DAI-Labor
TU Berlin

Grundlagen der Künstlichen Intelligenz

Planen

10.11.2005

Brijnesh J Jain, Stefan Fricke
bjj@dai-labor.de stefan.fricke@dai-labor.de

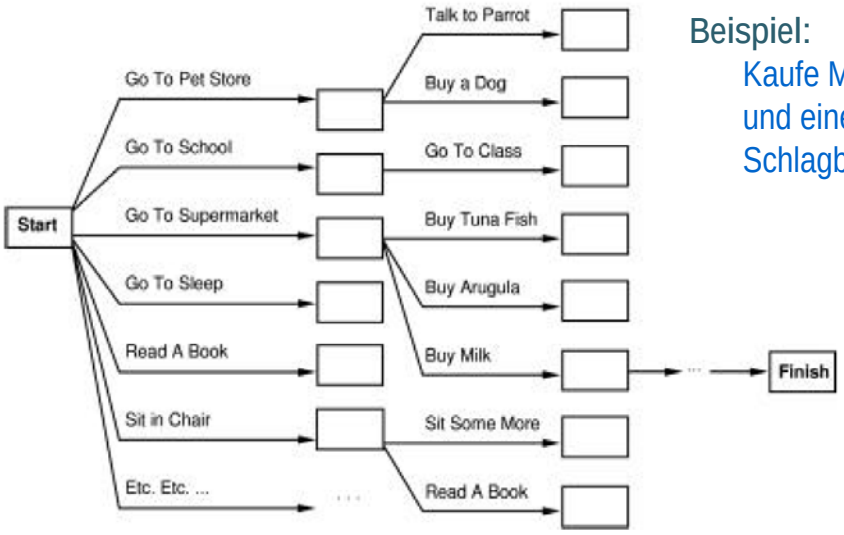
AIOT
Agententechnologien in betrieblichen Anwendungen und der Telekommunikation

Gliederung

- ⇨ Einführung
- ⇨ Klassische Planungssprache – STRIPS
- ⇨ Beispiele – STRIPS
- ⇨ Planungsalgorithmen
- ⇨ Menschliches Problemlösen (nach VL von Dr. Peter Geibel)
- ⇨ Zusammenfassung

- Planung beschäftigt sich mit der Formalisierung, Implementierung und Evaluierung von Algorithmen für die Konstruktion von Handlungsplänen.
- Plan:
  Folge von Aktionen, deren Ausführung einen Startzustand in einen vordefinierten Zielzustand überführt.
- Das sieht nach Problemlösen aus. Was ist der Unterschied?

Planung ist ein wichtiges Teilgebiet der KI.



Beispiel:

Kaufe Milch, Bananen und einen Schlagbohrer

Problemlösungsübung: Wir haben ein Ziel gegeben (Milch kaufen usw) und wollen das als Problem formalisieren und dann lösen. Dabei stellen wir fest, wie unpraktisch der Formalismus von Problemlösen in diesem Fall ist, d.h. wenn es viele mögliche Aktionen und Zustände gibt. Außerdem sind viele Aktionen gar nicht zielführend (z.B. Sit in Chair, Read a Book).

Einführung

Von Problemlösen zum Planen

- Problemraum: vollständige Zustandsbeschreibung.
- Aktionen generieren vollständige Zustandsbeschreibung.

- Probleme:
  - Zu viele Zustände sind zu betrachten.
  - Zu viele Aktionen sind zu betrachten, hoher Verzweigungsgrad führt zu schlechter Skalierung.
  - Irrelevante Aktionen können nicht eliminiert werden.

Irrelevante Aktionen können nicht eliminiert werden: Die Aktion „read a Book“ ist beispielsweise irrelevant, wenn ich Milch besorgen muss. Beim Problemlösen muss man alle Aktionen erst einmal anwenden, um festzustellen ob sie einen gewünschten Zustand („Milch haben“) liefern. Das liegt daran, dass Aktion und Zustand im Gegensatz zu Planungssprachen entkoppelt sind.

- ⇨ Frage: Wie löst man hoch skalierte Probleme?
- ⇨ Idee: (Kernidee des Planens):
  „*Öffne*“ Repräsentationen von Zielen, Zuständen und Aktionen
  - → Ziele und Zustände als Aussagen
  - → Aktionen als logische Beschreibungen mit Vorbedingungen und Effekten
  - → Direkte Verbindung zwischen Aktion und Zustand

Beim Problemlösen ist jeder Zustand vollständig beschrieben und Anwendung einer Aktion liefert einen vollständig beschriebenen Folgezustand. Beim Planen haben wir durch Vorbedingung und Effekt eine Verbindung zwischen Aktion und Zustand. Das ist die Öffnung.

- Sprache: Repräsentiert Zustände, Ziele und Aktionen
- Algorithmus: Erstellt einen Plan
- Anforderungen an Planungssysteme:
  - Genügend ausdruckstark um möglichst viele Probleme zu formalisieren
  - Genügend restriktiv, um möglichst effiziente Algorithmen anzuwenden
  - Planungsalgorithmus sollte logische Struktur des Problems ausnutzen

Unterschied zu den Suchverfahren: Dort sind die Algorithmen bekannt und es muss für jedes Problem eine geeignete Repräsentation gefunden werden. Hier findet die Repräsentation mittels einer formalen Sprache statt.

Es besteht ein Tradeoff zwischen Ausdrucksstärke und effizienter Verarbeitung. Beispielsweise ist die Prädikatenlogik (relativ) ausdrucksstark, aber gleichzeitig unentscheidbar. Auf der anderen Seite sind Graph- oder Baumstrukturen wenig ausdrucksmächtig, dafür existieren aber effiziente Algorithmen wie A* für diese Klasse von Problemen.

Planungsalgorithmus sollte logische Struktur des Problems ausnutzen, zum Beispiel Divide and Conquer, Partial Order, usw.

