

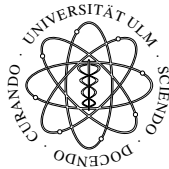
Intelligente Handlungsplanung

4. Deduktives Planen

Prof. Dr. Susanne Biundo-Stephan

Institut für Künstliche Intelligenz, Universität Ulm

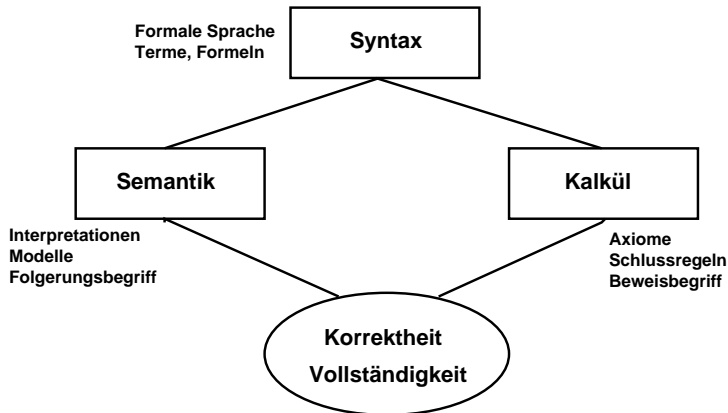
SS 2008



Inhalt

- 1 Exkurs: Formale Logik
- 2 Situationskalkül
- 3 Planen durch Beweisen
- 4 Planen durch Erfüllbarkeitstests

Exkurs: Formale Logik



Formale Logik 2

Syntax:

Der Syntax liegt eine Sprache erster Stufe zugrunde. Das *Alphabet* dieser Sprache ist:

$$A = V \cup J \cup Q \cup \{=\} \cup \{(\,,\,)\} \cup S$$

V : Menge von Variablensymbolen

J : Menge der logischen Verknüpfungen (Junktoren)

Q : Menge der Quantoren

S : Symbolmenge (Parameter) = $P \cup F$ mit

P : Menge der Prädikatensymbole

F : Menge der Funktionssymbole

Jedem Symbol aus S ist eine natürliche Zahl (Stelligkeit) zugeordnet.

Die Ausdrücke der logischen Sprache werden als *Terme* und *Formeln* bezeichnet und wie üblich induktiv definiert.

Formale Logik 3

Semantik

Deutung von Termen und Formeln in sogenannten Strukturen: (A, a)

- A : nichtleere Menge - Grundbereich oder Trägermenge
- a : auf S definierte Abbildung, die jedem n -stelligen Prädikat/-Funktionssymbol eine n -stellige Relation/-Funktion über A zuordnet.

Deutung von Variablen durch Belegungen $\beta : V \rightarrow A$. Formeln werden durch eine Interpretation I gedeutet mit $I = (\mathcal{A}, \beta)$, wobei \mathcal{A} eine Struktur ist.

Formale Logik 4

Modellbeziehung

- Normierung der umgangssprachlichen Junktoren
- Extensionaler Gebrauch, d.h. der Wahrheitswert zusammengesetzter Aussagen hängt ausschließlich von den Wahrheitswerten der Teilaussagen ab.

Eine Interpretation $I = (\mathcal{A}, \beta)$ ist Modell einer Formel φ (I erfüllt φ , φ gilt unter I , $I \models \varphi$) gdw. φ durch I in eine wahre Aussage übergeht.

Definition der Modellbeziehung induktiv über den Aufbau von Formeln (wie üblich).

Formale Logik 5

Definitionen

- Eine Formel φ folgt aus einer Formelmenge Φ ($\Phi \models \varphi$) gdw. jedes Modell von Φ auch ein Modell von φ ist.
- Eine Formel φ ist allgemeingültig (eine *Tautologie*) gdw. sie unter allen Interpretationen gilt.
- Eine Formel φ ist *erfüllbar* gdw. es mindestens eine Interpretation gibt, die Modell von φ ist.
- Eine Formel φ ist *unerfüllbar* gdw. sie nicht erfüllbar ist.
- Zwei Formeln φ und ψ sind *logisch äquivalent*, gdw. sie unter den selben Interpretationen gelten, d.h. $\varphi \models \psi$ und $\psi \models \varphi$, bzw. $\models \varphi \Leftrightarrow \psi$.

Satz: Für alle Formelmengen Φ und alle Formeln φ gilt $\Phi \models \varphi$ gdw. $\Phi \cup \{\neg\varphi\}$ ist unerfüllbar.

ACHTUNG: $(\Phi \models \neg\varphi) \Rightarrow (\Phi \not\models \varphi)$ aber nicht
 $(\Phi \not\models \varphi) \Rightarrow (\Phi \models \neg\varphi)$

Basis: Sortierte Prädikatenlogik

- Einführung von Sortensymbolen in die logische Sprache, als Ersatz für bestimmte einstellige Prädikatsymbole wie z.B. Block, Floor etc.
- Die Stelligkeit von Funktions- und Prädikatsymbolen wird durch Argument- und Zielsorten bestimmt, z.B. $on_{block,block}$.
- Eigenschaften müssen in Abhängigkeit von Situationen (zustandsabhängig) betrachtet werden.
- Operatoren beschreiben Zustandsübergänge.

Situationskalkül 1

(J. McCarthy 1963)

Prädikatausdrücke und Terme werden als sogenannte *fluents* betrachtet, die zustandsabhängig gedeutet werden.

