Grundlagen der Kognitiven Informatik

Problemlösen

Ute Schmid unterstützt von Michael Siebers

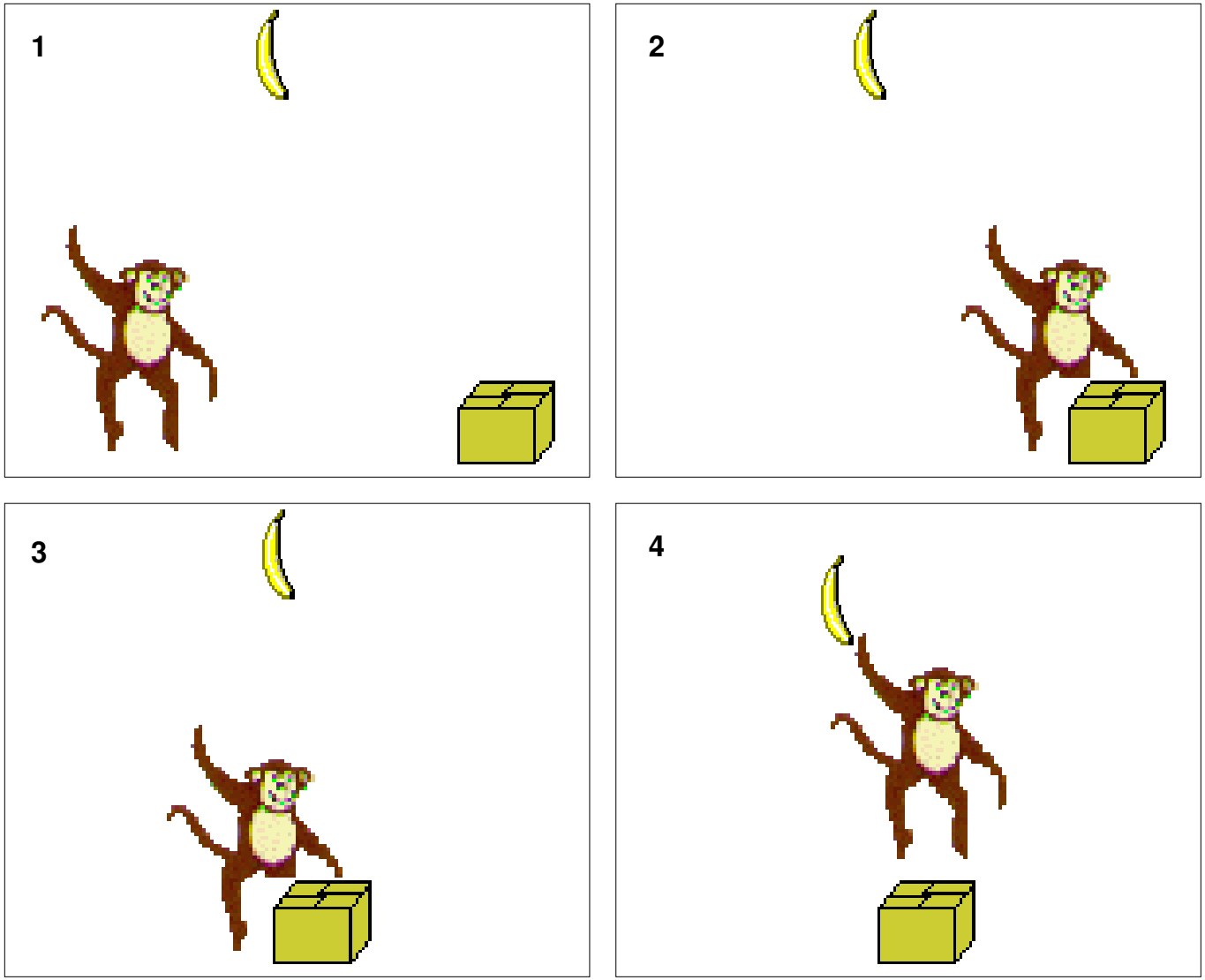
Kognitive Systeme, Angewandte Informatik, Universität Bamberg

letzte Änderung: 27. Oktober 2010

Problemlösen als Suche im Problemraum

- Konzept von Newell & Simon (Human Problem Solving, 1972)
- Geeignete Probleme:
  - klar definierte Problemzustände
  - Existenz einer Menge von Operatoren mit Anwendungsbedingungen zur Überführung eines Zustands in einen anderen
  - Klar definiertes Problemlöseziel (Menge von Zielzuständen)
- $\hookrightarrow$ Transformationsprobleme
- Beispiele: Turm von Hanoi, 8er-Puzzle, Missionare-und-Kannibalen
- $\hookrightarrow$ maschinell lösbar