Gliederung

- ⇨ Einführung
- ⇨ Klassische Planungssprache – STRIPS
- ⇨ Beispiele – STRIPS
- ⇨ Planungsalgorithmen
- ⇨ Menschliches Problemlösen
- ⇨ Zusammenfassung

STRIPS ist die Grundlage aller Planungssysteme. Viele Schwächen von STRIPS werden in aktuelleren Planungssprachen und –algorithmen adressiert.

Klassische Planungssprache – STRIPS

- ⇨ STRIPS = Stanford Research Institute Problem Solver

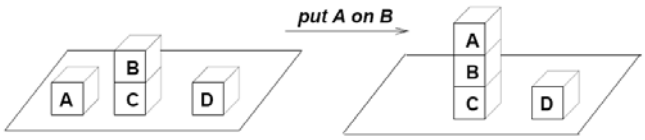
  (Fikes & Nilsson, 1971)

  - → Mengenbasiertes Planen (auf Mengen von Prädikaten)
  - → Klassischer Ansatz für Planungssysteme
  - → Grundprinzipien noch heute aktuell
  - → Restriktiver Formalismus; basiert auf Closed-World Assumption

Closed-World Assumption: Alles explizit beschriebene und logisch herleitbare Wissen wird als wahr angenommen, alles andere ist falsch.

STRIPS: Blockswelt

- Klassische Planungsdomäne
- Hier: Zur Illustration elementarer Definitionen
- Eigenschaften der Blockswelt
  - Menge von Blöcken, die auf einem Tisch liegen
  - Blöcke können gestapelt werden
  - Es liegt maximal ein Block *direkt* auf einem anderen Block



Klassische Planungssprache – STRIPS

Logische Sprache ist Tripel $L = (P, C, V)$ bestehend aus

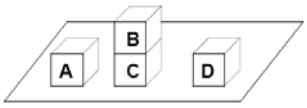
- ⇨ P : Menge von Prädikatensymbolen
  - → Jedes Prädikatensymbol besitzt eine Arität (Stelligkeit)
  - → Beispiel: $\{ ontable^1, clear^1, on^2 \}$
- ⇨ C : Menge von Konstantensymbolen
  - → Beispiel: $\{A, B, C, D\}$
- ⇨ V : Menge von Variablensymbolen
  - → Beispiel: $\{x, y, z, \ldots\}$
- ⇨ Logische Konnektoren: $\neg$ und $\wedge$

Bemerkungen:

1. Arität von Prädikatensymbolen in Beispielen durch hochgestellte Zahlen angedeutet
2. Keine Quantoren

Klassische Planungssprache – STRIPS

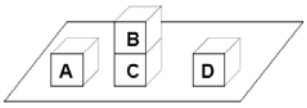
- ⇨ **Terme:** Konstanten und Variablen
- ⇨ **Atome:** Ausdrücke der Form $p(t_1, \ldots, t_n)$, wobei
  - → p = n-stelliges Prädikatensymbol
  - → t_i = Term für alle $1 \leq i \leq n$
  - → Beispiel: $ontable(A)$, $clear(x)$, $on(B,C)$
- ⇨ **Grundatome:** Atome, die keine Variablen enthalten
  - → Beispiel: $ontable(A)$, $on(B,C)$
- ⇨ **Literale:** Atome oder negierte Atome



Verschachtelte Ausdrücke wie p(q(A)) sind nicht gestattet.

STRIPS: Zustände

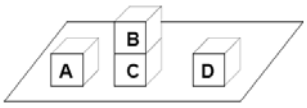
- ⇨ Konjunktion von positiven Grundatomen
- ⇨ Schreibweise: $p_1 \wedge p_2 \wedge \ldots \wedge p_n = \{ p_1, p_2, \ldots, p_n \}$
- ⇨ Aufgrund der Closed World Assumption werden nicht aufgeführte Literale als **false** aufgefasst
- ⇨ Beispiel:
  - → S = { ontable(A), ontable(C), ontable(D),
    clear(A), clear(B), clear(D), on(B,C) }



Veroderung findet in STRIPS nicht statt – es ist ja auch selten plausibel eine Situation als Veroderung von Teilsituationen zu interpretieren.

STRIPS: Ziele

- Konjunktion von positiven Grundatomen
- Typischerweise als partielle Zustandsbeschreibung
- Ein Zustand S ist ein Zielzustand Z genau dann, wenn $Z \subseteq S$
- Beispiel:
  - Z = { ontable(C), on(B,C) }



Ein Ziel ist nicht notwendigerweise eine vollständige Zustandsbeschreibung; die closed world assumption gilt hier also nicht. Mit dieser nur auf die relevanten Aspekte abzielenden Zielbeschreibung spart man sich viel Arbeit, denn ansonsten müssten sämtliche Zielzustände aufgeführt werden, so der in der Grafik angegebene mit A und B auf dem Tisch liegend, aber auch A auf B und B auf A liegend.

STRIPS: Aktionen

- ⇨ Ein Aktionsschema wird spezifiziert durch Vorbedingungen (PRE) und Effekte (EFF)
- ⇨ EFFekte werden repräsentiert durch ADD und DEL Listen
- ⇨ PRE, ADD sind Konjunktionen von *positiven Literalen*
- ⇨ DEL ist Konjunktion von *negativen Literalen*
- ⇨ Variablen werden als ∃-quantifiziert angenommen
- ⇨ Eine Aktion ist ein instantiiertes Aktionsschema

Vorbedingungen (PRE) müssen erfüllt sein, um eine Aktion auszuführen.

Effekte (EFF) beschreiben die Zustandsänderungen nach einer Aktion

Eine Aktion hat natürlich auch noch einen Namen.

