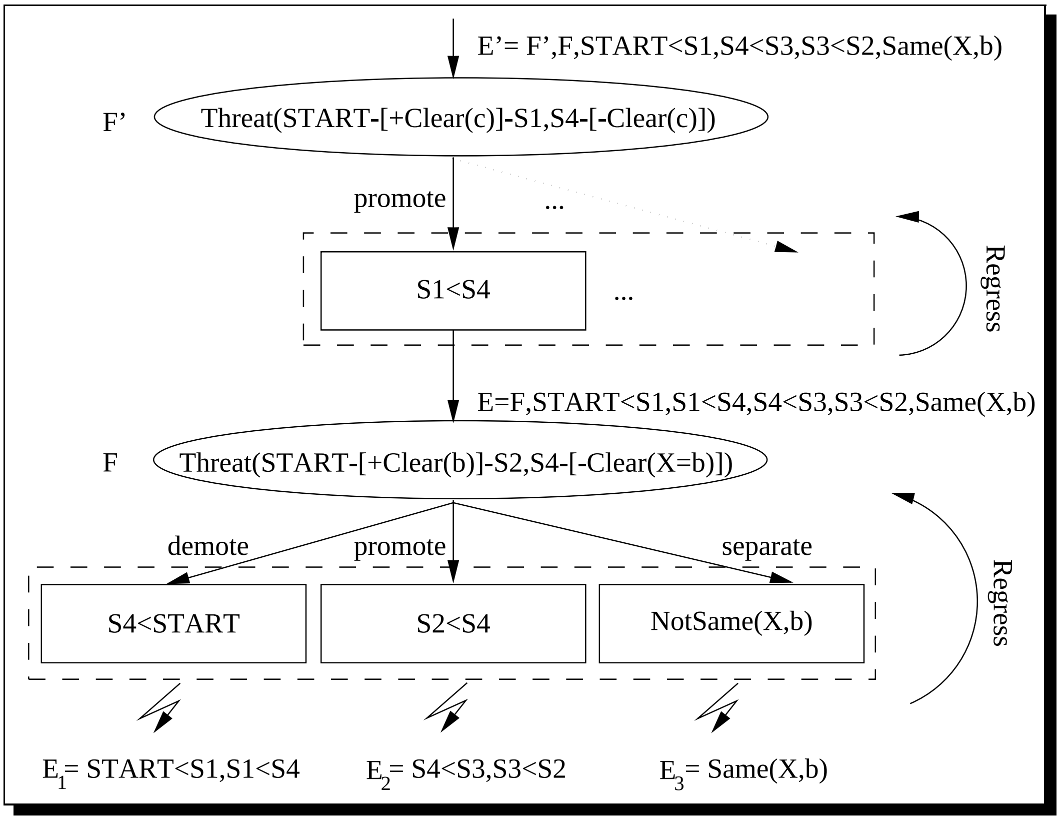

Abbildung 5.6: Beispiel für initiale Erklärungen und ihre Regression in CAPLAN/EBL

Regelerzeugung. Stehen am Ziel g_i noch andere Operatoren als Alternativen zur Verfügung (d. h. die Konfliktmenge von g_i ist nicht leer), dann wird aus $\mathcal{E}$ eine Regel erzeugt. Wichtig ist dabei die Generierung von Typ-Constraints, damit eine erlernte Regel nicht zu allgemein für CAPLAN ist.

Für das obige Beispiel erhalten wir folgende Regel:

```
rejectOrdering(S1,S4) :-

    hasThreat(START,add,Clear(b),S2,S4,del,Clear(X)),
    hasOrdering(START,S1),
    hasOrdering(S4,S3),
    hasOrdering(S3,S2),
    same(X,b).
```

Generalisierung. Die erzeugte Regel muß nun generalisiert werden, um nicht nur auf eine spezielle Situation zu passen. Da CAPLAN eine namen-insensitive Domänentheorie (siehe Abschnitt 4.5) verwendet, besteht die Verallgemeinerung hauptsächlich in einer Variablenersetzung (das Präfix `var` kennzeichnet hierbei Variablen) und in der Qualifizierung der Variable `varF3`, die für den Start-Schritt eingesetzt wird. Für die Qualifizierung dient das Prädikat *isStart*() (vgl. StepAddition-Entscheidung in Abschnitt 4.3.3). Zusätzlich müssen jedoch die Variablentypen beachtet werden. Dazu fügt der Regelgenerator für alle Variablen, die nicht für Planschritte stehen, Prädikate der Form $IsOfType()$ ein.

Es ergibt sich als Resultat der Generalisierung:

```
rejectOrdering(varF1,varF2) :-
    hasThreat(varF3,add,Clear(varF4),varF5,varF2,del,Clear(varF6)),
    hasOrdering(varF3,varF1),
    hasOrdering(varF2,varF7),
    hasOrdering(varF7,varF5),
    same(varF6,varF4),
    isOfType(Block,varF4),
    isOfType(Block,varF6),
    isStart(varF3).
```

An diesem Beispiel läßt sich deutlich der Unterschied zu SNLP+EBL bezüglich der typisierten Variablen erkennen. Die beiden Prädikate `isOfType(Block,varF4)` und `isOfType(Block,varF6)` wären im System SNLP+EBL nicht Bestandteil der Zurückweisungsregel.

5.5 Regelverarbeitung

Für die Abarbeitung der Regeln wurde eine existierend Anbindung zwischen SMALLTALK und dem Regelinterpreter PROLOG als Grundlage genutzt. Anhand des Ablaufs einer Operatorzurückweisung werden die Mechanismen näher erläutert.