Illustration: Affe-Banane-Problem



Grundbegriffe

- Ein *Problem* ist gegeben durch einen *Anfangszustand*, ein *Problemlöseziel* und eine Menge von *Aktionen* oder *Problemlöseoperatoren*
- Eine *Problemlösung* besteht dann darin, eine Folge von Aktionen zu finden, um vom Anfangszustand in einen Zustand zu gelangen, in dem das Problemlöseziel erreicht ist.

- Problemzustand: Beschreibung einer aktuellen Situation
  Affe auf Boden links, Kiste rechts, Bananen an Decke
- Problemlöseziel: Affe hat Bananen
  Beispiel für Ziel*zustand*: *Kiste unter Bananen, Affe auf Kiste, Affe hat Bananen*
- Problemlösung als Aktionsfolge
  Affe geht zur Kiste, schiebt Kiste unter Bananen, steigt auf Kiste, greift Bananen

Anmerkungen

- Damit Problem maschinell lösbar: formale Sprache zur Repräsentation von Problemzuständen und Operatoren/Aktionen
- Annahme im Informationsverarbeitungsansatz: Denken als Zustandstransformationen basiert auf mentalen Repräsentationen, die als formale Sprache beschreibbar sind
- Einfache Möglichkeit: Aktionen direkt in Zustandsübergangsfunktion repräsentieren
- Aktion: konkrete Handlung in einer Situation
  Operator: abstrakt formuliertes Schema mit Variablen, Instantiierung für konkreten Zustand
- Problem: keine triviale Erreichung des Zielzustands (sonst Aufgabe)
- Optimale Lösung: kürzeste/günstigste Aktionsfolge zur Erreichung des Ziels
- Länge der Aktionsfolge evtl. durch Auswahl der Aktionen bestimmt (Suchstrategie)

Zustandsübergangsfunktion

Anfangszustand: (links,rechts,boden,nein)

Ziel: (_, _, _, ja)

Bewegung im Raum:
(links,links,boden,nein) ⟶ (mitte,links,boden,nein)
(links,links,boden,nein) ⟶ (rechts,links,boden,nein)
(mitte,links,boden,nein) ⟶ (links,links,boden,nein)
(mitte,links,boden,nein) ⟶ (rechts,links,boden,nein)
(rechts,links,boden,nein) ⟶ (links,links,boden,nein)
(rechts,links,boden,nein) ⟶ (mitte,links,boden,nein)
(links,mitte,boden,nein) ⟶ (mitte,mitte,boden,nein)
(links,mitte,boden,nein) ⟶ (rechts,mitte,boden,nein)
(mitte,mitte,boden,nein) ⟶ (links,mitte,boden,nein)
(mitte,mitte,boden,nein) ⟶ (rechts,mitte,boden,nein)
(rechts,mitte,boden,nein) ⟶ (links,mitte,boden,nein)
(rechts,mitte,boden,nein) ⟶ (mitte,mitte,boden,nein)
(links,rechts,boden,nein) ⟶ (mitte,rechts,boden,nein)
(links,rechts,boden,nein) ⟶ (rechts,rechts,boden,nein)
(mitte,rechts,boden,nein) ⟶ (links,rechts,boden,nein)
(mitte,rechts,boden,nein) ⟶ (rechts,rechts,boden,nein)
(rechts,rechts,boden,nein) ⟶ (links,rechts,boden,nein)
(rechts,rechts,boden,nein) ⟶ (mitte,rechts,boden,nein)

Schieben der Kiste:
(links,links,boden,nein) ⟶ (mitte,mitte,boden,nein)
(links,links,boden,nein) ⟶ (rechts,rechts,boden,nein)
(mitte,mitte,boden,nein) ⟶ (links,links,boden,nein)
(mitte,mitte,boden,nein) ⟶ (rechts,rechts,boden,nein)
(rechts,rechts,boden,nein) ⟶ (links,links,boden,nein)
(rechts,rechts,boden,nein) ⟶ (mitte,mitte,boden,nein)