Klassische Planungssprache – STRIPS

Repräsentation

Beispiel:

⇨ ACT: put(x,y)

PRE: ontable(x), clear(x), clear(y)

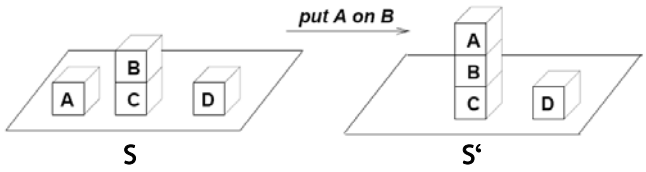
ADD: on(x,y)

DEL: ontable(x), clear(y)

Fragen:

⇨Ist put(x,y) auf S anwendbar?

⇨ Wie wird put(x,y) auf S angewendet?



Die beiden Kriterien der Anwendbarkeit (Test) und Ausführung (Handlung) werden nachfolgend erörtert.

STRIPS: Anwendbarkeit von Aktionen

⇨ Ein Aktionsschema **A** ist anwendbar für einen Zustand **S**, wenn **S** die Vorbedingung PRE von **A** erfüllt.

⇨ Die Anwendbarkeit von **A** für Zustand **S** wird durch Substitution θ der Variablen in PRE überprüft:

Substituiere alle Variablen von **A** durch Konstantensymbole

S erfüllt PRE von **A**, wenn $\theta PRE \subseteq S$

Häufig gibt es verschiedene Kombinationen von Variablensubstitutionen, sodass die Zahl der anwendbaren Aktionen sehr groß werden kann. Sie ist aber immer endlich, weil die Zahl der Konstantensymbole ebenfalls endlich ist.

STRIPS: Substitution

- Substituiere Variablen mit unterschiedlichen Namen mit unterschiedlichen Konstantensymbolen.
- Substituiere alle Variablen des Aktionsschemas,
  - d.h. auch Variablen in ADD und DEL.
- Ein vollständig instantiiertes Aktionsschema ist eine Aktion a
  - $\mathrm{PRE}(a)$, $\mathrm{ADD}(a)$, $\mathrm{DEL}(a)$ bezeichnen die jeweiligen Teilmengen von a

Gleiche Variablen werden natürlich mit gleichen Konstanten substituiert.

Beispiel: put(x,y) darf nicht zu put(A,A) substituiert werden, denn dann gälte on(A,A) (A liegt auf A) und das ist offensichtlich nicht möglich.

STRIPS: Anwendbarkeit von Aktionen

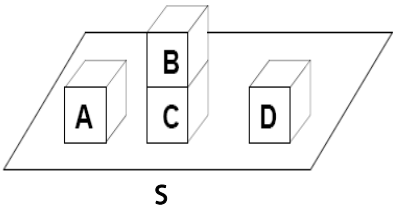
⇨ Beispiel: Aktion put(x,y) ist anwendbar auf Zustand S

S = { ontable(A), ontable(C), ontable(D), clear(A), clear(B), clear(D), on(B,C) } erfüllt die Vorbedingung

PRE: ontable(x), clear(x), clear(y)

von Aktion put(x,y) mit Substitution

$\theta = \{x \leftarrow A, y \leftarrow B\}$



S

ACT: put(x,y)
PRE: ontable(x), clear(x), clear(y)
ADD: on(x,y)
DEL: ontable(x), clear(y)

Eine 2. Substitution existiert (x ← D)

STRIPS: Anwendung von Aktionen

- Sei **S** ein Zustand und **a** ein vollständig instantiiertes Aktionsschema mit PRE(a) $\subseteq$ S
- Das Resultat einer Aktion **a**, ausgeführt auf einen Zustand **S**, ist ein Zustand **S'** mit

$$S' = S \cup ADD(a) - DEL(a)$$

- STRIPS Annahme:
  - Jedes nicht in ADD/DEL enthaltene Literal bleibt unverändert

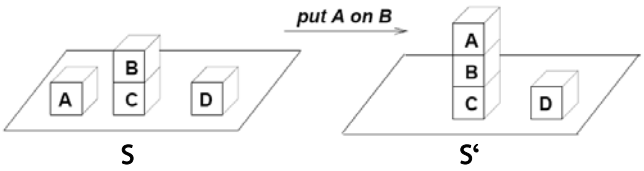
Der Folgezustand entspricht also dem aktuellen Zustand, nur dass die Propositionen der ADD-Liste hinzugefügt und die Ausdrücke der DEL-Liste entfernt werden. Da wir es mit Mengenoperationen zu tun haben, erscheinen Literale nicht doppelt im Folgezustand und ist das Entfernen eines Literals der DEL-Liste, das im Zustand gar nicht vorhanden ist, auch kein Fehler, sondern eine Null-Operation.

STRIPS: Aktionsbeispiel

Aktion put(A,B) angewendet auf Zustand

S = { ontable(A), ontable(C), ontable(D), clear(A), clear(B), clear(D), on(B,C) } liefert Zustand

S‘ = {ontable(C), ontable(D), clear(A), clear(D), on(B,C); on(A,B) }



ACT: put(A,B)
PRE: ontable(A), clear(A), clear(B)
ADD: on(A,B)
DEL: ontable(A), clear(B)

STRIPS: Formalisierung des Planungsproblems

- ⇨ STRIPS Planungsproblem: Tripel $(A, Z, \mathcal{A})$ mit
  - → A = Anfangszustand (Konjunktion von Literalen)
  - → Z = Zielzustand (Konjunktion von Literalen)
  - → $\mathcal{A}$ = Menge von Aktionsschemata
- ⇨ Plan:
  - → Eine Folge von Aktionen $p = (a_0, \ldots, a_m)$, sodass jede Aktion a_i anwendbar ist auf das Resultat der Aktion a_{i-1} für alle $1 \leq i \leq m$
- ⇨ Lösung: Plan, sodass
  - → a_0 anwendbar auf Anfangszustand A
  - → Jedes Literal von Zielzustand Z kommt auch im Folgezustand von a_m vor

Eine Lösung ist ein Zustand, der mindestens die Propositionen vom Zielzustand Z enthält. Es können natürlich noch weitere Dinge wahr sein, die hier aber nicht interessieren. Zum einen ist die Zielbeschreibung ja nur eine unvollständige Zustandsbeschreibung, zum andern können Aktionen auch so genannte Seiteneffekte produzieren (z.B. das „Erscheinen“ weiterer Blöcke). Seiteneffekte sind in der Realität häufig problematisch, da typischerweise ungewollt (z.B. die Nebenwirkung eines Medikaments, Kollateralschäden, etc.), aber das spielt in STRIPS keine Rolle.