5.5.1 Regelverarbeitung mit PROLOG

Für die Auswertung der Zurückweisungsregeln wurde PROLOG ausgewählt. PROLOG zeichnet sich insbesondere im Vergleich zu einem in SMALLTALK implementierten Regelinterpreter durch seine Effizienz in der Regelverarbeitung aus. Die Bekanntheit von Syntax und Semantik läßt eine bessere Bewertung der erlernten Regeln durch den Benutzer zu. Konkret wurde das als Public-Domain-Software zur Verfügung stehende SWI-PROLOG verwendet.

Die Anbindung des PROLOG-Interpreters an CAPLAN/EBL erfolgte über eine für KI-Anwendungen entwickelte Schnittstelle von [Niehaus & Lorscheid, 1995], die eine Kommunikation zwischen beiden Systemen mit Hilfe von UNIX-Sockets realisiert.[1]

Die Regelverarbeitung läßt sich in zwei Phasen einteilen: in die Initialisierungs- und in die Verarbeitungsphase.

Initialisierungsphase. Um unnötige Rückfragen an CAPLAN zu vermeiden, werden zu Beginn eines Problemlösungslaufs alle Fakten, die initial gelten und während des Problemlöselaufs unverändert bleiben, aus der statischen Wissenbasis, genauer, aus der Problembeschreibung, an PROLOG übertragen. Dies sind zum einen Prädikate vom Typ

[1] Die Kommunikationsschnittstelle kam bereits im Konfigurationssystem IDAX [Schirp, 1995] zur Verarbeitung von Präferenzregeln erfolgreich zum Einsatz.

initiallyTrue(), die angeben, was in der Start-Situation gilt und welche Invarianten zur Verfügung stehen. Zum anderen werden zu den Objekten, die in der Problembeschreibung definiert sind, Prädikate über deren Typ an PROLOG übertragen, z. B. $IsOfType(Block, a)$ für ein Block namens a. Ein weiterer Zugriff auf die statische Wissensbasis während des Problemlösevorgangs erfolgt danach nicht mehr.

Verarbeitungsphase. Um während des Planungsprozesses die Inkonsistenz eines Operators beweisen zu können, muß der Regelinterpreter auf die dynamische Wissensbasis zugreifen können. Abbildung 5.7 zeigt schematisch die Kommunikationsverbindungen zwischen PROLOG und verschiedenen Komponenten von CAPLAN/EBL. Da das PROLOG-System nicht unmittelbar auf die Daten in der dynamischen Wissensbasis zugreifen kann, müssen diese Fakten in die Datenbasis von PROLOG übertragen werden. In diesem Sinne stellt CAPLAN/EBL für den Regelinterpreter eine externe Datenbank dar. Auf diese Weise wurden die Probleme der Konsistenthaltung der Wissensbasen von CAPLAN/EBL und PROLOG umgangen.[2]

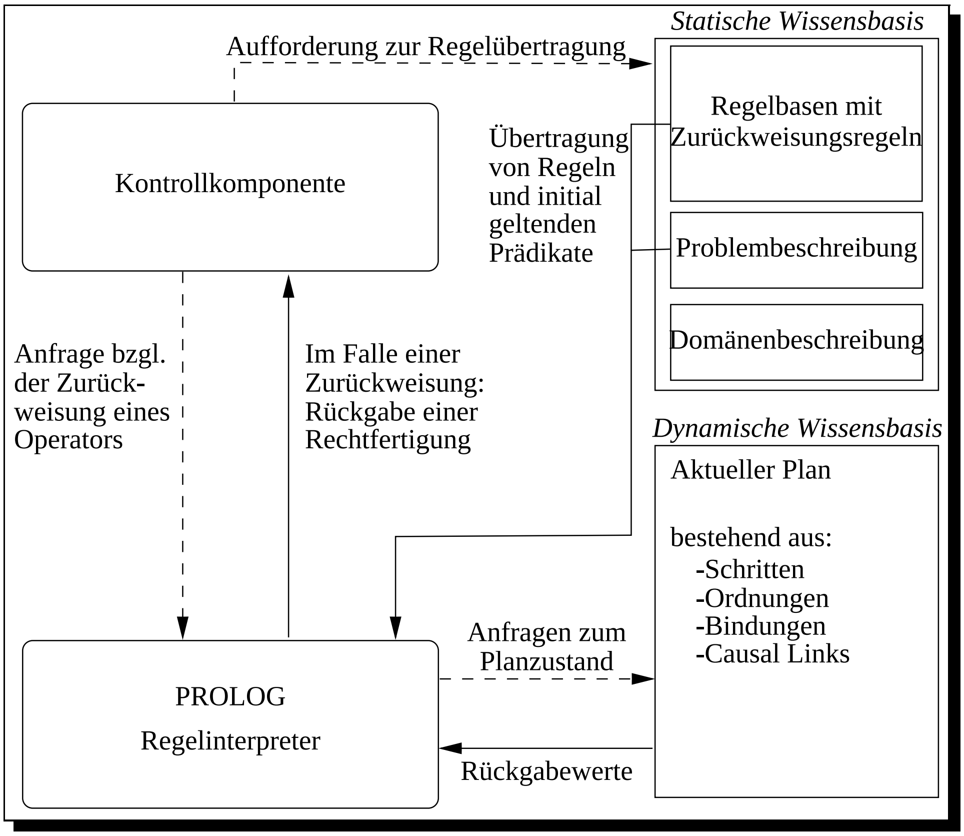

Abbildung 5.7: Schema einer Operatorzurückweisung mit Zurückweisungsregeln

Für die Regelauswertung wird der Regelinterpreter zum *Server*, an den CAPLAN/EBL als *Client* Anfragen bezüglich der Inkonsistenz von Operatoren in Form eines PROLOG-Prädikates richten kann, z. B.:

```
rejectOrdering(S1,S4).
```

[2]Eine allgemeine Diskussion der Kopplung von Expertensystem mit Regelinterpretern findet sich in [Richter, 1992].