⇒ Situationsparameter

Beispiel:

Σ : Menge von Sortensymbolen, z.B.: $\Sigma = \{block, floor, state\}$

$P = \{On_{block,block,state}, Clear_{block,state}, Onfloor_{block,state}\}$

Es gibt Konstanten verschiedenen Typs sowie Funktionssymbole zur Bezeichnung von Operatoren.

$F = \{A_{\epsilon,block}, B_{\epsilon,block}, S_{1_{\epsilon,state}}, \dots, move_{block,block,block,state,state}\}$

Situationskalkül 2

Situationsbeschreibung

$$\{On(A, B, S_1) \wedge Onfloor(B, S_1) \wedge Clear(A, S_1)\}$$

Beschreibung der Operatoren

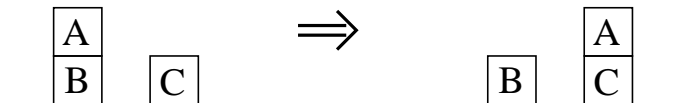
Ein Operatoraxiom ist eine geschlossene Formel der Form

$$\forall \dots [Vorbereitung \rightarrow Nachbedingung].$$

z.B. $\forall x, y, z : block \forall s : state$

$$\begin{aligned} & [[On(x, y, s) \wedge Clear(x, s) \wedge Clear(z, s)] \rightarrow \\ & [On(x, z, move(x, y, z, s)) \wedge Clear(y, move(x, y, z, s)) \\ & \wedge \neg Clear(z, move(x, y, z, s)) \wedge \neg On(x, y, move(x, y, z, s))]] \end{aligned}$$

Formulierung von Planungsproblemen



Ein Planungsproblem wird als eine Existenzaussage formuliert:

$$\begin{aligned} \varphi: & [[On(A, B, S_1) \wedge Clear(A, S_1) \wedge Clear(C, S_1) \\ & \wedge Onfloor(B, S_1) \wedge Onfloor(C, S_1)] \\ & \rightarrow \exists s : state[On(A, C, s) \wedge Onfloor(C, s) \wedge Onfloor(B, s)]] \end{aligned}$$

Deduktives Planen 1

Erzeugung von Plänen durch Beweise

Basis:

- Logische Sprache
- Kalkül

Die **Wissensbasis** ist eine Menge nichtlogischer Axiome. Sie enthält **Aktionsaxiome**, die die Effekte aller möglichen Aktionen beschreiben, sowie **Rahmenaxiome** und evtl. sogenannte **Weltgesetze** (domain constraints).

Planspezifikationen beschreiben Anfangs- und Zielzustand und fordern die Existenz eines Planes, der die beiden ineinander überführt.

Pläne entstehen durch den **konstruktiven Beweis** von Planspezifikationen.

Wichtig: Damit sind Pläne immer **beweisbar korrekt** bzgl. ihrer Spezifikation.

Deduktives Planen 2

Beispiel:

- Sortierter Situationskalkül
- Resolution

Situationskalkül: Prädikatenlogische Sprache, in der alle Funktions- und Prädikatsymbole ein zusätzliches Situationsargument haben.

Planungsverfahren: Ein Resolutionsbeweiser für Prädikatenlogik

Zustandsbeschreibungen: Formeln, z.B.: $Holding(a, s_0) \wedge On(b, c, s_0)$

Operatorbeschreibungen: Formeln, die den Zusammenhang zwischen Zuständen und möglichen Folgezuständen beschreiben

- Aktionsaxiome
- Rahmenaxiome

Deduktives Planen 3

Das Aktionsaxiom für *unstack*:

$Op_u: \forall x, y : block \forall s : state$

$$\begin{aligned} & [[Handempty(s) \wedge x \neq y \wedge On(x, y, s) \wedge Clear(x, s)] \\ & \Rightarrow [Holding(x, unstack(x, y, s)) \wedge Clear(y, unstack(x, y, s)) \wedge \\ & \quad \neg Handempty(unstack(x, y, s)) \wedge \neg On(x, y, unstack(x, y, s)) \wedge \\ & \quad \neg Clear(x, unstack(x, y, s))]] \end{aligned}$$

Ein Rahmenaxiom für *unstack*:

$$\begin{aligned} & \forall x, y, u, v : block \forall s : state \\ & [[u \neq x \wedge v \neq y \wedge On(u, v, s)] \Rightarrow On(u, v, unstack(x, y, s))] \end{aligned}$$

Beachte

- Aktionsnamen sind Funktionssymbole.
- Aktionen und Pläne werden durch Terme dargestellt.
- Operatorbeschreibungen sind ausdrucksmächtiger als in STRIPS.
 - ▶ Negation in Vor- und Nachbedingungen
 - ▶ Gleichheit
 - ▶ Quantifizierung

Deduktives Planen 4

Planspezifikationen

Existenzaussage der Form: Zu einem gegebenen Anfangszustand s gibt es einen bestimmten Zielzustand.

PS: $\forall u, v, w : \text{block}$

$$\begin{aligned} & [[u \neq v \wedge v \neq w \wedge u \neq w \wedge \\ & \text{Holding}(u, s_0) \wedge \text{On}(v, w, s_0) \wedge \text{Clear}(v, s_0)] \\ & \Rightarrow \exists t : \text{state Holding}(v, t)] \end{aligned}$$

Planen

- Beweis der Planspezifikation.
- Extraktion des Planes (= lösender Term für die existenzquantifizierte Zustandsvariable t) aus den Substitutionen für t .