Gliederung

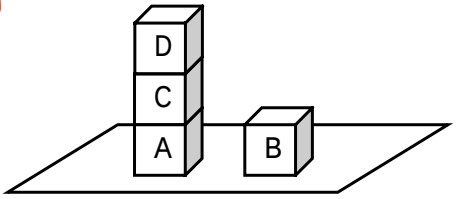
- ⇨ Einführung
- ⇨ Klassische Planungssprache – STRIPS
- ⇨ Beispiele – STRIPS
- ⇨ Planungsalgorithmen
- ⇨ Menschliches Problemlösen
- ⇨ Zusammenfassung

STRIPS-Beispiel: Blockswelt

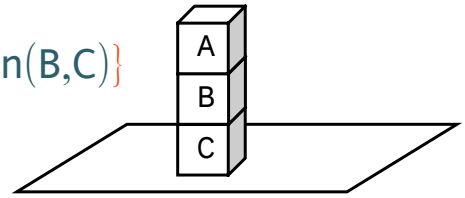
- Blockswelt beinhaltet
  - das Stapeln von Blöcken und das
  - Abbauen von Blockstapeln.
- Logische Sprache
  - $P = \{ontable^1, clear^1, on^2\}$
  - $C = \{A, B, C, D\}$
  - $V = \{x, y\}$

STRIPS-Beispiel: Blockswelt

- Anfangszustand A:
  - $A = \{on(D,C), on(C,A), clear(D), clear(B), ontable(A), ontable(B)\}$



- Zielzustand Z:
  - $Z = \{on(A,B), on(B,C)\}$



STRIPS-Beispiel: Blockswelt: Aktionsschemata

⇨ACT: put(x, y)

PRE: ontable(x), clear(x), clear(y)

ADD: on(x, y)

DEL: ontable(x), clear(y)

⇨ACT: puttable(x)

PRE: clear(x), on(x, y)

ADD: ontable(x), clear(y)

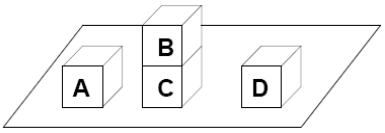
DEL: on(x, y)

⇨ACT: put(x, y)

PRE: on(x, z), clear(x), clear(y)

ADD: on(x, y), clear(z)

DEL: on(x, z), clear(y)



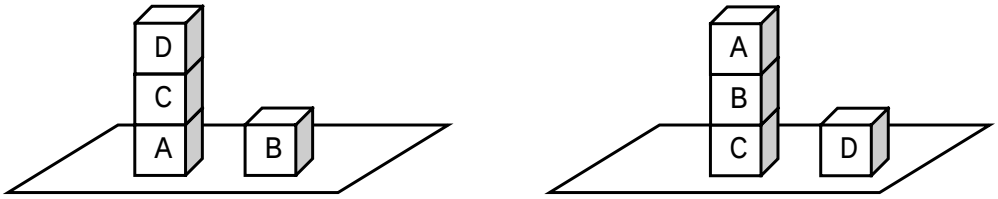
Die beiden put-Aktionen unterscheiden sich in der Position von x – einmal liegt es direkt auf dem Tisch, das andere Mal oben auf einem Stapel. Dem entsprechend sind auch die Effekte unterschiedlich. Die erste put-Aktion kann von A und D (als x) auf D,A oder B (als y) angewendet, die 2. put-Aktion von B auf A oder D angewendet werden.

STRIPS-Beispiel: Blockswelt: Lösung

⇨ A = {on(D,C), on(C,A), clear(D), clear(B),
ontable(A), ontable(B)}

⇨ Z = {on(A,B), on(B,C)}

⇨ Plan = [puttable(D), puttable(C), put(B,C),
put(A,B)]



Plan wird hier synonym als Lösung verwendet. Natürlich existieren auch noch andere (Lösungs-)pläne, die aber aus mehr Aktionen bestehen.

STRIPS-Beispiel: Air Cargo Transport

- Air Cargo Transport beinhaltet
  - Beladen von Flugzeugen
  - Transport zwischen Flughäfen
  - Entladen von Flugzeugen
- Logische Sprache
  - $P = \{\text{Airport}^1, \text{Cargo}^1, \text{Plane}^1, \text{At}^2, \text{In}^2\}$
  - $C = \{C1, C2, P1, P2, JFK, TXL\}$
  - $V = \{a, c, p, from, to\}$

STRIPS-Beispiel: Air Cargo Transport

- Anfangszustand A:
  - $A = \{ At(C1, TXL), At(C2, JFK), At(P1, TXL), At(P2, JFK), Cargo(C1), Cargo(C2), Plane(P1), Plane(P2), Airport(JFK), Airport(TXL) \}$

- Zielzustand Z:
  - $Z = \{ At(C1, JFK), At(C2,TXL) \}$

Notation: C1 = Cargo 1, C2 = Cargo 2, P1 = Plane 1, P2 = Plane 2, JFK = John F Kennedy, Airport, TXL = Tegel

Mittels der Cargo, Plane und Airport-Prädikate wird implizit eine Typisierung eingeführt. STRIPS selbst kennt keine Datentypen.