Steigen auf Kiste:
(links,links,boden,nein) ⟶ (links,links,kiste,nein)
(mitte,mitte,boden,nein) ⟶ (mitte,mitte,kiste,nein)
(rechts,rechts,boden,nein) ⟶ (rechts,rechts,kiste,nein)

Greifen der Banane:
(mitte,mitte,kiste,nein) ⟶ (mitte,mitte,kiste,ja)

Operatoren

Anfangszustand: pos(Affe, Links), auf-boden, pos(Kiste, Rechts), pos(Banane, Mitte)

Ziel: hat-banane

Raumpositionen: ort(Links), ort(Mitte), ort(Rechts)

Operatoren:

GeheVonNach(x,y):
Anwendungsbedingung: ort(x), ort(y), pos(Affe, x), auf-boden
Auswirkung: ADD pos(Affe,y); DEL pos(Affe, x)

SchiebeKiste(x,y):
Anwendungsbedingung: ort(x), ort(y), pos(Affe, x), pos(Kiste, x), auf-boden
Auswirkung: ADD pos(Affe, y), pos(Kiste, y); DEL pos(Affe, x), pos(Kiste, x)

SteigeAufKiste(x):
Anwendungsbedingung: ort(x), pos(Affe, x), pos(Kiste, x), auf-boden
Auswirkung: ADD auf-kiste; DEL auf-boden

GreifeBanane(x):
Anwendungsbedingung: ort(x), pos(Affe, x), pos(Banane, x), auf-kiste
Auswirkung: ADD hat-banane

Anmerkungen

- Repräsentiert werden nur „relevante“Aspekte der Situationen (z.B. nicht Farbe der Kiste)
- Raumpositionen sind hier nur grob festgelegt. Die drei Raumpositionen sind *statics*, d.h. Teile der Problembeschreibung, die nie durch Aktionen verändert werden
- Repräsentation mit Zustandsübergangsfunktion: aufwendig und unübersichtlich
- Repräsentation durch Operatoren eleganter (vgl. KI-Planungssprache PDDL), aber: Algorithmen werden aufwendiger wg. Variableninstantiierung
- **Instantiierung**: Ersetzen von Variablen durch konstante Ausdrücke gleiche Variablennamen über den gesamten Operator durch gleiche Ausdrücke; verschiedene Variablennamen können durch gleiche Ausdrücke ersetzt werden, wenn nicht explizit Ungleichheit gefordert wird!
- Abgeschlossenheitsannahme (**closed world assumption**): Was nicht explizit genannt ist, wird als falsch angenommen
  $\hookrightarrow$ dadurch muss nicht repräsentiert werden, dass der Affe die Banane **nicht** hat