Beispiel 1

Gegeben sei folgendes Planungsproblem:

Op_u, Op_{pd}, PS , wobei Op_u und PS wie oben angegeben und

$Op_{pd}: \forall x : block \forall s : state$

[$Holding(x, s) \Rightarrow$

$[Handempty(x, putdown(x, s)) \wedge$

$Ontable(x, putdown(x, s)) \wedge$

$Clear(x, putdown(x, s)) \wedge$

$\neg Holding(x, putdown(x, s))$

Zeige: $Op_u, Op_{pd} \models PS$

Durch: $KNF(Op_u \wedge Op_{pd} \wedge \neg PS) \vdash []$

Beispiel 2

Herstellung der Klauselnormalform

$Op_U: \{C1, \dots, C5\}$

$C1: \{\neg Handempty(s), \neg On(x, y, s), \neg Clear(x, s), Holding(x, unstack(x, y, s))\}$

$C2: \{\neg Handempty(s), \neg On(x, y, s), \neg Clear(x, s), Clear(y, unstack(x, y, s))\}$

$C3: \{\neg Handempty(s), \neg On(x, y, s), \neg Clear(x, s), On(x, y, unstack(x, y, s))\}$

$C4: \{\neg Handempty(s), \neg On(x, y, s), \neg Clear(x, s),$
 $\quad \neg Handempty(unstack(x, y, s))\}$

$C5: \{\neg Handempty(s), \neg On(x, y, s), \neg Clear(x, s), \neg Clear(x, unstack(x, y, s))\}$

$Op_{pd}: \{C6, \dots, C9\}$

$C6: \{\neg Holding(x, s), Handempty(putdown(x, s))\}$

$C7: \{\neg Holding(x, s), Ontable(x, putdown(x, s))\}$

$C8: \{\neg Holding(x, s), Clear(x, putdown(x, s))\}$

$C9: \{\neg Holding(x, s), \neg Holding(x, putdown(x, s))\}$

Beispiel 3

\neg PS: $\{T1, \dots, T4\}$

T1: $\{Holding(sk_1, s_0)\}$

T2: $\{On(sk_2, sk_3, s_0)\}$

T3: $\{Clear(sk_2, s_0)\}$

T4: $\{\neg Holding(sk_2, t)\}$

\Rightarrow 4 unäre Klauseln

Zur **automatischen Akkumulation von Antwortsubstitutionen** wird in die Klauseln der negierten Planspezifikation ein Antwortliteral eingefügt, bestehend aus dem einstelligen Prädikatsymbol A und derjenigen Variable, für die eine Antwortsubstitution gesucht wird (hier t).

In unserem Beispiel wird das Antwortliteral nur in die Klausel T4 eingefügt.

Die Klauseln T1 - T3 sind variablenfrei, insbesondere kommt die Variable t nicht in ihnen vor.

Das bedeutet, ein Resolutionsschritt mit Literalen aus diesen Klauseln erfordert keine Substitution, die t betrifft.

Beispiel 4

Beweis

C6: $\{\neg \text{Holding}(x, s), \text{Handempty}(\text{putdown}(x, s))\}$

T1: $\{\text{Holding}(sk_1, s_0)\}$

R1: $\{\text{Handempty}(\text{putdown}(sk_1, s_0))\}$

$\Rightarrow \sigma_1 = [x/sk_1, s/s_0]$

C1: $\{\neg \text{Handempty}(s), \neg \text{On}(x, y, s), \neg \text{Clear}(x, s),$
 $\text{Holding}(x, \text{unstack}(x, y, s))\}$

T4: $\{\neg \text{Holding}(sk_2, t), A(t)\}$

R2: $\{\neg \text{Handempty}(s), \neg \text{On}(sk_2, y, s), \neg \text{Clear}(sk_2, s),$
 $A(\text{unstack}(sk_2, y, s))\}$

$\Rightarrow \sigma_2 = [x/sk_2, t/\text{unstack}(sk_2, y, s)]$

Beispiel 5

R2: $\{\neg \text{Handempty}(s), \neg \text{On}(sk_2, y, s), \neg \text{Clear}(sk_2, s), A(\text{unstack}(sk_2, y, s))\}$

R1: $\{\text{Handempty}(\text{putdown}(sk_1, s_0))\}$

R3: $\{\neg \text{On}(sk_2, y, \text{putdown}(sk_1, s_0)), \neg \text{Clear}(sk_2, \text{putdown}(sk_1, s_0)), A(\text{unstack}(sk_2, y, \text{putdown}(sk_1, s_0)))\}$

$\Rightarrow \sigma_3 = [s/\text{putdown}(sk_1, s_0)]$

Die leere Klausel kann nur mit Hilfe von Rahmenaxiomen für *putdown* abgeleitet werden.

$$\forall x, y, u : \text{block} \forall s : \text{state} [\text{On}(x, y, s) \Rightarrow \text{On}(x, y, \text{putdown}(u, s))]$$
$$\forall x, y : \text{block} \forall s : \text{state} [\text{Clear}(x, s) \Rightarrow \text{Clear}(x, \text{putdown}(y, s))]$$

RA1: $\{\neg \text{On}(x, y, s), \text{On}(x, y, \text{putdown}(u, s))\}$

RA2: $\{\neg \text{Clear}(x, s), \text{Clear}(x, \text{putdown}(y, s))\}$

Beispiel 6

R3: $\{\neg \text{On}(sk_2, y, \text{putdown}(sk_1, s_0)), \neg \text{Clear}(sk_2, \text{putdown}(sk_1, s_0)),$
 $A(\text{unstack}(sk_2, y, \text{putdown}(sk_1, s_0)))\}$

RA1: $\{\neg \text{On}(x, y, s), \text{On}(x, y, \text{putdown}(u, s))\}$

R4: $\{\neg \text{On}(sk_2, y, s_0), \neg \text{Clear}(sk_2, \text{putdown}(sk_1, s_0))$
 $A(\text{unstack}(sk_2, y, \text{putdown}(sk_1, s_0)))\}$