Sei nun in der Regelbasis von PROLOG nur die in Abschnitt 5.4 gelernte Regel zur Zurückweisung einer Ordnungsentscheidung enthalten. Dieses Prädikat versucht der Regelinterpreter zu beweisen, wobei innerhalb des Beweises Rückfragen an CAPLAN/EBL gestellt werden können, z. B. die Anfrage nach negativen Threats im aktuellen Plan:

```
hasThreat(varF3,add,Clear(varF4),varF5,S4,del,Clear(varF6))
```

So kann PROLOG Fakten aus der dynamischen Wissensbasis erhalten, die zum Beweis von Bedingungen aus dem Rumpf der Regeln benötigt werden. Da in der obigen Rückfrage an CAPLAN/EBL noch freie Variablen enthalten sind, besteht die Antwort für PROLOG in eine Liste von Threats, auf die das Muster paßt, beispielsweise könnten folgende beiden Threats im aktuellen Plan enthalten sein:

```
((S2,add,Clear(c),S3,S4,del,Clear(c))
(S5,add,Clear(a),S14,S4,del,Clear(a)))
```

Die erste Bedrohung besteht also zwischen dem Causal Link $s_2 \overset{+Clear(c)}{\longrightarrow} s_3$ und dem bedrohenden Schritt $s_4 \mapsto -Clear(c)$, die zweite zwischen dem Causal Link $s_5 \overset{+Clear(a)}{\longrightarrow} s_{14}$ und $s_4 \mapsto -Clear(a)$.

Für Rückfragen wurden entsprechende Prädikate definiert, mit deren Hilfe die relevanten Fakten der dynamischen Wissensbasis, also des aktuellen Plans, beschrieben werden können. Syntax und Semantik der Prädikate sind in Anhang A kurz erklärt.

Für jedes vordefinierte Prädikat ist in der PROLOG-Regelbasis mindestens eine Regel definiert, die die Auswertung des entsprechenden Prädikates beschreibt. Besitzt ein vordefiniertes Prädikat Parameter, die beim Aufruf wahlweise freie Variable oder feste Werte enthalten können, so existiert für jede dieser Möglichkeiten eine entsprechende Auswertungsregel. Bei der Abarbeitung des Bedingungsteils der Auswertungsregel eines Prädikates erfolgt immer eine Anfrage an die dynamische Wissenbasiss, um von dort die Gültigkeit der Aussage zu erfragen, die vom betreffenden Prädikat repräsentiert wird. Eine solche Anfrage erfolgt, indem der Regelinterpreter über die Schnittstelle eine SMALLTALK-Methode aufruft, die die entsprechende Frage aufgrund der Fakten in der dynamischen Wissensbasis beantwortet. Zu jeder Auswergungsregel gehört genau eine bestimmte Methode in CAPLAN/EBL, so daß bei Auswertung von vordefinierten Prädikaten stets Paare von PROLOG-Regeln und SMALLTALK-Methoden eingesetzt werden.

Die Anfrageprädikate an die dynamische Wissensbasis lassen sich bezüglich der Antwort von CAPLAN/EBL in zwei Gruppen unterteilen (vgl. [Schirp, 1995]):

- Prädikate, die nach der *Gültigkeit einer Bedingung* fragen, z. B.: *Existiert eine Ordnung zwischen den Schritten s_1 und s_2?*
- Prädikate, die ein Faktum als Ergebnis benötigen, z. B.: *Gib mir alle Schritte, die hinter s_1 liegen.*

Um für REDUX die Zurückweisung eines Operators zu begründen, muß CAPLAN/EBL noch einen weiteren Wert liefern. Zusätzlich zu den eben erwähnten Antworten müssen diejenigen Knoten aus dem Abhängigkeitsnetzwerk des TMS ermittelt werden, die die Gültigkeit der Antwort garantieren. Für unsere Zwecke genügen die Zuweisungsknoten (assignments), da diese die aktuellen Variablenbindungen und Ordnungen im Plan festhalten.

Hat PROLOG einen Beweis für die Inkonsistenz eines Operators gefunden, kann CAPLAN/EBL aus den TMS-Knoten eine Rechtfertigung für die Zurückweisung erzeugen, die sich auf den aktuell gültigen Planzustand bezieht.

5.5.2 Ablauf einer Operatorzurückweisung

Betrachten wir noch einmal die Backtracking-Punkte im SNLP-Algorithmus (Abschnitt 2.1.3). Der eine Backtracking-Punkt ist dort die Threat-Auflösung durch Promotion, Demotion oder Separation. Der andere Punkt ist die Auswahl eines Operators, um ein Ziel mittels einer Phantomisierung oder mit Hilfe eines neuen Schrittes zu reduzieren.

Wie bereits in Abschnitt 2.1.3 erwähnt, wurde in CAPLAN eine einheitliche Sicht auf eine Menge von Zielen und darauf anzuwendende Operatoren durch die Einführung eines *Protection Goal* realisiert. Die Konfliktmenge (siehe Abschnitt 2.2) definiert die Möglichkeiten, Operatoren auf ein Ziel anzuwenden. Aus dieser Menge werden von CAPLAN alle Operatoren zurückgewiesen, die entweder ordnungs- oder bindungsinkonsistent sind.

Eine Erweiterung der Konsistenzprüfung erlaubt CAPLAN/EBL, die verbliebenen Operatoren mit Hilfe der erlernten Regeln darauf zu testen, ob die Weiterverfolgung dieses Pfades des Suchbaums überhaupt zu einem Erfolg führen kann. Beweist PROLOG eine Zurückweisungsregel, so liefert der Regelinterpreter die aktuell gültigen Zuweisungen, die zu einem Fehlschlag des betrachteten Planes führen würden, sollte der Operator angewendet werden. Aus diesen Zuweisungen erzeugt CAPLAN/EBL Rechtfertigungen für REDUX, damit der zurückgewiesene Operator mindestens solange von einer Anwendung ausgeschlossen bleibt, bis die von PROLOG gefundenen Zuweisungen nicht mehr gelten.

