VL 4: Produktionssysteme und KI Planung

Grundlagen der Kognitiven Informatik

Ute Schmid

Cognitive Systems, Applied Computer Science, University of Bamberg
www.uni-bamberg.de/cogsys

Last change: 18. November 2019

Problemlösen und Planen

KI: Problemlösen/Planen

- Problemlösen: spezielle, eingeschränkte Repräsentationen für Zustände und Operatoren, Einbeziehung von Wissen in Form von Heuristiken
  - Suchverfahren (am bekanntesten $A*$, Nilsson, 1971)
- Planung: Effiziente Algorithmen zur Erzeugung von Aktionssequenzen
  - Suche als *ein* möglicher Zugang
  - Suchbasierte Planungsalgorithmen arbeiten (wie Produktionssysteme) mit Match-Select-Apply Zyklen (Erläuterung später)
  - alternative Ansätze: Deduktion, Case-based reasoning
  - domain-unabhänge und domain-spezifische Ansätze (hierarchisches Planen vgl. zielgesteuerte Produktionssysteme)
  - domain-unabhängige Ansätze: *keine* Vorgabe von Wissen (z.B. HSP, Bonet und Geffner, 1999: Verfahren zur automatischen Schätzung von Restwegkosten)
  - Standard-Sprache: PDDL (*planning domain definition language*)

PDDL-artige Repräsentation

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

Blocksworld in PDDL

```
(define (domain blocksworld-adl)
(:requirements :strips :equality :conditional-effects)
(:predicates (on ?x ?y)
             (clear ?x)) ; clear(Table) is static
(:action puton
:parameters (?x ?y ?z)
:precondition (and (on ?x ?z) (clear ?x) (clear ?y)
                   (not  (= ?y ?z)) (not (= ?x ?z))
                   (not (= ?x ?y)) (not (= ?x Table)))
:effect (and (on ?x ?y) (not (on ?x ?z))
             (when (not (eq ?z Table)) (clear ?z))
             (when (not (eq ?y Table)) (not (clear ?y)))))
)
```

KI Planung

- PDDL als uniforme Repräsentationssprache für beliebige Domains (Operatoren) und Probleme (Anfangszustand und Ziele)
- Operatoren mit Variablen
- Match: Existiert eine Variablenbelegung so, dass die Anwendungsbedingungen im aktuellen Zustand enthalten sind?
- Select: Wenn mehr als ein instantiierter Operator (auch der selbe mit verschiedenen Instantiierungen) anwendbar ist: Auswahl (im einfachsten Fall *top-down*)
- Apply: Lösche die DEL Ausdrücke im aktuellen Zustand und füge die ADD Ausdrücke hinzu