$\Rightarrow \sigma_4 = [x/sk_2, u/sk_1, s/s_0]$

R4: $\{\neg \text{On}(sk_2, y, s_0), \neg \text{Clear}(sk_2, \text{putdown}(sk_1, s_0))$
 $A(\text{unstack}(sk_2, y, \text{putdown}(sk_1, s_0)))\}$

RA2: $\{\neg \text{Clear}(x, s), \text{Clear}(x, \text{putdown}(y, s))\}$

R5: $\{\neg \text{Clear}(sk_2, s_0), \neg \text{On}(sk_2, sk_1, s_0),$
 $A(\text{unstack}(sk_2, sk_1, \text{putdown}(sk_1, s_0)))\}$

$\Rightarrow \sigma_5 = [x/sk_2, y/sk_1, s/s_0]$

Beispiel 7

R5: $\{\neg \text{Clear}(sk_2, s_0), \neg \text{On}(sk_2, y, s_0),$
 $A(\text{unstack}(sk_2, y, \text{putdown}(sk_1, s_0)))\}$

T3: $\{\text{Clear}(sk_2, s_0)\}$

R6: $\{\neg \text{On}(sk_2, y, s_0),$
 $A(\text{unstack}(sk_2, y, \text{putdown}(sk_1, s_0)))\}$

R6: $\{\neg \text{On}(sk_2, y, s_0),$
 $A(\text{unstack}(sk_2, y, \text{putdown}(sk_1, s_0)))\}$

T2: $\{\text{On}(sk_2, sk_3, s_0)\}$

[]: $\{A(\text{unstack}(sk_2, sk_3, \text{putdown}(sk_1, s_0)))\}$

$\Rightarrow \sigma = [y/sk_3]$

Beispiel 8

Bei voll instantiierten Planspezifikationen können die Pläne unmittelbar aus den entsprechenden Beweisen gewonnen werden (Argument des Antwortliterals).

Bei nicht voll instantiierten Planspezifikationen enthalten die extrahierten Pläne Skolemkonstanten. Diese müssen in einem **Reskolemisierungsschritt** wieder durch Variablen ersetzt werden, um einen der Planspezifikation entsprechenden, variablen Plan zu erhalten.

Beispiel 9

Aus dem Beweis der Planspezifikation

PS: $\forall u, v, w : \text{block}$

$$\begin{aligned} & [[\text{Holding}(u, s_0) \wedge \text{On}(v, w, s_0) \wedge \text{Clear}(v, s_0)] \\ & \Rightarrow \exists t : \text{state Holding}(v, t)] \end{aligned}$$

hat man eine Substitution

$$\sigma = [t / (\text{unstack}(sk_2, sk_3, \text{putdown}(sk_1, s_0)))]$$

für die Zustandsvariable t erzeugt.

Durch “rückgängig machen” der Skolemisierung

$$[u / sk_1, v / sk_2, w / sk_3]$$

erhält man schließlich als Lösung den Plan

$$\text{unstack}(v, w, \text{putdown}(u, s_0)).$$

Konstruktion bedingter Pläne 1

Beispiel:

Definition eines neuen Operators *take*

$Op_t: \forall x: block \forall s: state$

$$[\neg Clear(x, s) \Rightarrow Holding(x, take(x, s))]$$

Operator *unstack* wie oben

Planspezifikation

PS': $\forall u, v: block$

$$[[Handempty(s_0) \wedge On(u, v, s_0)]$$

$$\Rightarrow \exists t: state Holding(u, t)]$$

Klauselmenge

$Op_u: \{C1, \dots, C5\}$ wie oben

$Op_t: \{C10\}$

$C10: \{Clear(x, s), Holding(x, take(x, s))\}$

Konstruktion bedingter Pläne 2

Theoremklauseln

\neg PS': $\{T1', \dots, T3'\}$

T1': $\{Handempty(s_0)\}$

T2': $\{On(sk_1, sk_2, s_0)\}$

T3': $\{\neg Holding(sk_1, t), A(t)\}$

Skolemisierung: $[u/sk_1, v/sk_2]$

Beweis:

C10: $\{Clear(x, s), Holding(x, take(x, s))\}$

T3': $\{\neg Holding(sk_1, t), A(t)\}$

R1: $\{Clear(sk_1, s), A(take(sk_1, s))\}$

$\Rightarrow \sigma = [x/sk_1, t/take(sk_1, s)]$

Konstruktion bedingter Pläne 3

C1: $\{\neg Handempty(s), \neg On(x, y, s), \neg Clear(x, s),$
 $*Holding(x, unstack(x, y, s))\}*$

T1: $\{Handempty(s_0)\}$

R2: $\{\neg On(x, y, s_0), \neg Clear(x, s_0),$
 $*Holding(x, unstack(x, y, s_0))\}*$

$\Rightarrow \sigma = [s/s_0]$

R2: $\{\neg On(x, y, s_0), \neg Clear(x, s_0),$
 $*Holding(x, unstack(x, y, s_0))\}*$ T2': $\{On(sk_1, sk_2, s_0)\}$

R3: $\{\neg Clear(sk_1, s_0),$
 $*Holding(sk_1, unstack(sk_1, sk_2, s_0))\}*$

$\Rightarrow \sigma = [x/s_1, y/sk_2]$

Konstruktion bedingter Pläne 4

R3: $\{\neg \text{Clear}(sk_1, s_0), \text{Holding}(sk_1, \text{unstack}(sk_1, sk_2, s_0))\}$

T3': $\{\neg \text{Holding}(sk_1, t), A(t)\}$

R4: $\{\neg \text{Clear}(sk_1, s_0), A(\text{unstack}(sk_1, sk_2, s_0))\}$

$\Rightarrow \sigma = [t/\text{unstack}(sk_1, sk_2, s_0)]$

R4: $\{\neg \text{Clear}(sk_1, s_0), A(\text{unstack}(sk_1, sk_2, s_0))\}$

R1: $\{\text{Clear}(sk_1, s), A(\text{take}(sk_1, s))\}$

R5: $\{A(\text{unstack}(sk_1, sk_2, s_0)), A(\text{take}(sk_1, s))\}$

$\Rightarrow \sigma = [s/sk_0]$

Damit ist die Planspezifikation PS' bewiesen.