Werden diese Zuweisungen im Zuge weiteren Backtrackings aufgehoben, kann ein Operator, der durch eine Zurückweisungsregel aufgrund zu jenem Zeitpunkt bestehender Ordnungsbeziehungen und Bindungen zurückgewiesen wurde, durchaus wieder anwendbar werden.

5.6 Erste Testergebnisse

Der Lernmechanismus wurde ersten Tests unterzogen und ergaben durchweg positive Resultate in Bezug auf die Anzahl der Inferenzschritte, die CAPLAN/EBL ohne und mit Verwendung von Regeln zur Problemlösung benötigte. Im folgenden werden drei Domänen, davon eine künstliche Domäne, betrachtet und die Resultate jeweils anhand von Testergebnissen erläutert.

Die Tabellen sind folgendermaßen aufgebaut: Die erste Spalte listet die Namen der einzelnen Problemstellungen auf. Spalte zwei zeigt die Anzahl der Inferenzschritte, die *ohne* Verwendung von Regeln zur Lösung des angegebenen Problems benötigt wurden. Die weiteren Spalten enthalten jeweils die Anzahl der Inferenzschritte, die CAPLAN/EBL *mit* der Anwendung von Regeln benötigte, die aus dem im Kopf der Spalte genannten Problem gelernt wurden. Die Abkürzung `n.e.` bedeutet hierbei, daß der Wert aus Zeitgründen *nicht ermittelt* wurde.

Als eine äußerst hilfreiche Heuristik entpuppte sich an dieser Stelle eine Einschränkung des Lernens von Zurückweisungsregeln auf jene Regeln, deren Rumpf höchstens zwölf[3] Planzustandsprädikate enthalten. Die EBL-Komponente erzeugt je nach Benutzerwunsch auch erheblich größere Regeln, deren Kosten-Nutzen-Verhältnis aufgrund ihrer seltenen Anwendungsmöglichkeit wesentlich schlechter ist.

Ein erstaunliches Resultat ist die Tatsache, daß schon bei einfacheren Problemen diejenigen Regeln gelernt werden, die in den Regelmengen schwierigerer Probleme enthalten sind. Diese

[3] Diese Zahl ergab sich aus zufälligen Tests. Ab einer Anzahl von fünfzehn Prädikaten im Rumpf begann die Leistung des Gesamtsystems aufgrund der Vielzahl an möglichen Inferenzen stark nachzulassen.

Regeln führen zur Leistungssteigerung, können jedoch anhand leichterer Probleme gewonnen werden.

5.6.1 Die künstliche Domäne ART-1D-RD

In der typenlosen Testdomäne ART-1D-RD (vgl. [Kambhampati, 1993]) liegt die Schwierigkeit für den Planer in alternierenden Operatoranwendungen, wobei ein Operator einen Effekt erzeugt und der andere Operator dessen Effekt wieder zerstört. Die Lösung für ein Problem in dieser Domäne erlaubt außerdem aufgrund von Elementen in der Delete-Liste nur eine Reihenfolge der Operatoren und erfordert somit häufiges Backtracking. Die Zahl im Problemnamen gibt hierbei die Anzahl der Operatoren an, die, in der richtigen Reihenfolge ausgeführt, das Problem lösen.

	ohne Regeln	mit gelernten Regeln aus		
Problem		P3RD	P5RD	P8RD
P3RD	714	114	n. e.	n. e.
P5RD	1937	831	441	n. e.
P8RD	8921	n. e.	n. e.	484

Abbildung 5.8: Testergebnisse aus der künstlichen ART-1D-RD-Domäne.

Die Ergebnisse in Abbildung 5.8 zeigen eindrucksvoll die Nützlichkeit der gelernten Regeln für die betrachteten Kombinationen von Problemen und Regelmengen, für weitergehende Aussagen stehen hier jedoch zuwenige Werte zur Verfügung.

5.6.2 Die Frosch-Domäne

Die *Frosch*-Domäne stammt von Hector Muñoz und zeichnet sich durch die Irreversibilität der Operatoranwendung aus, d. h. von Operatoren zerstörte Effekte können nicht durch andere Operatoranwendungen wiederhergestellt werden. Die Frosch-Domäne entspricht u. a. der Flugzeug-Domäne, die in der Beschreibung von DERSNLP+EBL verwendet wird, mit der Beschränkung, daß ein Flughafen nur einmal angeflogen werden darf (vgl. [Ihrig & Kambhampati, 1996]). Die Zahlentripel geben jeweils an, wieviele Blätter, Frösche (hier nur ein Frosch) und Fliegen in der Problemstellung vorhanden sind. Der Frosch soll dabei von Blatt zu Blatt hüpfen und die dort sitzenden Fliegen verzehren. Beim Sprung zum nächsten Blatt wird das Blatt, von dem der Frosch weggesprungen ist, versenkt und keine Aktion des Frosches kann diesen Effekt rückgängig machen. Dies kann nur durch Backtracking behoben werden.

	ohne Regeln	mit gelernten Regeln aus		
Problem		3-1-2	4-1-3	5-1-4
3-1-2	18	15	15	15
4-1-3	353	347	282	291
5-1-4	1858	n. e.	n. e.	1015

Abbildung 5.9: Testergebnisse aus der Frosch-Domäne.