STRIPS-Beispiel: Air Cargo Transport: Aktionen $\mathcal{A}$

- ACT: load(**c, p, a**)
  - **PRE:** At(c, a), At(p, a), Cargo(c), Plane(p), Airport(a)
  - **ADD:** In(c, p)
  - **DEL:** At(c, a)
- ACT: unload(**c, p, a**)
  - **PRE:** In(c, p), At(p, a), Cargo(c), Plane(p), Airport(a)
  - **ADD:** At(c, a)
  - **DEL:** In(c, p)
- ACT: fly(**p, from, to**)
  - **PRE:** At(p, from), Plane(p), Airport(from), Airport(to)
  - **ADD:** At(p, to)
  - **DEL:** At(p, from)

Notation: C1 = Cargo 1, C2 = Cargo 2, P1 = Plane 1, P2 = Plane 2, JFK = John F Kennedy, Airport, TXL = Tegel

Aktion load umgangssprachlich: c wird auf p in a geladen. PRE: c bezeichnet ein Cargo in a, p ist ein Plane in a, a ist ein Airport. ADD: c ist in p. DEL: c ist in a.

STRIPS-Beispiel: Air Cargo Transport: Lösung

- A = { At(C1, TXL), At(C2, JFK), At(P1, TXL), At(P2, JFK), Cargo(C1), Cargo(C2), Plane(P1), Plane(P2), Airport(JFK), Airport(TXL) }
- Z = { At(C1, JFK), At(C2,TXL) }
- Plan = [
  load(C1, P1, TXL), fly(P1,TXL, JFK), unload(C1, P1, JFK),
  load(C2, P2, JFK), fly(P2, JFK, TXL); unload(C2, P2, TXL)
  ]

Notation: C1 = Cargo 1, C2 = Cargo 2, P1 = Plane 1, P2 = Plane 2, JFK = John F Kennedy, Airport, TXL = Tegel

A und Z wie 2 Folien vorher beschrieben. Plan: Zuerst C1 in P1 in Tegel beladen, dann nach JFK fliegen und wieder ausladen. Danach (2. Zeile) dasselbe umgekehrt mit Plane P2. Alternativ könnte natürlich auch wiederum P1 für den Rückflug gewählt werden.

Gliederung

- ⇨ Einführung
- ⇨ Klassische Planungssprache – STRIPS
- ⇨ Beispiele – STRIPS
- ⇨ Planungsalgorithmen
- ⇨ Menschliches Problemlösen
- ⇨ Zusammenfassung

Planungsalgorithmen: Planen als Suche im Zustandsraum

- Progression planners
  - Vorwärts gerichtete Suche im Zustandsraum
  - Betrachtet Effekt aller möglichen Aktionen angewendet auf gegebenen Zustand
- Regression planners
  - Rückwärts gerichtete Suche im Zustandsraum
  - Rekonstruiert Vorgänger-Zustände des Zielzustands für gegebene Aktionen

Progression Planner wird auch Vorwärtsverkettung; regression planner auch Rückwärtsverkettung genannt.

Problemlösen durch Vorwärtssuche:

- ⇨ Formulierung als Suchproblem
- ⇨ Beginne mit Anfangszustand A
- ⇨ Prüfe die Vorbedingungen aller Aktionsschemata
- ⇨ Berechne Folgezustände durch Ausführen anwendbarer Aktionen

Planungsalgorithmen: Formulierung als Suchproblem

Progression Planner

- Problemraum & Anfangszustand
  - Zustände entsprechen Zuständen des Planungsproblems
  - Fehlende Literale werden als `false` angenommen
- Zielzustand
  - Jeder Zustand des Planungsproblems, der das Ziel erfüllt
- Aktionen
  - Anwendbar sind alle Aktionen, die Vorbedingungen erfüllen.
  - Füge positive Effekte hinzu, lösche negative Effekte.

Planungsalgorithmen — Progression Planner

- Das Planungsproblem kann nun als Suchproblem mit beliebigen Suchalgorithmen gelöst werden.

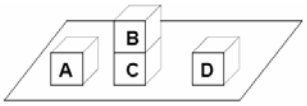
- Problem: Irrelevante Aktionen müssen ausgeführt werden
  - Hoher Verzweigungsgrad
  - Vorwärtssuche ineffizient
  - Verwende Heuristiken

Hoher Verzweigungsgrad entsteht dadurch, dass typischerweise sehr viele Aktionen pro Zustand anwendbar sind (vgl. das motivierende Beispiel am Anfang des Foliensatzes), wobei viele Aktionen nicht zielführend sind. Diese Tatsache ergibt eine sehr komplexe Suche, da der Baum stark aufspannt und bereits in geringer Tiefe eine sehr große Anzahl Blätter enthält.

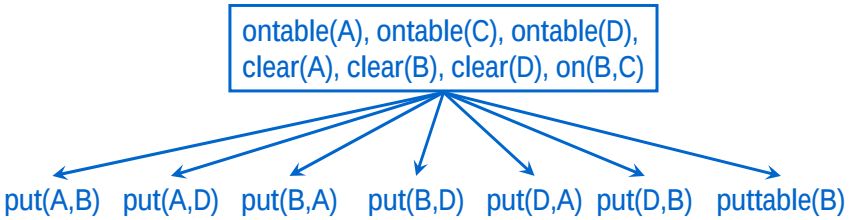
Heuristiken: z.B. bei der Auswahl einer Aktion deren Effekte untersuchen und diejenige Aktion wählen, die dem Zielzustand am nächsten kommt.

Planungsalgorithmen: Beispiel zur Suche — Progression Planner

⇨ A = { ontable(A), ontable(C), ontable(D), clear(A), clear(B), clear(D), on(B,C) }



⇨ Z = { on(A, B), on(B, C) }



⇨ nur die Aktion put(A, B) ist relevant, alle anderen sind irrelevant

Nur die Aktion put(A,B) führt direkt ins Ziel, alle anderen Aktionen tragen nichts zur Lösung bei (z.B. put(A,D)) bzw. entfernen sich vom Ziel (z.B. put(D,A)).