R5 liefert eine **disjunktive Antwort**:

Lösungsterme (nach Reskolemisierung) sind

$\text{unstack}(u, v, s_0)$ bzw. $\text{take}(u, s_0)$

Konstruktion bedingter Pläne 5

Die unmittelbare Konstruktion eines Planes aus den Antwortsubstitutionen ist nicht möglich.

⇒ Inspektion bestimmter Beweisschritte

⇒ Generierung **bedingter Pläne**

Mit Hilfe einer Fallunterscheidung, die aus den Elternklauseln von R5 gewonnen wird, entsteht folgender Plan:

```
if  $Clear(u, s_0)$  then  $unstack(u, v, s_0)$   
if  $\neg Clear(u, s_0)$  then  $take(u, s_0)$ 
```

bzw.

```
if  $Clear(u, s_0)$  then  $unstack(u, v, s_0)$   
          else  $take(u, s_0)$  fi
```

Exkurs Programmsynthese 1

Durch den Beweis der Planspezifikation PS' wird implizit ein neuer, **definierter Operator** erzeugt.

⇒ Plan- bzw. **Programmsynthese**

In unserem Beispiel:

Beschreibung des neuen Operators mit Hilfe

- der Operatoren *unstack* und *take*
- des Plan- bzw. **Programmkonstruktors**
if ... then ... else

Explizit gemacht wird eine solche Operatordefinition, indem man für die existenzquantifizierte Zustandsvariable in einer Planspezifikation einen entsprechenden Skolemterm einsetzt.

Exkurs Programmsynthese 2

PS: $\forall u, v : \text{block} \forall s : \text{state}$
[$[\text{Handempty}(s) \wedge \text{On}(u, v, s)]$
 $\Rightarrow \exists t : \text{state} \text{Holding}(u, t)$]

Setze für t den Skolemterm $f(u, v, s)$ ein und betrachte das **if ... then ... else** - Konstrukt des bedingten Planes als den Rumpf von f :
 $f(u, v, s) \leftarrow$

if $\text{Clear}(u, s)$ **then** $\text{unstack}(u, v, s)$
else $\text{take}(u, s)$ **fi**

Damit hat man einen Operator erzeugt, der die Planspezifikation erfüllt.

Exkurs Programmsynthese 3

- Synthese **rein funktionaler** Programme, d.h. keine Seiteneffekte.
- Programmkonstrukte sind die **Hintereinanderausführung** (durch Funktionalkomposition) und die vollständige Fallunterscheidung (**if ... then ... else**).
- **Rekursive Programme** können nur durch Verwendung geeigneter **Induktionsaxiome** erzeugt werden. Dazu ist die explizite Einführung der Skolemfunktion notwendig.

Fazit:

Plangenerierung

Planspezifikation (Existenzaussage)

⇒ Plan bzw. definierter Operator

Programmsynthese

Programmspezifikation (Existenzaussage)

⇒ Programm

Rahmenproblem 1

frame problem

Planungsprobleme können i.d.R. nur dann gelöst werden, wenn das Domänenmodell sogenannte *Rahmenaxiome* enthält.

Rahmenaxiome (*frame axioms*)

Spezifikation *persistierender* Eigenschaften:

Was läßt eine Aktion *unverändert*?

Domänenmodell

- ⇒ Effektaxiome
- ⇒ Rahmenaxiome
- ⇒ Domain Constraints

Kontrolle

Verwendung der unterschiedlichen Axiome im Planungsprozess

Rahmenproblem 2

Praktikabilität

- vollständige Erfassung
- Größe des Suchraumes
- Steuerung des Suchprozesses

⇒ Ist diese formallogische Repräsentation von Anwendungsdomäne und Planungsproblem die geeignete ?

Lösung des Rahmenproblems in STRIPS
... mit GRAPHPLAN

Qualifikationsproblem 1

Die Operatoraxiome geben an, welche *Vorbedingungen* erfüllt sein müssen, damit ein Operator angewandt werden kann und die spezifizierten Effekte erzeugt.

Beispiel:

$Op_u: \forall x, y : block \forall s : state$

$[[Handempty(s) \wedge On(x, y, s) \wedge Clear(x, s)]$

$\Rightarrow [Holding(x, unstack(x, y, s)) \wedge Clear(y, unstack(x, y, s)) \wedge$
 $\neg Handempty(unstack(x, y, s)) \wedge \neg On(x, y, unstack(x, y, s)) \wedge$
 $\neg Clear(x, unstack(x, y, s))]]$

Zusätzlich muss gelten:

- x ist nicht zu schwer
- Die Greifhand ist korrekt justiert
- x läßt sich von y lösen
- ...

Qualifikationsproblem 2

Die erfolgreiche Ausführung eines jeden Operators hängt also prinzipiell von *unendlich vielen* Bedingungen - **Qualifikationen** - ab.