Produktionsregeln

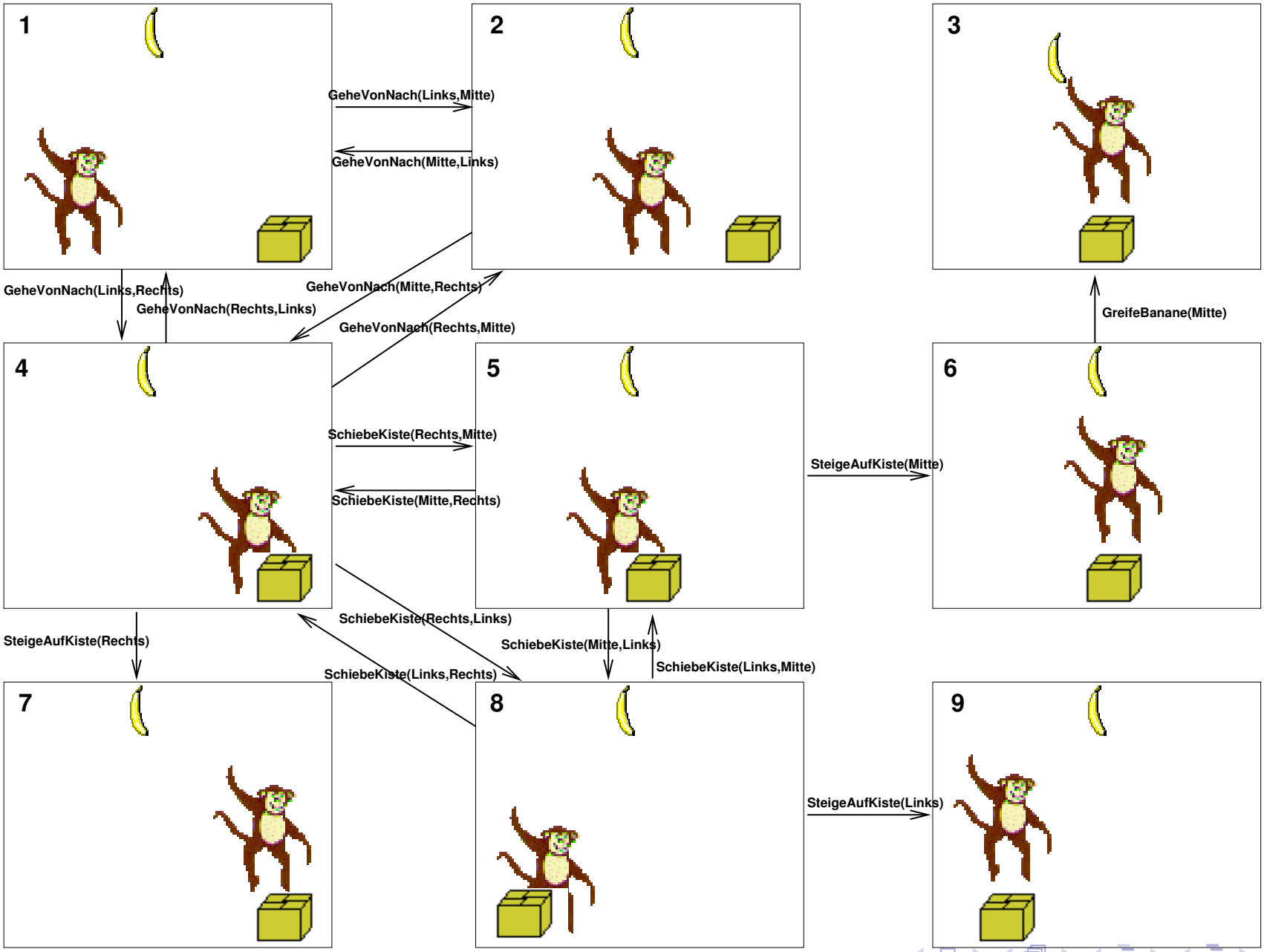
- Operatoren können als Produktionsregeln verstanden werden
- Produktionsregel: WENN ⟨Bedingung⟩ DANN ⟨Aktion⟩
- Anwendbar auf einen Zustand nur, wenn Bedingung erfüllt ist (Instantiierung der Variablen möglich so, dass Bedingung in Zustandsbeschreibung enthalten)
- Ausführung der Aktion: Zustand wird durch syntaktisches Hinzufügen der (instantiierten) Ausdrücke in ADD und Löschen der (instantiierten) Ausdrücke in DEL in einen neuen Zustand überführt

aktueller Zustand	**Operatoranwendung**	**Folgezustand**
pos(Affe, Links) auf–boden pos(Kiste, Rechts) pos(Banane, Mitte)	geheVonNach(Links, Rechts) →	~~pos(Affe, Links)~~ auf–boden pos(Kiste, Rechts) pos(Banane, Mitte) *pos(Affe, Rechts)*
ort(Links) ort(Mitte) ort(Rechts)		ort(Links) ort(Mitte) ort(Rechts)

Problemraum

- Anfangszustand und Menge von Operatoren/Aktionen definieren einen Problemraum
- Ausgehend vom Anfangszustand können *alle möglichen* Zustände erzeugt werden
- Graph mit Zuständen als Knoten und Aktionen als gerichtete Kanten
- Problemraum ist üblicherweise *nicht* explizit gegeben
- Durch Problemlösen als Suche im Problemraum wird ein Teil des Problemraums konstruiert
- Im Allgemeinen kann ein Problemraum sehr groß sein

Problemraum für das Affe-Banane Problem



Problemlösungen

- Startzustand 1
- Zielzustand 3
- Optimaler Pfad: $1 - 4 - 5 - 6 - 3$

GeheVonNach(Links, Rechts),
SchiebeKiste(Rechts, Mitte),
SteigeAufKiste(Mitte),
GreifeBanane(Mitte).