Abbildung 5.9 zeigt deutlich die Leistungsfähigkeit der erlernten Regeln. Mit zunehmender

Komplexität des Problems ergibt sich ein günstigeres Bild für die Regelanwendung. Das Problem *3-1-2* war sehr einfach und bot kaum eine Chance für den Erwerb und die Anwendung von Kontrollwissen. Am nächsten Problem kann man deutlicher die Leistungssteigerung erkennen. Mit den Regeln, die aus demselben Problem gelernt wurden, ergab sich eine Verringerung der Inferenzenzahl, die beim Lernen aus dem schwierigeren Problem *5-1-4* nicht erreicht wurde. Die Auswertung von Problem *5-1-4* konnte in der zur Verfügung stehenden Zeit nicht weiter ausgeführt werden. Der Nutzen der aus demselben Problem gewonnenen Regeln läßt sich jedoch nicht bestreiten.

5.6.3 Die Drehteil-Domäne

Auch die Drehteil-Domäne zeichnet sich durch die Irreversibilität der Operatoranwendungen aus. Hier sind die Objekte und Aktionen modelliert, die es erlauben, für rotationssymmetrische Drehteile entsprechende Arbeitspläne zu ihrer Fertigung auf CNC-Drehmaschinen zu erstellen. Die Domäne enthält eine Vielzahl primitiver Aktionen, deren Anwendung unterschiedliche Aspekte bezüglich der Drehteile realisieren (vgl. [Muñoz-Avila *et al.*, 1995]). Die Drehteil-Domäne ist von den drei genannten die realistischste.

	ohne Regeln	mit gelernten Regeln aus			
Problem		complete	undercut	undercut-1-G	undercut-2-G
complete	97	92	92	92	92
undercut	164	153	153	153	153
undercut-1-G	112	98	98	96	98
undercut-2-G	133	122	122	119	120

Abbildung 5.10: Testergebnisse aus der Drehteil-Domäne.

In den Beispielen von Abbildung 5.10 war jeweils ein Arbeitsplan zur Fertigung einer Welle zu erstellen. Das erste Problem bezog sich auf deren vollständige Erstellung, die anderen auf eine teilweise Erstellung bezüglich einer Hinterschneidung, sowie bezüglich einer Hinterschneidung und einer bzw. zwei Nuten.

In der Tabelle ist ebenfalls eine Abnahme in der Zahl der Inferenzschritte bei Hinzunahme von Regeln zu verzeichnen. Auch hier ist wieder (wenn auch schwach) zu erkennen, daß Regeln, die aus einfacheren Problemstellungen gelernt wurden, bereits dieselben Ergebnisse liefern, wie die Regeln aus schwierigeren Problemen. Diese Beobachtungen müssen jedoch erst noch durch genauere Test untermauert werden. Die Ersparnis ist offensichtlich auch nicht so groß wie in der künstlichen Domäne ART-1D-RD, was auf der Vielzahl möglicher Threats in dieser Domäne beruhen könnte. Anhand der Domäne ART-MD-RD, die durch eine große Anzahl von Interaktionen gekennzeichnet ist, könnte diese Beobachtung näher untersucht werden.

Kapitel 6

Diskussion und Ausblick

Mit dem System CAPLAN/EBL wurde eine Grundlage für weitergehende Arbeiten im Bereich des erklärungsbasierten Lernens in der partiell-ordnenden Planung geschaffen. Dieses Kapitel faßt die interessanten Aspekte des erklärungsbasierten Lernens aus Fehlschlägen zusammen und gewährt einen Ausblick auf Erweiterungen von CAPLAN/EBL.

6.1 Diskussion

Erklärungsbasiertes Lernen ist ein Lernverfahren, dessen Stärke von der Mächtigkeit der Hintergrundtheorie abhängt. Für diese Arbeit ist das SNLP-Planungsverfahren die Hintergrundtheorie, in deren Rahmen eine Erklärung dafür erzeugt wird, daß ein Trainingsbeispiel das Zielkonzept erfüllt. Das Zielkonzept beschreibt hier Planungsfehlschläge, die sich als Inkonsistenzen im Plan manifestieren. Das Operationalisierungskriterium verlangt hier, daß Fehlschläge in Form von Causal Links, Threats, Ordnungsbeziehungen und Bindungen zu beschreiben sind, da diese Elemente gut in einem Plan zu testen sind. Die verwendeten Trainigsbeispiele sind Fehlschläge des Planungsverfahrens und bestehen genau aus den beschriebenen operationalen Elementen.

Darauf aufbauend wurden Prädikate zum Test auf Planzustände entwickelt, mit deren Hilfe ein Regel-Interpreter, in diesem Falle PROLOG, Anfragen bezüglich der Anwendbarkeit eines Operators beantworten kann. Dabei mußte der Tatsache Rechnung getragen werden, daß in CAPLAN bereits bearbeitete Planzustände nicht gespeichert werden, sondern daß REDUX über die Abhängigkeiten zwischen Operatoren und Zielen buchführt. In dieser Besonderheit von CAPLAN lag eine der Schwierigkeiten bei der Umsetzung des EBL-Verfahrens für CAPLAN. Für die Zurückweisung eines Operators mußten Rechtfertigungen geliefert werden, damit ein durch Regeln zurückgewiesener Operator korrekt von REDUX verwaltet werden kann.

Als ein Vorteil von REDUX ergab sich, daß die initialen Erklärungen direkt aus den Rechtfertigungen für die Zurückweisung eines Operators durch das Grundsystem gewonnen werden können. Auch wenn sich SNLP+EBL und CAPLAN/EBL in der Struktur ihrer Suchbäume unterscheiden, bleibt hier festzuhalten, daß die Regression initialer Fehlererklärungen und auch die Propagierung der Erklärungen im Suchbaum analog verläuft. Mithin ist kein gesonderter Aufbau eines Beweisbaums wie bei [Minton, 1988] nötig, sondern der Suchbaum kann direkt verwendet werden. Ein Umstand, der die Effizienz des Lernverfahrens erhöht.