Reinforcement Learning

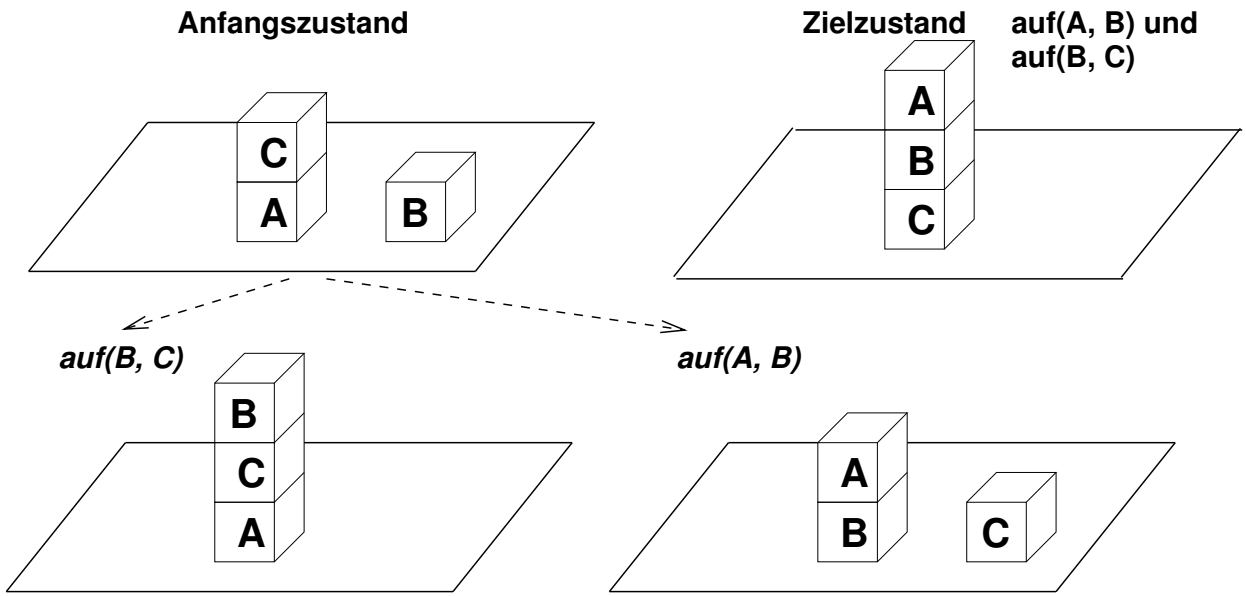
Lernen von Aktionsfolgen

- Speziell in nicht-deterministischen Umgebungen wird häufig **reinforcement learning** eingesetzt, um die beste *policy* für eine Problem-Domäne festzulegen.
- Für jeden Zustand wird diejenige Aktion ausgewählt, die am erfolgreichsten scheint.
- Statt zunächst Ermittlung der kompletten Aktionsfolge: Verzahung von Aktionsauswahl und Ausführung.

STRIPS

- Eines der ersten Planungssysteme: STRIPS (*Stanford Research Institute Problem Solver*), Fikes und Nilsson, 1971
- Idee der ADD-DEL Listen (*closed world assumption*)
- Repräsentationssprache ist Kern von PDDL
- Planungsalgorithmus wird dagegen nicht mehr verwendet
- Problem (wie bei MEA): lineares Planen, kann nicht mit abhängigen Teilzielen umgehen = **Sussman Anomalie**

Die Sussman Anomalie



Erläuterung der Sussman-Anomalie

- Um die Ziele *auf(A, B)* **UND** *auf(B, C)* von dem gegebenen Zustand aus zu erreichen, könnte B unmittelbar auf C gestellt werden.
- Damit wäre eines der Teilziele erreicht, aber um *auf(A,B)* zu erreichen, müsste der Turm komplett wieder abgebaut werden.
- Andererseits könnte man zunächst C auf den Tisch legen und dann A auf B setzen, aber wieder ist damit nur ein Teilziel erfüllt.

- Das Problem bei STRIPS und der Mittel-Ziel-Analyse ist, dass jeweils *ein* aktuelles Ziel fokussiert wird und zunächst alle Aktionen ausgeführt werden, die dieses Ziel herstellen.
- Durch diese **lineare Strategie** sind manche Probleme nicht lösbar.
- Die Sussman-Anomalie ist leicht auflösbar, wenn, nachdem Block C auf den Tisch gestellt wurde, zunächst B auf C und dann A auf B gesetzt wird.

Nicht-Lineare Strategie

- Stelle *C* auf den Tisch, stelle dann *B* auf *C* und dann *A* auf *B*.
- Hierbei erfolgt ein Fokuswechsel von Ziel *auf(A, B)* auf Ziel *auf(B, C)*
- Um das Ziel *auf(A, B)* zu erfüllen, muß zunächst *C* auf den Tisch gelegt werden. Dadurch wird *A* frei und damit Operator *Stelle A auf B* anwendbar.
- Nachdem *A* und *B* frei sind, kann aber auch der Operator *Stelle B auf C* angewendet werden.
- Algorithmisch kann dies leicht erreicht werden, indem die Teilziele nicht in einem *Stack*, sondern als Menge verwaltet werden.
- Das erste Planungssystem, das mit solchen Abhängkeiten umgehen konnte, war NOAH (Sacerdoti, 1977).
- Alle modernen Planer arbeiten mit einer solchen nicht-linearen Strategie, bei der Teilziele "verschränkt" abgearbeitet werden können.