Weitere (nicht optimale) Lösung:

GeheVonNach(Links, Mitte),
GeheVonNach(Mitte, Rechts),
SchiebeKiste(Rechts, Links),
SchiebeKiste(Links, Mitte),
SteigeAufKiste(Mitte),
GreifeBanane(Mitte).

Suchstrategien

- Problemlösen als Suche im Problemraum
- Systematische Bewegung im Problemraum
- Strategie bedingt Lösungsweg, Lösungserfolg
- Uninformierte/blinde Strategien:
  - Versuch und Irrtum: bei Menschen häufig, wenn Problem zu komplex; kann durch Zufall zu Lösung führen
  - Systematisch, aber kognitiv (und auch für KI-Planung) zu aufwendig: **Breitensuche**
  - Kann schnell zum Erfolg führen (bei Problemen, mit vielen Lösungsmöglichkeiten), liefert aber nicht garantiert die kürzeste Lösung: **Tiefensuche**
  - Plausibler für menschliches Problemlösen: **heuristische Suche**

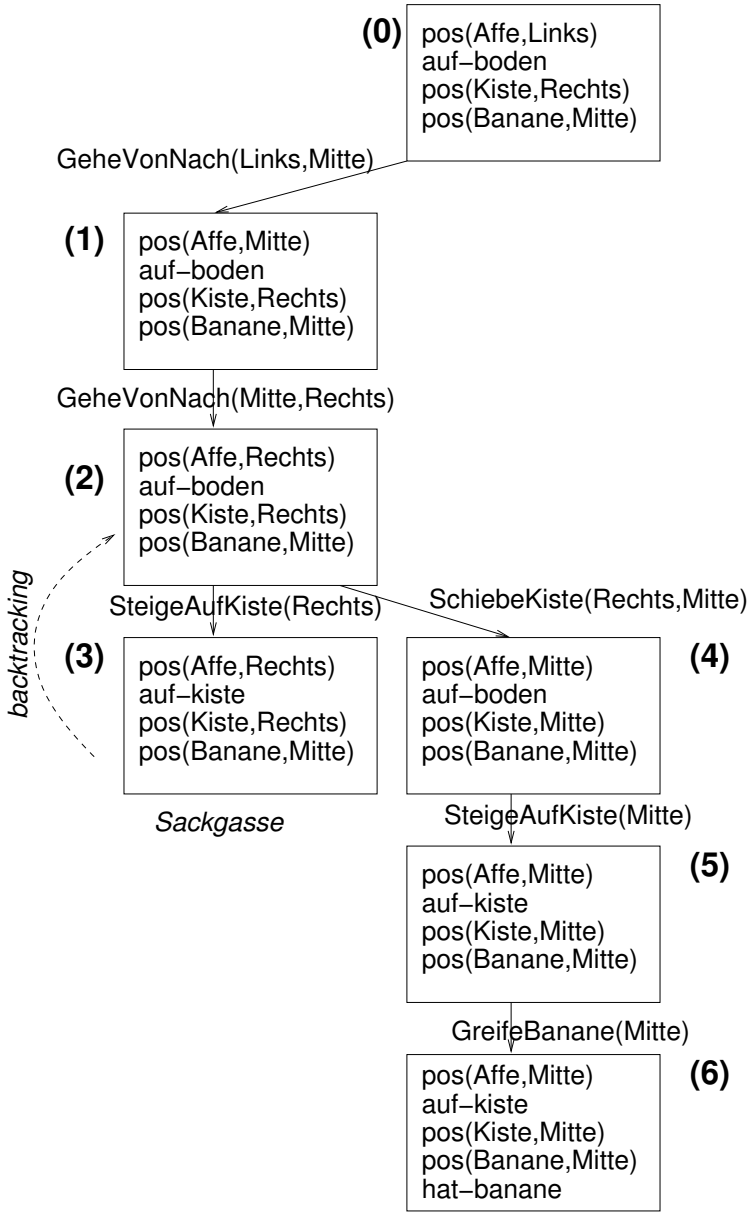
Tiefensuche

Ordnung auf den Operatoren:

(1) GreifeBanane,
(2) SteigeAufKiste,
(3) SchiebeKiste,
(4) GeheVonNach

Bevorzugung von Orten:
(1) Mitte,
(2) Rechts,
(3) Links

Die statischen Prädikate *ort(Links)*, *ort(Mitte)* und *ort(Rechts)* werden bei den Zustandsbeschreibungen der Übersichtlichkeit halber weggelassen.