Eine Nutzenanalyse wurde bisher nur auf sehr einfache Weise durchgeführt und beschränkte sich auf die Anwendungshäufigkeit von Regeln. Die Anzahl der Prädikate im Rumpf wurde als grobes Maß für die Match-Kosten genommen. Hier wäre eine Dynamisierung wünschenswert, d. h. bei Plänen mit wenigen Planschritten sollten auch nur Regeln mit wenigen Prädikaten verwendet werden, bei größeren Plänen entsprechend längere Regeln.

Ein weitere Unterschied zwischen SNLP+EBL und CAPlan/EBL besteht in der Verwendung von Typen. CAPlan verwendet eine Typenhierarchie, die bei der Generalisierung von Regeln bisher nicht beachtet wird. Hier sollte ebenfalls eine Verallgemeinerung des Variablentyps bis zum allgemeinsten Typ, der in Bezug auf die definierte Domäne noch Sinn macht, stattfinden. An dieser Stelle müsste somit die Domänenbeschreibung erweitert werden.

Die EBL-Komponente wurde so in das modulare Grundsystem eingebracht, daß sie problemlos mit den bereits realisierten Kontrollkomponenten zur Planungssteuerung eingesetzt werden kann. Die verwendete Planungssteuerung wirkt sich auch auf den Lernmechanismus aus, da ein verändertes Suchverfahren auch zu unterschiedlichen Zielblockaden führt bzw. führen kann. Hier wäre es denkbar, mit dem einen Verfahren zur Operatorwahl zu lernen und dann die gelernten Regeln bei einem anderen Verfahren zur Einschränkung der Operatorwahl zu nutzen. Auch hier können nur gezielte Tests Klarheit verschaffen.

In diesem Zusammenhang hat Redux ebenfalls einen Vorteil zu bieten. Die Abhängigkeitsverwaltung ist ein Mittel zur Einbeziehung des Benutzers in den Planungsprozeß. Der Benutzer kann in CAPlan Entscheidungen treffen und auch zurücknehmen. Bei jeder dieser Aktionen sorgt Redux für einen konsistenten Plan. Wird der Planer durch den Benutzer in eine Sackgasse geführt, sorgt die Abhängigkeitsverwaltung entsprechend für die Auslösung des notwendigen Backtrackings. Somit kann der Benutzer gezielt Situationen zum Lernen aus Fehlschlägen erzeugen.

Eine wichtige Erweiterung von CAPlan/EBL wäre eine Möglichkeit, Regeln wieder zu vergessen. Bisher werden nicht angewendete Regeln manuell aus der Regelbasis entfernt, um die Leistung des Systems zu erhalten. Ein automatisches Verfahren sollte den Benutzer jedoch hierbei entlasten.

Der Hauptnachteil des erklärungsbasierten Lernens in SNLP liegt in der Tatsache begründet, daß nur solche Entscheidungen lernbar sind, die als Backtrackingpunkte ausgezeichnet sind, d. h. nur die Operatorauswahl und die Threatauflösung. Für diese Punkte können jedoch Regeln erlernt werden, die den Suchraum stark einschränken. EBL darf jedoch nicht isoliert gesehen werden, sondern sollte durch statische Kontrollmethoden erweitert werden, wie sie beispielsweise in [Kettnaker, 1995] implementiert und analysiert wurden. Der dort beschriebene Toolbox-Ansatz stellt statische Methoden zur Verfügung, die mit dem EBL-Verfahren durch den modularen Aufbau von CAPlan gut kombiniert werden können.

6.2 Ausblick

Folgende Arbeiten stehen noch an:

- Die Verschränkung von Lernen und Auswerten von Zurückweisungsregeln, wie es in Abschnitt 4.2.3 indirekt zum Ausdruck kommt, muß noch in Einklang mit der Abhängigkeitsverwaltung Redux gebracht werden. Feuert eine Zurückweisungsregel, dann muß sichergestellt werden, daß die Informationen, die zum Zeitpunkt der Regelerzeugung zur Verfügung stehen, auch vom Regelinterpreter wieder an CAPlan/EBL zurückgeliefert werden, um diese Constraints dem Regressions- und Propagierungsmechanismus zuzuführen.

- Aus dem Propagierungsmechanismus von Abschnitt 4.4 ergibt sich auf natürliche Weise ein Verfahren für abhängigkeitsgerichtetes Backtracking. Im Rahmen des Lernens während der Planung dienen dabei die Fehlererklärungen dem Planer als Entscheidungshilfe, ob ein Pfad im Suchbaum weiter beschritten werden soll oder gefahrlos übergangen werden kann. Bleibt eine Fehlererklärung für den Knoten n bei der Regression über die Entscheidung $d(n)$ unverändert, dann kann der Planer alle Nachfolgeknoten von n gefahrlos übergehen und zum Vaterknoten von n gehen. Der Prozeß kann dort weiterverfolgt werden, bis sich eine Änderung der Fehlererklärung ergibt. [Kambhampati, 1996] formalisiert abhängigkeitsgerichtetes Backtracking im Rahmen des erklärungsbasierten Lernens.

- Das Erkennen von Suchtiefenüberschreitungen und das Lernen von Kontrollwissen daraus (vgl. Abschnitt 4.6) ist ebenfalls ein vielversprechende Erweiterung von CAPlan/EBL, die weiterverfolgt wird.

- Das Lernen von Fehlschlägen mit Verwendung von Operatorschemata (vgl. [Keller, 1996]) ist ebenfalls ein Gebiet, welches in nächster Zukunft näher betrachtet werden wird. Die Implementierung von Operatorschemata erfolgte als modulare Erweiterung von CAPlan und Redux, und kann somit in den Lernprozeß einbezogen werden.

Anhand dieser Liste wird offensichtlich, wieviele Möglichkeiten CAPlan/EBL bietet, um ein besseres Verständnis für die generische Planung in Verbindung mit Lernen zu gewinnen.