Problemlösen durch Rückwärtssuche:

- ⇨ Beginne mit Ziel Z_0
- ⇨ Bestimme alle Aktionen $a_1, \ldots, a_k$ mit $ADD(a_i) \cap Z_0 \neq \varnothing$
- ⇨ Berechne Vorgängerzustände $Z_{11}, \ldots, Z_{1k}$ durch

$$Z_{1i} = Z_0 - ADD(a_i) \cup PRE(a_i)$$

- ⇨ Terminiere, wenn Zustand $A' \subseteq A$ erreicht.

Bei der Rückwärtssuche wird der Problemraum vom Ziel her bis zum Anfangszustand, also rückwärts, durchsucht. Es wird also der Suchbaum von allen Zielzuständen her durch (logisch korrekte) umgekehrte Anwendung der Regeln expandiert.

Ohne Einschränkung der Vollständigkeit und Korrektheit berechnet sich der Vorgängerzustand aus dem Folgezustand, aus dem die ADD-Liste (d.h. die positiven Effekte) entfernt und dafür die Precondition hinzugefügt wird. Von der Gültigkeit dieses Ansatzes kann man sich überzeugen, indem man, vorwärts suchend, die Aktion vom Vorgängerzustand aus anwendet und sieht, dass tatsächlich der Nachfolgezustand erreicht wird (siehe auch nachfolgendes Beispiel). Die DEL-Liste spielt bei der Rückwärtsverkettung keine Rolle!

Interessant ist die Terminierungsbedingung: Es wird nicht notwendigerweise der vollständige Anfangszustand „generiert“ sondern typischerweise nur eine Teilmenge. Denn diejenigen Zustandspropositionen, die für den Lösungsplan keine Rolle spielen, werden nicht generiert. Dadurch, dass die Rückwärtsplanung derart irrelevante Alternativen nicht berücksichtigt, ist sie effizienter als Vorwärtsplanung. Beispiel: Auf der letzten Folie ist D eine solche irrelevante Alternative, denn D taucht im Lösungsplan nirgends auf.

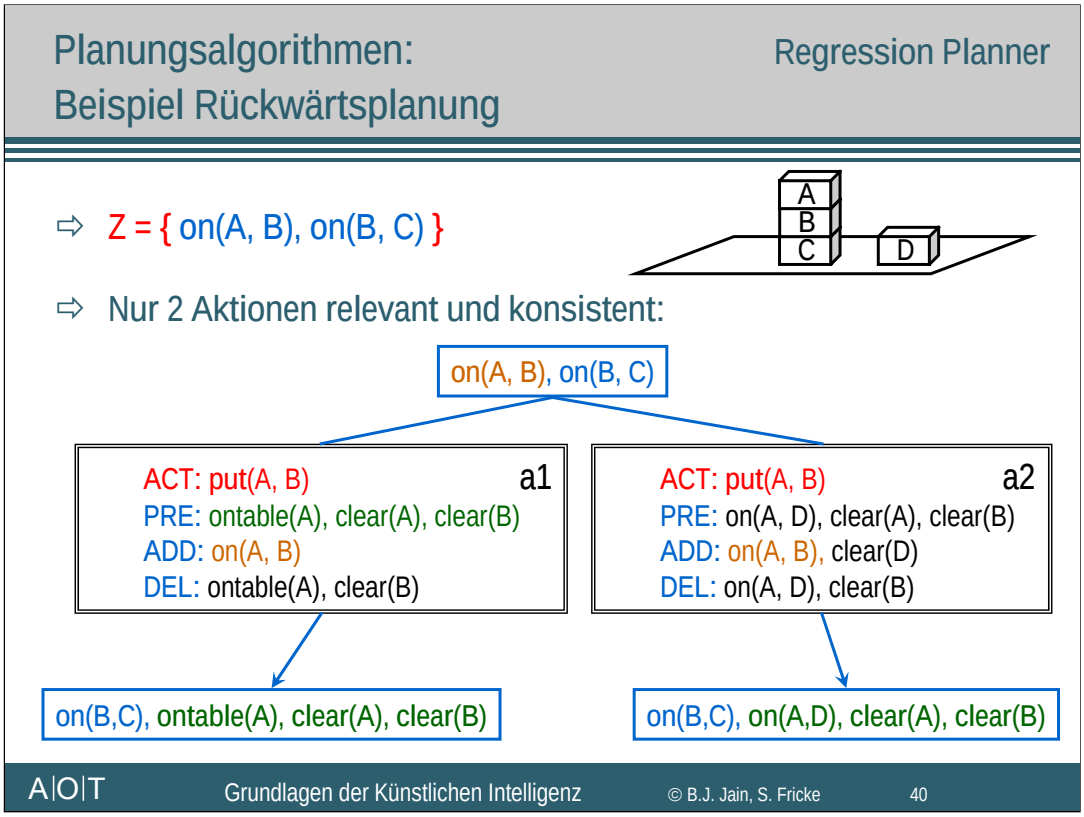
Planungsalgorithmen: Bemerkungen

- ⇨ Aktion **a** ist relevant für **Z**, wenn **ADD(a)** ∩ **Z** ≠ ∅.
- ⇨ Aktion a ist konsistent, wenn es kein Literal in Z löscht.
- ⇨ Nur relevante und konsistente Aktionen werden angewendet.
- ⇨ Dieses Planungsproblem ist ebenfalls ein Suchproblem.
- ⇨ Vorteile:
  - ➙ Nur relevante und konsistente Aktionen werden berücksichtigt
  - ➙ Häufig geringerer Verzweigungsgrad als bei Vorwärtssuche

Relevanz: Die Aktion führt tatsächlich zum Ziel bzw. zur Teilerfüllung des Ziels.