Closed-world Assumption

CWA

- Alles, was nicht explizit als wahr bewiesen werden kann wird als falsch angenommen
- Alles, was nicht modelliert ist, existiert im Modell auch nicht und ist nicht beweisbar, und damit falsch
- In der Prädikatenlogik gilt die CWA nicht (aber die logische Programmiersprache Prolog basiert darauf)
- Beispiel Bahnfahrplan: Falls ein Zug planmäßig ausschließlich jede volle Stunde abfährt, ist der Umkehrschluss, dass er zu anderen Zeiten nicht abfährt, durchaus legitim.

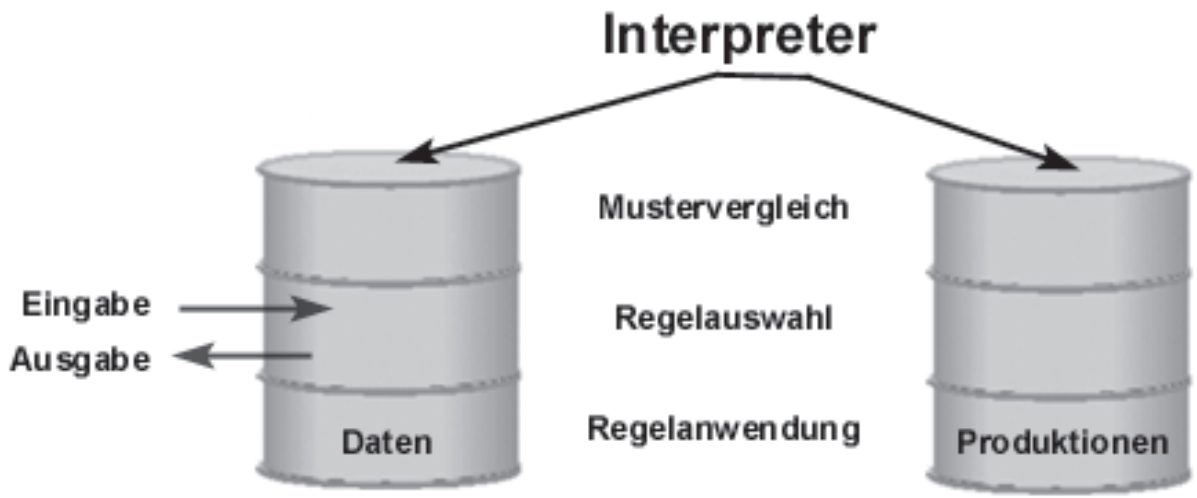
Kognitive Architekturen und Produktionssysteme

- Kognitive Modellierung: Computersimulation der kognitiven Strukturen und Prozesse, von denen auf Grundlage einer psychologischen Theorie angenommen wird, dass Menschen ein Problem auf diese Weise bearbeiten (*white box*)
- Häufiger Zugang: Kognitive Architektur
  - Festlegung allgemeiner Prinzipien der Informationsverarbeitung
  - Modelle, die im Rahmen einer kognitiven Architektur realisiert werden, basieren alle auf den gleichen Grundprinzipien
- Kognitive Architekturen sind üblicherweise als Produktionssystem realisiert

Produktionssystem

- Ein System, das die Anwendung von (bedingten) Regeln, *Produktionsregeln* oder Produktionen genannt, auf Daten steuert, heißt Produktionssystem.
- Die aktuell im Arbeitsspeicher befindlichen Daten können beispielsweise die Beschreibung eines Problemzustands sein und die gespeicherten Regeln Problemlöseoperatoren der Form "Wenn ⟨Bedingung⟩ Dann ⟨Aktion⟩".
- Die Steuerung der Regelanwendung erfolgt durch einen *Interpreter*.
- Zunächst werden Ausgangsdaten über eine Eingabe-Schnittstelle in den Arbeitsspeicher gebracht.
- Danach werden solange Regeln auf die Daten im Arbeitsspeicher angewendet, bis entweder keine Regel mehr anwendbar ist oder eine Regel zur Anwendung kommt, deren Aktion ein Kommando zum Stoppen der Abarbeitung ist.
- Danach werden die aktuellen Daten als Ergebnis ausgegeben.

Aufbau eines Produktionssystems



Match-Select-Apply Zyklen

Die Transformation der Daten durch Anwendung von Regeln erfolgt in sogenannten *Match-Select-Apply*-Zyklen:

Mustervergleich (*match*): Suche alle Produktionsregeln, deren Bedingungsteil mit den Daten verträglich ist.