Anhang A

Planzustandsprädikate

Am Beispiel des Prädikates `hasOrdering()` soll das Zusammenspiel zwischen PROLOG und CAPLAN/EBL verdeutlicht werden. Auf Seiten des Regelinterpreters ist das Prädikat Teil einer Zurückweisungsregel (vgl. z. B. Abschnitt 5.4) und folgendermaßen definiert (Variablen fangen nach PROLOG-Konvention immer mit Großbuchstaben an):

Regel 1:

```
hasOrdering(Plan, Step1, Step2, Ass1, Ass3) :-
    nonvar(Step1),
    nonvar(Step2),
    Step1 <> Step2,!,
    smalltalk(Plan,hasOrdering:Step1,before:Step2,Ass2),
    append(Ass1, [Ass2], Ass3).
```

Regel 2:

```
hasOrdering(Plan, Step1, Step2, Ass1, Ass3) :-
    eachPossibleOrdering(Plan, Step1, Step2),
    smalltalk(Plan,hasOrdering:Step1,before:Step2,Ass2),
    append(Ass1, [Ass2], Ass3).
```

Die erste Regel wird angewandt, wenn der Regelkopf vollständig instanziiert werden kann, d. h. die Variablen `Step1` und `Step2` sind mit Werten belegt. Die Variable `Plan` wird bereits von CAPLAN/EBL beim Aufruf einer Regel übergeben, damit die Schnittstelle weiß, an welches Objekt die Rückfragen zu stellen sind. Eine erste Plausibilitätsüberprüfung testet, ob die beiden Schritte auch unterschiedlich sind, da ansonsten eine unnötige Anfrage an CAPLAN/EBL zu erhöhtem Kommunikationsaufwand führen würde. Waren diese Tests erfolgreich, verhindert ein Cut („`!`“) den Versuch einer erneuten Auswertung mit anderer Variablenbindung, da dafür an dieser Stelle kein Interesse besteht. Eine andere Variablenbindung kann hier nur von CAPLAN/EBL kommen. Nach dem Cut erfolgt die eigentliche Auswertung des Tests mit Hilfe des Anfrageprädikates `smalltalk()`, welches zu einem Methodenaufruf von *hasOrdering: Step1 before: Step2* im Objekt *Plan* führt (vgl. [Niehaus &

Lorscheid, 1995] für eine ausführlichere Darstellung des `smalltalk`-Prädikates). Wird eine Ordnung zwischen den Planschritten `Step1` und `Step2` im aktuellen Plan gefunden, dann werden die zugehörigen Assignemnts an PROLOG übertragen und an die bisherige Liste von Zuweisungen `Ass1` per `append()` angehängt. Die neue Menge `Ass3` wird dann an das aufrufende Prädikat weitergegeben.

Die zweite Regel wird angewendet, wenn entweder nur ein Schritt oder beide Planschritte noch nicht feststehen. Mit Hilfe des Prädikates `eachPossibleOrdering` werden nacheinander alle Paare von Schritten aufgezählt, für die eine Ordnung besteht. Trifft im Rahmen einer Zurückweisungsregel eine Ordnung zu, dann wird diese in die Assignment-Liste aufgenommen.

Ein Beispiel für ein Prädikat zur Effizienzsteigerung ist z. B. das Prädikat `isOfType(Plan, Type, X,)`, welches die Frage nach dem Typ von `X` beantwortet, falls `X` mit einem Wert belegt ist, oder zu gegebenem Typ und gebundener Variable entscheidet, ob diese Bindung in der Datenbasis vorliegt. Diese Regel greift nur auf die Fakten in der Datenbasis zu, da die Typinformation zu den Objekten in der Initialisierungsphase nach PROLOG übertragen wurden und sich während des Problemlöselaufs nicht mehr ändern. Gleiches gilt für das Prädikat `initiallyTrue()`, welches Auskunft über den Start-Zustand gibt.

Anhang B

Danksagung

An dieser Stelle möchte ich Prof. M. M. Richter und allen aus der Arbeitsgruppe Expertensysteme danken, die mir während der Implementations- und Dokumentationsphase hilfreich mit Erklärungen, Diskussionen, ein paar aufmunternden Worten und viel Geduld zur Seite gestanden haben.

Unter ihnen geht mein Dank besonders an Hector Muñoz, Werner Schirp, Frank Weberskirch und Wolfgang Wilke. Jeder hat seinen Teil dazu beigetragen.

Literaturverzeichnis

BARRETT, A. UND WELD, D.S. 1992. *Partial-Order Planning: Evaluating Possible Efficiency Gains.* Bericht 92-05-01. Dep. of Computer Science and Engineering, University of Washington, Seattle, WA 98195.

BRATKO, I. 1990. *PROLOG - Programming for Artificial Intelligence.* 2. Ausg. Addison-Wesley.

CARBONELL, JAIME. 1986. Derivational Analogy: A Theory of Reconstructive Problem Solving and Expertise Acquisition. *Machine Learning*, **2**.

CHAPMAN, D. 1987. Planning for Conjunctive Goals. *Artificial Intelligence*, **32**, 333–377.

DEJONG, G. 1981. Generalizations Based on Explanations. *Seiten 67–69 aus: Proceedings of IJCAI-81.*

DEJONG, G. UND MOONEY, R. 1986. Explanation-Based Learning: An Alternative View. *Machine Learning*, **1**, 145–176.

DOYLE, J. 1979. A Truth Maintenance System. *Artificial Intelligence*, **12**(3), 231–272.

ETZIONI, O. 1993. Acquiring search-control knowledge via static analysis. *Artificial Intelligence*, **62**, 255–301.

FIKES, R.E. UND NILSSON, N.J. 1971. STRIPS: A New Approach to the Application of Theorem Proving in Problem solving. *Artificial Intelligence*, **2**, 189–208.