Konsistenz: Die DEL-Liste darf keine Proposition enthalten, die auch im Zielzustand vorkommt. Eine solche Aktion würde das Ziel ja zerstören.

Dieses Planungsproblem kann ebenfalls als Suchproblem mit beliebigen Suchalgorithmen gelöst werden.



Unten die schwarz notierten Literale sind die Vorbedingungen der Aktíonen, die nun nach rückwärtsgewandter Anwendung in den Zustand Z1 mit aufgenommen werden.

Durch Vorwärts-Anwendung der Aktionen (von unten nach oben) kann man sich der Korrektheit überzeugen. Effizient bei diesem Ansatz ist, dass in a2 das Element clear(D) nicht berücksichtigt wird, da es offensichtlich nicht relevant ist.

Der Baum hat an dieser Stelle einen Verzweigungsgrad von 2. Alle anderen Aktionen, die theoretisch auch vom Zustand Z‘ ausgeführt werden könnten, werden nicht untersucht (z.B. puttable(B)). Mittels Vorwärtssuche hätten wir also in diesem Fall einen höheren Verzweigungsgrad und damit mehr Suchaufwand.

Gliederung

- ⇨ Einführung
- ⇨ Klassische Planungssprache – STRIPS
- ⇨ Beispiele – STRIPS
- ⇨ Planungsalgorithmen
- ⇨ Menschliches Problemlösen
- ⇨ Zusammenfassung

Menschliches Problemlösen

- ⇨ Beim menschlichen Problemlösen erfolgt die Wahl der *„passenden“* Repräsentation in Wechselwirkung mit dem Problemlöseprozess.
- ⇨ Fokussierung: Der Mensch konzentriert sich aufgrund der begrenzten Aufmerksamkeitsspanne immer nur auf einen Teil des Problemraums bzw. der Zustandsbeschreibung
- ⇨ Strategien der Problemraumeinengung bzw. –erweiterung werden verwendet.
  - ➙ Dazu müssen oft (perzeptive) Fixierungen auf „*falsche*“ Repräsentationen überwunden werden.

Menschliches Problemlösen

- ⇨ Beim menschlichen Problemlösen sind meist Lösungen von Teilproblemen bekannt, die wieder verwendet und zu neuen Lösungen komponiert werden
  - → Menschen benutzen anstelle einer allgemeinen Suchstrategie häufig Analogien.
  - → Übertragung des Wissens bereits gelöster ähnlicher Probleme.
- ⇨ Menschen verwenden häufig greedy-Strategien

Übertragung des Wissens von schon gelösten ähnlichen Problemen: Fallbasiertes Schließen.

Menschliches Problemlösen

- ⇨ Der Mensch bildet beim wiederholten Lösen ähnlicher Probleme Automatismen aus (Verhaltensprogramme)
  - → Learning by doing (Anderson, 1983)
- ⇨ These: Ein Charakteristikum menschlicher Intelligenz ist die Fähigkeit zur Wahl problemadäquater Repräsentationen und einer optimalen Problemreduktion.

Beim *maschinellen* Problemlösen wird ein Weg vom Start zum Ziel erzeugt und dieser wieder „vergessen“. Tritt dasselbe Problem noch einmal auf, muss die Suche wiederholt werden

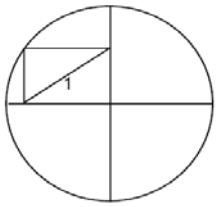
Menschliches Problemlösen

⇨ Die Wahl einer geeigneten Problempräsentation beeinflusst das (einfache) Gelingen der Problemlösung

Aufgabe

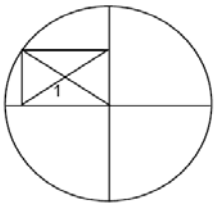
Bestimme die Länge der Linie 1, wenn der Radius des Kreises 15cm beträgt.

Umständliche Zeichnung



Linie 1 bildet eine Seite eines rechtwinkligen Dreiecks

Einfachere Zeichnung

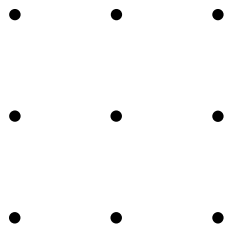


Linie 1 bildet einen Teil der Diagonale eines Rechtecks

Aus Köhler (1969)

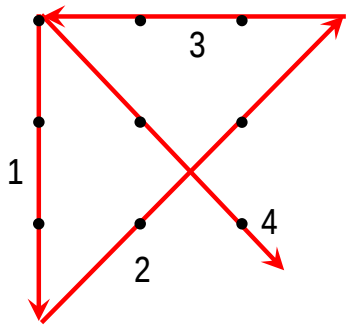
Menschliches Problemlösen: Das 9-Punkte Problem

⇨ Neun in Form eines Quadrates angeordnete Punkte sollen durch vier gerade Linen in einem Zug verbunden werden.



Menschliches Problemlösen: Das 9-Punkte Problem

⇨ Neun in Form eines Quadrates angeordnete Punkte sollen durch vier gerade Linen in einem Zug verbunden werden.



Menschliches Problemlösen: Das 9-Punkte Problem

- ⇨ Die sprachliche Formulierung bzw. perzeptive Fixierung lenkt die Aufmerksamkeit auf eine unvollständige Repräsentation des Problemraumes.

- ⇨ Die notwendige Suchraumerweiterung kann nur über eine abstrakte Ordnungsregel, durch Weiterführung des Aufbaugesetzes der Konfiguration erfolgen.