Anmerkungen zur Tiefensuche

- Ordnung der Operatoren soll die Präferenzen des Affen widerspiegeln
- Tiefensuche kann in **Sackgassen** führen
- **Backtracking** zur Rücknahme von Aktionen
- Schlimmster Fall: kompletter Problemraum muss durchsucht werden
- Problem: **Zyklen**, Bewegung innerhalb des Problemraums immer wieder im Kreis zum selben Zustand zurück (Zyklenprüfung in jedem Pfad des Suchbaums!)
- Suchstrategie erzeugt **Suchbaum** (satt Graph gerichtete Kanten)

Breitensuche

- Pfad wird nicht zuerst in die Tiefe verfolgt, sondern der Baum wird ebenenweise aufgebaut
- Auf jeden Zustand werden alle möglichen Aktionen angewendet, es wird also nicht nur ein sondern alle zulässigen Folgezustände erzeugt
- Durch den ebenenweisen Aufbau des Suchbaums wird diejenige Lösung gefunden, die mit der geringsten Anzahl von Operator-Anwendungen zum Ziel führt.
- Im Allgemeinen wird bei der Breitensuche jedoch ein größerer Teil des Problemraums generiert als bei der Tiefensuche, die Suche ist also mit mehr Aufwand verbunden.

Heuristische Suche

- Bei den blinden Suchstrategien werden ausgehend vom Anfangszustand Folgezustände geniert, solange bis ein erreichter Zustand das Problemlöseziel erfüllt.
- Die Suche ist also nicht auf das Problemlöseziel hin ausgerichtet.
- Alternativ zur Operatorauswahl nach einer beliebigen Ordnung kann in jedem Zustand derjenige Operator angewendet werden, der den Unterschied des aktuellen Zustands zum Zielzustand am meisten verringert (Unterschiedsreduktion).
- Die Suche wird damit durch eine *heuristische Bewertung* gesteuert.
- Suchverfahren, bei denen die Operatorauswahl aufgrund heuristischer Bewertungen erfolgt, werden als heuristische Suchstrategien bezeichnet.

Hill Climbing und Bewertungsfunktionen

- Eine auf Unterschiedsreduktion basierende Strategie ist das **Hill Climbing** ("Bergsteigen").
- Wenn man sich das Ziel als einen Berggipfel vorstellt, wählt man den nächsten Schritt immer so, dass man möglichst viel an Höhe gewinnt, da dadurch die Distanz zum Ziel am meisten verringert wird.
- Affe-Banane-Problem: Hin-und-Her-Laufen im Raum verringert Unterschied zum Zielzustand nicht, Schieben der Kiste in die Mitte unter die Bananen dagegen schon.
- Allerdings basiert die Entscheidung beim *Hill Climbing* immer nur auf dem aktuellen Zustand, ist also lokal.
- Durch die Vernachlässigung der globalen Struktur des Problems kann es passieren, dass man sich "verrennt"
  - Selbst wenn jede Operatoranwendung den Problemlöser näher ans Ziel bringt, kann es sein, dass er in einem Zustand landet, von dem aus er sich erst wieder vom Ziel entfernen (also "absteigen") müsste, um tatsächlich ans Ziel zu gelangen.
  - *Hill Climbing* kann also zu lokalen Maxima führen.