GOLDBERG, A. UND ROBINSON, D. 1983. *Smalltalk 80 - The Language and Its Implementation.* 2. Auflage. Stuttgart: Addison-Wesley.

IHRIG, L. UND KAMBHAMPATI, S. 1996. An Explanation-Based Approach to Improve Retrieval in Case-based Planning. *Aus:* GHALLAB, M. UND MILANI, A. (Hrsg.), *New Directions in AI Planning.* IOS Press.

KAMBHAMPATI, S. 1993. On the Utility of Systematicity: Understanding Tradeoffs between Redundancy and Commitment in Partial-order Planning. *Seiten 116–125 aus: Proceedings of IJCAI-93.*

KAMBHAMPATI, S. 1996. Formalizing Dependency Directed Backtracking and Explanation Based Learning in Refinement Search. *Aus: Proceedings of AAAI-96.*

KAMBHAMPATI, S., KATUKAM, S. UND QU, Y. 1995a. *Failure Driven Dynamic Search Control for Partial Order Planners: An explanation-based approach.* Bericht ASU-CSE-TR-95-010. Dept. of Computer Science and Engineering, Arizona State University.

KAMBHAMPATI, S., KNOBLOCK, C. UND YANG, Q. 1995b. *Planning as Refinement Search: A unified framework for evaluating design tradeoffs in partial order planning.* To appear: Artificial Intelligence Special Issue on Planning an Scheduling. ?Check again?

KELLER, C. 1996. *Realisierung hierarchischer Aktionsdekomposition im Planungswerkzeug CAPlan zur Unterstützung der Arbeitsplanerstellung bei der computerintegrierten Fertigung (CIM).* Diplomarbeit, Universität Kaiserslautern.

KELLER, R.M. 1988. Defining Operationality for Explanation-Based Learning. *Artificial Intelligence*, **35**, 227–241.

KETTNAKER, V. 1995. *Konzeption und Realisierung einer Toolbox statischer Kontrollmethoden zur Steuerung eines Causal Link Planers.* Diplomarbeit, Universität Kaiserslautern.

KNOBLOCK, C.A UND YANG, Q. 1994. Evaluating the Tradeoffs in Partial-Order Planning Algorithms. *Aus: Proceedings of the Canadian AI Conference.*

MCALLESTER, D. UND ROSENBLITT, D. 1991. Systematic Nonlinear Planning. *Seiten 634–639 aus: Proceedings of AAAI-91.*

MINTON, S. 1988. *Learning Search Control Knowledge: An Explanation-Based Approach.* Boston: Kluwer Academic Publishers.

MITCHELL, T.M., KELLER, R.M. UND KEDAR-CABELLI, S.T. 1986. Explanation-Based Generalization: A Unifying View. *Machine Learning*, **1**, 47–80.

MUÑOZ-AVILA, H., PAULOKAT, J. UND WESS, S. 1995. Controlling non-linear hierarchical planning by case replay. *Aus:* KEANE, M., HALTON, J.P. UND MANAGO, M. (Hrsg.), *Advances in Case-Based Reasoning. Selected Papers of the 2nd European Workshop (EWCBR-94).* Lecture Notes in Artificial Intelligence, Nr. 984. Springer.

NIEHAUS, A. UND LORSCHEID, S. 1995. *Eine Client-Server Kommunikationsschnittstelle zwischen Smalltalk und Prolog für KI-Anwendungen.* Projektarbeit, Universität Kaiserslautern.

NILSSON, N.J. 1980. *Principles of Artificial Intelligence.* Tioga, Palo Alto: Tioga Publishing Co.

PENBERTHY, J.S. UND WELD, D.S. 1992. UCPOP: A Sound, Complete, Partial Order Planner for ADL. *Aus: Proceedings of KR-93.*

PEOT, M. UND SMITH, D. 1993. Threat-removal strategies for partial-order planning. *Seiten 492–499 aus: Proceedings of AAAI-93.*

PETRIE, CH. 1991. *Planning and Replanning with Reason Maintenance.* Dissertation, University of Texas at Austin, CS Dept.

RICHTER, M.M. 1992. *Prinzipien der künstlichen Intelligenz.* 2. Auflage. Stuttgart: B.G. Teubner.

ROSENBLOOM, P.S. UND LAIRD, J.E. 1986. Mapping Explanation-Based Generalization onto SOAR. *Aus: Proceedings of the National Conference on Artificial Intelligence.*

ROTH-BERGHOFER, T. 1995. *Das Modal Truth Criterion für partiell geordnete Pläne und seine Realisierung im Planungswerkzeug CAPlan.* Projektarbeit, Universität Kaiserslautern.

SCHIRP, W. 1995. *Konzeption und Implementierung einer Komponente zur regelbasierten Ausfwahl von Konfigurationsschritten.* Diplomarbeit, Universität Kaiserslautern.

WEBERSKIRCH, F. 1994. *Realisierung eines nichtlinearen Planungssystems zur Unterstützung der Arbeitsplanerstellung bei der computerintegrierten Fertigung (CIM).* Diplomarbeit, Universität Kaiserslautern.

WEBERSKIRCH, F. 1995. *Combining SNLP-like Planning and Dependency-Maintenance.* Bericht LSA-95-10E. Centre for Learning Systems and Applications, University of Kaiserslautern, Germany.

WIELEMAKER, JAN. 1993. *SWI-Prolog 1.8.* University of Amsterdam, Dept. of Social Science Informatics (SWI), Roeterstraat 15, 1018 WB Amsterdam.

WINSTON, P.H., BINFORD, T.O., KATZ, B. UND LOWRY, M. 1983. Learning physical descriptions from functional definitions. *Seiten 433–439 aus: Proceedings of NCAI-83.*