Auswahl (*select*): Wähle – nach einer vorgegebenen *Konfliktauflösungs-Strategie* – eine dieser Regeln aus.

Anwendung (*apply*): Wende die Regel auf die Daten im Arbeitsspeicher an.

- In jedem Zyklus kommt also eine Regel zur Anwendung, die die Daten im Arbeitsspeicher verändert.
- Eine einfache Anleitung zur Implementation eines Produktionssystems in der Programmiersprache Lisp gibt Winston (1989).

Beispiel: Kaffee-Bohnen-Problem

Gegeben ist eine Kaffeedose, in der schwarze (S) und weiße (W) Bohnen in einer festen Reihenfolge angeordnet sind,
beispielsweise: *W W S S W W S S*.
Gegeben sind folgende Regeln:

$$\begin{aligned} \text{S W} &\rightarrow \text{S} \\ \text{W S} &\rightarrow \text{S} \\ \text{S S} &\rightarrow \text{W} \end{aligned}$$

Das Ziel ist, am Ende möglichst wenige Bohnen zu haben. Die Konfliktlösungs-Strategie sei, immer die oberste anwendbare Regel auszuwählen.

W W S S W W S S
W W S S W S S
W W S S S S
W S S S S
S S S S
W S S
S S
W

Mustervergleich

- Im einfachsten Fall (wie beim Kaffee-Bohnen Problem) meint Mustervergleich zu prüfen, ob der Bedingungsteil einer Regel mit einem Ausschnitt der Daten übereinstimmt.
- Im Allgemeinen kann der Bedingungsteil einer Regel Variablen enthalten.
- Mustervergleich (*pattern matching*) meint dann zu prüfen, ob ein vorgegebenes Muster mit den Daten verträglich ist.
- Verträglichkeit heißt, dass die Variablen im Muster so belegt werden können, dass der entstandene Ausdruck mit einem Ausschnitt der Daten übereinstimmt.

Mustervergleich beim Affe-Banane-Problem

Der Bedingungsteil der Regel *SteigeAufKiste* ist beispielsweise mit Daten-1 verträglich, aber nicht mit Daten-2:

Muster: *ort(x), pos(Affe, x), pos(Kiste, x), auf-boden*

Daten-1: *pos(Affe, Mitte), auf-boden, pos(Kiste, Mitte), pos(Banane, Mitte)*

Daten-2: *pos(Affe, Links), auf-boden, pos(Kiste, Rechts), pos(Banane, Mitte).*

- Wenn Variable x im Muster mit *Mitte* belegt ist, dann kommen alle Ausdrücke des Musters in Daten-1 vor.
- Für Daten-2 gibt es keine Belegung für x, die diese Bedingung erfüllt.
- Würde x mit *Links* belegt, so ist der Fakt *pos(Kiste, Links)* nicht mit den Daten verträglich; für $x = Rechts$ ist der Fakt *pos(Affe, Rechts)* unverträglich; für $x = Mitte$ sind die Fakten *pos(Affe, Mitte)* und *pos(Kiste, Mitte)* unverträglich.

Konfliktlösung

- Das Ergebnis des Mustervergleichs liefert die Menge aller Regeln, die auf die aktuellen Daten anwendbar sind.
- Ist diese Menge leer, so hält das System an.
- Enthält die Menge genau eine Regel, so wird diese auf die Daten angewendet.
- Enthält die Menge mehr als eine Regel, so muss eine Entscheidung getroffen werden, welche dieser Regeln zur Anwendung kommen soll.
- Dies geschieht im Allgemeinen dadurch, dass man eine Präferenz-Ordnung auf den Regeln definiert.
- Diese Präferenz-Ordnung kann auf verschiedene Weise, also durch verschiedene Strategien zur Konfliktlösung, definiert werden.

Strategien zur Konfliktauflösung

Erste Übereinstimmung: Nimm die erste Regel, die mit den Daten verträglich ist. Dabei ist "erste" bezüglich der Anordnung der Regeln im Produktionsspeicher festgelegt.