Lokale Maxima

Psychologische Befunde zum Hill Climbing

- Empirische Hinweise, dass menschliche Problemlöser eine Strategie der Unterschiedsreduktion anwenden
- Bei der Lösung von „Fluss-Überquerungs“Problemen, z.B. *Hobbits and Orcs* (Missionare und Kannibalen) (Greeno, 1974)
- eine Gruppe von *Hobbits* und *Orcs*will mit einem Boot einen Fluss überqueren, wobei die *Orcs* nie in Überzahl sein dürfen
- Probanden haben vor allem Schwierigkeiten mit dem Übergang von Zustand (6) zu Zustand (7)
- Es müssen zwei Passagiere zurück zum Ausgangsufer gebracht werden: es muss also ein Zustand hergestellt werden, der sich stärker vom Ziel unterscheidet, als ein bereits vorliegender Zustand.
- Modellierung mit einem Produktionsystem (Schmalhofer & Polson, 1986)

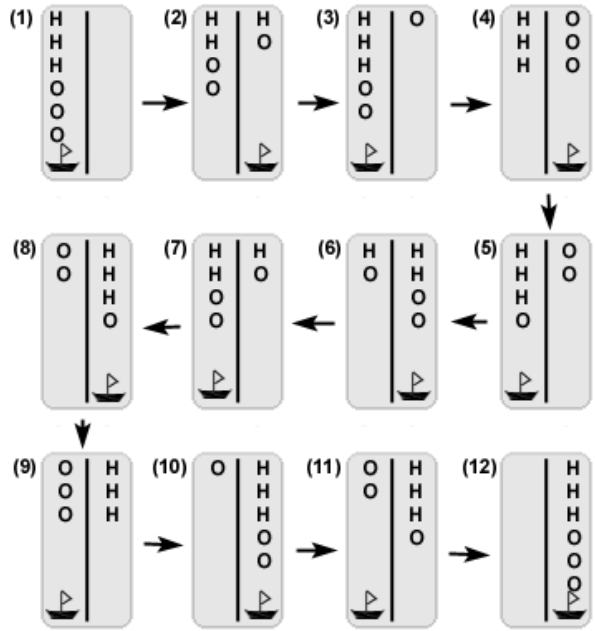
Hobbits and Orcs

Anfangszustand: Drei Hobbits (H) und drei Orcs (O) befinden sich am linken Ufer eines Flusses.

Zielzustand: Alle sechs befinden sich am rechten Ufer des Flusses.

Operator: Mit einem Boot (b) können mindestens ein und maximal zwei Passagiere gleichzeitig von einem Ufer zum anderen transportiert werden.
Anwendungsbedingung: An keinem Ufer dürfen mehr Orcs als Hobbits sein (da die Orcs sonst die Hobbits auffressen).

Lösungsweg:

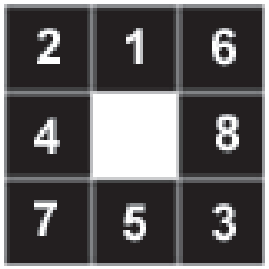


Definition von Heuristiken

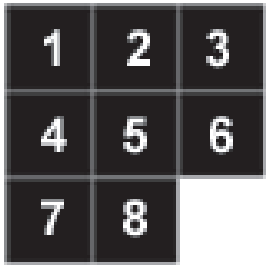
- Damit eine Heuristik von einem computer-implementierten Suchprogramm zur Steuerung der Suche verwendet werden kann, muss die Heuristik formal definiert werden.
- Sehr einfache Bewertungsfunktion: jeweils die Anzahl der im aktuellen Zustand schon erfüllten Komponenten des Problemlöseziels ermitteln
- *Hobbits-and-Orcs*: Anzahl der Passagiere am rechten Flussufer
- Achterpuzzle: Anzahl von Plättchen, die bereits korrekt plaziert sind.
- Je mehr Wissen über die Struktur eines Problems verfügbar ist, um so differenzierter kann die Bewertung von Zuständen erfolgen und um so wahrscheinlicher wird es, dass der bei der Suche eingeschlagene Pfad zum Erfolg führt.
- Achter-Puzzle: wichtiger, dass die Plättchen in die korrekte Reihenfolge gebracht werden als dass einzelne Plättchen schnellstmöglich an ihre Zielposition befördert werden.
- Schachcomputer *Deep Blue* arbeitet mit einer von Schachexperten entwickelten sehr ausgefeilten Bewertungsfunktion für Spielpositionen

Achterpuzzle

möglicher Anfangszustand:



Zielzustand:



Operator: Verschiebe das leere (weiße) Feld um eine Position nach links, rechts, oben oder unten