Gliederung

- ⇨ Einführung
- ⇨ Klassische Planungssprache – STRIPS
- ⇨ Beispiele – STRIPS
- ⇨ Planungsalgorithmen
- ⇨ Menschliches Problemlösen
- ⇨ Zusammenfassung

Zusammenfassung: Planen kombiniert Logik und Suche

- ⇨ Planungssysteme sind Problemlösealgorithmen, die auf einer expliziten propositionalen Repräsentation der Zustände und Aktionen operieren.
- ⇨ Die Planungssprache STRIPS beschreibt
  - → Aktionen durch Vorbedingung und Effekt (ADD/DEL)
  - → Zustände durch Konjunktionen von Literalen
  - → Action Description Language ist eine Erweiterung von STRIPS

Planen ist als Suche oder als Existenzbeweis auffassbar.

STRIPS ist PSPACE-vollständig.

Mehr zu Action Description Language im Anhang.

Zusammenfassung

- ⇨ Suche im Zustandsraum kann vorwärtsgerichtet (progressiv) oder rückwärtsgerichtet (regressiv) verlaufen
  - → Die Wahl des richtigen Verfahrens ist eine Kunst
- ⇨ „Planungsalgorithmus sollte logische Struktur des Problems ausnutzen“
  - → Informationen zu nicht klassischen Planungssystemen im Anhang, in den Referenzen und bei Russel & Norvig, 2003
- ⇨ Exkurs in menschliches Problemlösen als Abschluss des VL-Blocks Problemlösen.

DAI-Labor
TU Berlin

Informierte Suchverfahren

nächster Termin: 17.11.2005

Logik & Wissensrepräsentation 1

Brijnesh J Jain, Stefan Fricke
bjj@dai-labor.de stefan.fricke@dai-labor.de

AIOIT
Agententechnologien in betrieblichen Anwendungen und der Telekommunikation

Action Description Language als Erweiterung von STRIPS

- ⇨ Positive und negative Literale in Zuständen
- ⇨ Open World Assumption: Nicht erwähntes ist unbekannt
- ⇨ Effekt P $\wedge \neg$ **Q ==> add P** und $\neg$ **Q, del** $\neg$P und **Q**
- ⇨ Quantifizierte Variablen in Zielen
- ⇨ Auch Disjunktionen in Zielen erlaubt
- ⇨ Bedingte Effekte
- ⇨ Typisierte Variablen
- ⇨ Gleichheitsprädikat

Gliederung

- ⇨ Einführung
- ⇨ Klassische Planungssprache – STRIPS
- ⇨ Beispiele – STRIPS
- ⇨ Planungsalgorithmen
- ⇨ Planungssysteme
- ⇨ Menschliches Problemlösen
- ⇨ Zusammenfassung

Planungssysteme

- Hierarchisches Planen
  - Erzeugung von Plänen auf verschiedenen Abstraktionsebenen
  - Klassischer Ansatz: NOAH (Sacerdotti, 1974)
- Deduktives Planen
  - Beweis von Planspezifikationen durch Resolution, basierend auf Situationskalkül (Green, 1969)
  - Planen als deduktive Programmsynthese (Manna & Waldinger 1986)
  - Modallogische Verfahren (dynamische Logiken, Temporallogik)

Hierarchisches Planen: hierarchische Strukturierung, immer nur Teile der Hierarchie feinplanen, vorstrukturierte abstrakte Lösungen verwenden, Ressourcen berücksichtigen, Abhängigkeiten berücksichtigen. Z.B. Rundreisen, Umzüge.

Deduktives Planen: Plan entsteht durch konstruktiven Beweis, z.B. durch das Resolutionskalkül (Robinson, 1965).

Planungssysteme

- Nicht-klassisches Planen
  - Behandlung von Unsicherheit
  - Beispiel: Universelles Planen (Schoppers, 1987)
    - Beschreibung des Handlungswissen über gesamten Bereich
    - Robust gegenüber unvorhergesehene Änderungen
    - Effizient
- Weitere wichtige Ansätze (s. Russel & Norvig, 2003)
  - Anytime Planner
  - Planen und Lernen
  - Partial Order Planning
  - Graphplan

Nicht klassisches Planen

•Universelles Planen:

Die Idee des Universellen Planes stammt von Schoppers [Sch87], der damit Planung in nat¨urlicher Umgebung nutzbar machen wollte, insbesondere im Gebiet der Robotik, wo sich Zust¨ande m¨oglicherweise auch ohne Anwendung eines Operators ¨andern k¨onnten. Nehmen wir an, dass ein Roboter einen Turm aus den Bl¨ocken *A*, *B* und *C* bauen soll, d.h. als Ziel ist *fon(A,B), on(B,C)g* vorgegeben und der Startzustand sei *fontable(A), ontable(B), ontable(C)g*. Er plant seine Vorgehensweise und beginnt den Plan auszuf¨uhren. Jetzt hat er schon den Block B auf den Block C gesetzt und pl¨otzlich kommt ein Kind vorbei und dreht den schon gebauten Turm einfach um. Also C steht auf B und B steht auf dem Tisch. Der Roboter h¨atte nun keine Chance mehr, das Ziel zu erf¨ullen, ohne eine komplette Neuplanung zu initiieren. Schoppers Idee war nun, dass der Roboter schon vorher, also beim ersten Planungsschritt, alle m¨oglichen Widrigkeiten mitberechnet und so also nur in seinem bereits fertigen Plan (der jetzt anders definiert werden muss als oben) zu dem aktuell vorliegenden Zustand springen muss und dann gleich ohne Neuplanung weiterarbeiten kann.

•Es wird nicht ein Plan erzeugt, den man zur Lösung eines Problems (Anfangszustand) verwendet, sondern es wird das Handlungswissen über den gesamten Bereich (Problemraum) beschrieben.

•Flexible Reaktion auf unvorhergesehene Änderungen (z.B. ein Block wird umgestoßen), da für jede Situation eine Folge von Aktionsschemata entwickelt wurde

•Sehr effizient

Weitere Ansätze

•Anytime Planer:

Planungsalgorithmen