Frage:

Wie vermeidet man unendliche Qualifikationen in den Vorbedingungen von Operatoren?

⇒ Finden eines *adäquaten* Modells

- Auswahl der wichtigsten Merkmale
- Aufführung aller relevanten Bedingungen in der Operatorbeschreibung

Maßnahmen:

⇒ Kopplung mit *reaktiven* Systemen

⇒ *Replanning*

Verzweigungsproblem 1

ramification problem

Effektaxiome \Rightarrow “*unmittelbare*” Effekte

Rahmenaxiome \Rightarrow *persistierende* Eigenschaften

Häufig möchte man zusätzlich auf *Seiteneffekte* zurückgreifen.

Ableitbare und implizite Effekte

Bewegung von Objekten

$[At(Box, R1, S) \rightarrow At(Box, R2, move(Box, R1, R2, S))]$

Ein Rahmenaxiom:

$\forall s : state \forall obj : object \forall r_1, r_2 : room \forall b : box$

$[In(obj, b, s) \rightarrow In(obj, b, move(b, r_1, r_2, s))]$

Domain Constraints:

$\forall s : state \forall obj : object \forall r : room \forall b : box$

$[[In(obj, b, s) \wedge At(b, r, s)] \rightarrow At(obj, r, s)]$

$\forall s : state \forall r_1, r_2 : room \forall b : box$

$[[At(b, r_1, s) \wedge At(b, r_2, s)] \rightarrow r_1 = r_2]$

Verzweigungsproblem 2

Die Verwendung von *Domain Constraints* erlaubt eine *kompakte* Beschreibung der Effekte von Operatoren.

Weitere zur Beweiskonstruktion benötigte Eigenschaften können *bei Bedarf* abgeleitet werden.

Fragen:

- Was wird über Effekte, was über Weltgesetze modelliert ?
- Welche sind die relevanten Eigenschaften ?

Erfüllbarkeitsbasiertes Planen

Grundprinzip:

- *Kodierung* des Planungsproblems in eine aussagenlogische Formel
 - Konstruktion eines *Modells*, das die Formel erfüllt
⇒ *erfüllende Belegung*
 - *Extraktion* des Planes aus dieser Belegung
- ⇒ *erfüllbarkeitsbasiertes Planen*
satisfiability-based (SAT-based) Planning

Aussagenlogische Erfüllbarkeit 1

Die Grundbausteine der Aussagenlogik sind atomare Aussagen:
Propositionen

Syntax: abzählbares Alphabet atomarer Formeln
(aussagenlogische Variablen) P, Q, R, \dots
logische Konnektoren ($\wedge, \vee, \rightarrow, \dots$)

Semantik: Interpretation aussagenlogischer Formeln
durch Belegung der aussagenlogischen
Variablen mit Wahrheitswerten:
true bzw. *false*

Erfüllbarkeit: Eine aussagenlogische Formel φ ist
erfüllbar gdw. es eine Belegung der
Variablen in φ gibt, so daß φ wahr wird.
 \Rightarrow Wahrheitstafelmethode

Aussagenlogik ist *entscheidbar*.

Aussagenlogische Erfüllbarkeit 2

Gegeben:

Eine aussagenlogische Formel φ in *konjunktiver Normalform* ($\text{KNF}(\varphi)$)

Definitionen:

Eine Formel ist in *KNF*, wenn sie eine Konjunktion von Disjunktionen von Literalen ist.

⇒ Klauseldarstellung

Ein *Literal* ist eine *atomare* Formel oder eine *negierte* atomare Formel. *Einerklauseln* (unit clauses) sind Klauseln, die genau *ein* Literal enthalten.

Für jede Einerklausel $\langle L \rangle$ in $\text{KNF}(\varphi)$ muss L mit *true* belegt werden, damit $\text{KNF}(\varphi)$ erfüllt wird.

Kommt ein Atom Q in jedem Literal in $\text{KNF}(\varphi)$, das Q enthält, in derselben Polarität vor, so heißt Q bzw. $\neg Q$ *pures Literal* (pure literal).

Beispiel

$$\varphi:$$
$$(A \vee B \vee \neg E) \wedge (B \vee \neg C \vee D) \wedge (\neg A) \wedge (B \vee C \vee E) \wedge (\neg D \vee \neg E)$$

$$\varphi(\neg A):$$
$$(B \vee \neg E) \wedge (B \vee \neg C \vee D) \wedge (B \vee C \vee E) \wedge (\neg D \vee \neg E)$$

$$\varphi(B):$$
$$(\neg A) \wedge (\neg D \vee \neg E)$$

Der DPLL Algorithmus 1

(Davis/Putnam/Logemann/Loveland 1962)

- Tiefensuche durch den Raum der partiellen Belegungen
- Backtracking
- *unit-clause* – Heuristiken
- *pure-literal* – Heuristiken
- Divide & Conquer Strategie

Der DPLL Algorithmus 2

```
procedure DPLL(KNF( $\varphi$ ))
if ( $\varphi = \emptyset$ ) then return yes
else if ( $\varphi$  enthält die leere Klausel ( $[\ ] \in \varphi$ ))
  then return no
  else if (L ist pures Literal in  $\varphi$ )
    then return DPLL(SIM( $\varphi(L)$ ))
    else if ( $\{L\}$  ist Einerklausel in  $\varphi$ )
      then return DPPL(SIM( $\varphi(L)$ ))
      else choose Q Atom in  $\varphi$ 
        if DPLL(SIM( $\varphi(Q)$ )) then return yes
        else return DPLL(SIM( $\varphi(\neg Q)$ ))
```

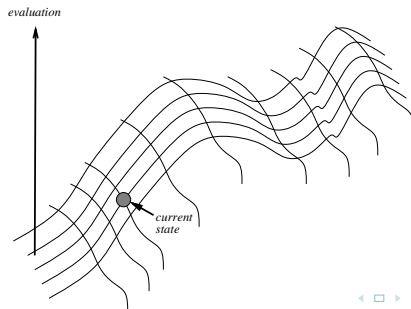
Exkurs: Lokale Suche

Lösung von Problemstellungen, bei denen nur der Zielzustand selbst relevant ist, weniger der “Weg zum Ziel” (8-Damen Problem, VLSI Design, TSP).

Lassen sich Zielzustände zudem durch ein *Qualitätsmaß* beschreiben, so kann man *lokale Suche* verwenden.

Idee:

Schrittweise Verbesserung einer zufällig gewählten Anfangskonfiguration



Hill Climbing 1

Das Bergsteigerprinzip: “Versuche immer weiter nach oben zu kommen.”

- Festlegung von Nachbarschaftsbeziehungen zwischen Zuständen
- Bestimmung eines Ausgangszustandes
- Schrittweiser Übergang zu benachbarten Zuständen besserer Qualität
- Ausgabe des besten Zustandes

```
function HILL-CLIMBING(problem) returns a solution state
  inputs: problem, a problem
  static: current, a node
           next, a node

  current ← MAKE-NODE(INITIAL-STATE[problem])
  loop do
    next ← a highest-valued successor of current
    if VALUE[next] < VALUE[current] then return current
    current ← next
  end
```

Hill Climbing 2

Probleme:

- *Lokale Maxima*: Der Algorithmus gibt einen lokal besten Zustand aus, der u.U. weit vom Optimum entfernt ist.
- *Plateaus*: Die Evaluierungsfunktion ist hier flach. Entscheidende Verbesserungen werden nicht mehr erreicht.
- *Grate*: Der Algorithmus oszilliert möglicherweise.

Lösung:

- *Neustart*, wenn keine Verbesserung mehr erreicht wird
- Durchführung mehrerer Iterationen und Aufheben des soweit besten Resultats

Simuliertes Abkühlen

Simulated Annealing erlaubt auch Schritte, die das bereits erzielte Ergebnis verschlechtern.

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state

inputs: *problem*, a problem

schedule, a mapping from time to “temperature”

static: *current*, a node

next, a node

T, a “temperature” controlling the probability of downward steps

current \leftarrow MAKE-NODE(INITIAL-STATE[*problem*])

for *t* \leftarrow 1 **to** ∞ **do**

T \leftarrow *schedule*[*t*]

if *T*=0 **then return** *current*

next \leftarrow a randomly selected successor of *current*

$\Delta E \leftarrow$ VALUE[*next*] – VALUE[*current*]

if $\Delta E > 0$ **then** *current* \leftarrow *next*

else *current* \leftarrow *next* only with probability $e^{\Delta E/T}$

Falls die Abkühlung langsam genug erfolgt, findet der Algorithmus ein globales Optimum.

Der GSAT Algorithmus 1

Stochastische Lösung von Erfüllbarkeitsproblemen

(Kautz & Selman 1992)

- nicht-erschöpfende, gierige Suche (greedy search)
- hill-climbing
- zufällige Auswahl von Belegungen für *alle* Atome (aussagenlogische Variablen) in der Formel
- Bestimmung der Anzahl von Klauseln, die unter dieser Belegung wahr werden
- Änderung der Belegung eines Atoms, so dass diese Anzahl maximal ansteigt (evtl. 0 oder negativ)

Der GSAT Algorithmus 2

- ⇒ Algorithmus kann unendlich lange auf einem “Plateau” herumwandern
 - ⇒ Terminierung nach einer gewissen Anzahl von Änderungen (max-flips)
- ⇒ Algorithmus kann mit einem lokalen Maximum terminieren
 - ⇒ Neustart mit anderer Zufallsbelegung (max-tries)

procedure GSAT(KNF(φ),max-tries,max-flips)

for i:=1 to max-tries do

 A := zufällige Belegung für φ

 for j:=1 to max-flips do

 if A erfüllt φ then return **yes**

 else choose Atom Q, so dass die Änderung der Belegung von Q den größten Zuwachs an erfüllten Klauseln erzeugt. Modifiziere A durch entsprechende Änderung der Belegung von Q

 end for

end for

Modifikationen von GSAT

Walksat:

(Kautz & Selman 1996)

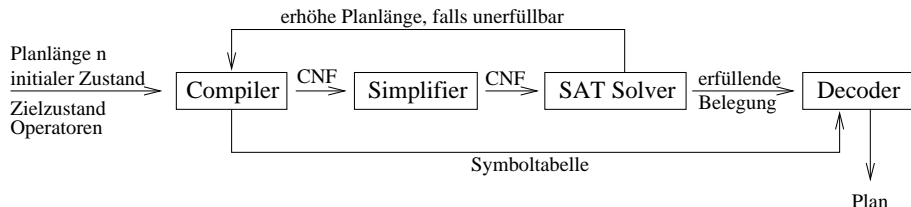
- erweiterte Zufallssteuerung
- wähle bei der Änderung einer Belegung mit einer Wahrscheinlichkeit p die Variable, die GSAT wählen würde, mit der Wahrscheinlichkeit $1 - p$ irgendeine Variable aus einer unerfüllten Klausel

Weitere Varianten:

Zufallssteuerung von DPLL

Laufzeit ist abhängig von der Auswahl der “splitting” Variable

SAT-basierte Planungssysteme



Übersetzung von Planungsproblemen

Ziel: "kleine" Formeln

Denn: Erfüllbarkeitstest ist exponentiell aufwändig

Größe von Formeln:

- Anzahl der Variablen
- Anzahl der Klauseln
- Anzahl der Literale je Klausel

Kodierung von Planungsproblemen 1

Planungsprobleme werden in STRIPS-Notation spezifiziert: (*init*, *goal*, *O*) wobei eine endliche Menge von Objekten unterstellt wird.

Kodierungsschema für Operatoren:

- Festlegen der *Planlänge* (N)
- Erweitern der Parameterlisten der Operatoren und Prädikate um *Zeitpunkte* (n).
- Bilden aller entsprechenden *Grundinstanzen* von Operatoraxiomen und Rahmenaxiomen (bzw. Folgezustandsaxiomen).

Beispiel:

Kodierung von Planungsproblemen 2

Kodierung von Operatoraxiomen:

- Operatorname + Parameterliste + Zeitpunkt
⇒ Prädikatausdruck $Op(\bar{c},t)$
- $Op(\bar{c},t) \Rightarrow Prec[t]$
- $Op(\bar{c},t) \Rightarrow Effects[t+1]$
- Auffassung der entstandenen Grundatome als aussagenlogische Variablen.

Beispiel:

Objekttypen: Truck, Location, Cargo

$move(Truck37,Paris,London,1) \Rightarrow at(Truck37,Paris,1)$

$move(Truck37,Paris,London,1) \Rightarrow at(Truck37,London,2) \wedge$
 $\neg at(Truck37,Paris,2)$

Aufwand

Jede Operatorgrundinstanz wird durch eine Variable repräsentiert.

⇒ $N * |\text{Schemata}| * |\text{Dom}|^P$ Variablen

| | |
|----------|--|
| N | Planlänge |
| Schemata | Anzahl der Operatorschemata |
| Dom | Anzahl der Domänenobjekte |
| P | maximale Anzahl von Parametern je Operator |

Der Aufwand zur systematischen Lösung von Erfüllbarkeitsproblemen ist exponentiell in der Anzahl der Variablen.

Auch bei stochastischen Verfahren spielt diese Anzahl eine große Rolle.

⇒ Reduktion der Variablenmenge durch Einführung von k zweistelligen Prädikatausdrücken für jeden k-stelligen Operator.

⇒ Anzahl der benötigten Variablen reduziert sich auf $N * |\text{Schemata}| * |\text{Dom}| * P$

Beispiel

Statt $\text{move}(\text{Truck37}, \text{Paris}, \text{London}, 1)$:

$\text{move}_1(\text{Truck37}, 1)$, $\text{move}_2(\text{Paris}, 1)$, $\text{move}_3(\text{London}, 1)$

Jedoch:

Jede Operatorinstanz $\text{Op}(\bar{c}, i)$ wird nun durch eine *Konjunktion* von Atomen P_{op} für $1 \leq i \leq n$ beschrieben ($op = |c|$).

Weitere Optimierungen

- Verwendung “erklärender” Rahmenaxiome (explanatory frame axioms)
- Aufzählung der Aktionen, die eine bestimmte Zustandsänderung verursacht haben können.

$$[[\text{at}(\text{Truck37}, \text{Paris}, t) \wedge \neg \text{at}(\text{Truck37}, \text{Paris}, t+1)] \\ \text{move}(\text{Truck37}, \text{London}, t) \vee \text{move}(\text{Truck37}, \text{Brussels}) \vee \dots]$$

- Kontraposition: Falls keine dieser verändernden Aktionen stattfindet, bleibt die Eigenschaft erhalten.
- Argumentbasierten Darstellung:

$$[[\text{at}(\text{Truck37}, \text{Paris}, t) \wedge \neg \text{at}(\text{Truck37}, \text{Paris}, t+1)] \\ \Rightarrow [[\text{move}_1(\text{Truck37}, t) \wedge \text{move}_2(\text{Paris}, t) \wedge \text{move}_3(\text{London}, t)] \\ [\vee \text{move}_1 \dots]]]$$

Faktorisierende Optimierung:

$$[[\text{at}(\text{Truck37}, \text{Paris}, t) \wedge \neg \text{at}(\text{Truck37}, \text{Paris}, t+1)] \\ \Rightarrow [\text{move}_1(\text{Truck37}, t) \wedge \text{move}_2(\text{Truck37}, \text{Paris})]]$$

- Kodierung des Anfangs- und Zielzustandes durch die Eigenschaften zum Zeitpunkt 0 bzw. N.

GRAPHPLAN-basierte Kodierung 1

Motivation:

- Aufbau von Planungsgraphen
- Kodierung von Planungsgraphen als aussagenlogische Formel
- Erfüllbarkeitstest und Lösungsextraktion

Kodierung eines Planungsgraphen:

Tiefe N

- Grundinstanzen von Operatoren und Fakten als aussagenlogische Variable
- Zeitpunkte sind die entsprechenden Fakten- bzw. Aktionenebenen
- Fakten des Start- bzw. Zielzustandes: Faktenebene 0 bzw. N
- Jedes Faktum einer Ebene $i+1$ ($i \geq 1$) impliziert die Disjunktion aller Aktionen der Aktionsebene i , die das Faktum als ADD-Effekt haben
- Aktionen $op(\bar{c},i)$ und $op'(\bar{c}',i)$ ($i \geq 1$), die sich gegenseitig ausschließen:
 $\neg op(\bar{c},i) \vee \neg op'(\bar{c}',i)$