Höchste Priorität: Nimm die Regel mit dem höchsten Prioritätswert. Der Prioritätswert kann dabei beispielsweise über einen "Stärkewert" der Regel ermittelt werden, der dynamisch verändert werden kann, oder er kann problemspezifisch definiert werden. (in ACT, Anderson, 1983)

Spezifischste Bedingung: Nimm die Regel, die die spezifischsten Anwendungsbedingungen hat.

Aktualität: Nimm die Regel, die sich auf ein Datenelement bezieht, das erst kürzlich (*most recently*) erzeugt worden ist.

Neuigkeit: Nimm eine Regel, die noch nicht angewendet wurde (mit einer noch nicht betrachteten Variablenbelegung).

Zufall: Wähle zufällig eine Regel.

Keine Wahl: Exploriere alle anwendbaren Regeln.

Letztgenannte Strategie: Breitensuche, die anderen: Tiefensuche

Regelanwendung

- Ist eine Regel ausgewählt, so wird sie auf die Daten im Arbeitsspeicher angewendet. Man sagt auch, die Regel „feuert“.
- Dadurch wird der Inhalt des Arbeitsspeichers verändert.
- Beispielsweise transformiert die Anwendung eines Problemlöse-Operators den aktuellen Zustand in einen Folgezustand.
- Es gibt zwei grundlegende Verarbeitungsstrategien für Regeln: **Vorwärtsverkettung** und **Rückwärtsverkettung**.

Datengetriebene Systeme

- Vorwärtsverkettung meint, dass ausgehend von den aktuellen Daten im Arbeitsspeicher jeweils der Aktionsteil einer Regel – also die rechte Regelseite – ausgeführt wird.
- Dadurch werden die Daten schrittweise verändert. Man spricht hier auch von *datengesteuerter* (*data-driven*, *bottom-up*) Regelanwendung.
- Ist neben den aktuellen Daten ein Ziel vorgegeben, so existiert üblicherweise eine Regel, der Form WENN ⟨Ziel erreicht⟩ DANN halt.
- Vorwärtsverkettung bewirkt also, dass die Daten schrittweise in Richtung Zielzustand transformiert werden.

Vergleiche: Problemlösen durch Tiefensuche

Zielgesteuerte Systeme

- Rückwärtsverkettung meint dagegen, dass ausgehend von einem gegebenen Ziel immer die Regel ausgeführt wird, deren rechte Seite das Ziel unmittelbar herstellen kann.
- Ist die linke Seite der Regel – die Anwendungsbedingungen oder Teilziele – im aktuellen Zustand nicht erfüllt, so werden die dort genannten Bedingungen als Teilziele eingeführt und eine Regel gesucht, die diese Teilziele unmittelbar herstellt.
- Rückwärtsverkettung bewirkt also, dass ausgehend vom Problemlöseziel das Problem so lange in Teilziele zerlegt wird, bis Teilziele gefunden sind, die gelten oder im aktuellen Zustand unmittelbar erfüllt werden können.
- Man spricht hier auch von *zielgesteuerter* (*goal-driven*, *top-down*) Regelanwendung.

Vergleiche: Mittel-Ziel-Analyse

Beispiele

- Die Mittel-Ziel-Analyse kombiniert beide Strategien:
  - Ein Zustand wird vorwärts in einen Folgezustand transformiert, wenn ein Operator, der die Distanz zum Ziel minimiert, angewendet werden kann.
  - Ansonsten wird als neues Ziel gesetzt, die Anwendungsbedingung des Operators zu erfüllen, also rückwärts das Ziel in Teilziele zerlegt.
- Die Programmiersprache Prolog basiert auf Rückwärtsverkettung.

Beispiel: Umformung von mathematischen Ausdrücken

Produktionen:

$$
\begin{array}{lll}
R_1 & 1 \times x & \Longrightarrow \quad x \\
R_2 & x \times (y + z) & \Longrightarrow \quad x \times y + x \times z \\
R_3 & 2 \times x & \Longrightarrow \quad x + x \\
R_4 & x \times y & \Longrightarrow \quad y \times x
\end{array}
$$

Arbeitsweise

(0) Working Memory: $2 \times (1 + 1 \times 2)$

Match: Applicable Rules

R_1 for 1×2
R_2 for complete expression
R_3 for complete expression
R_4 for 1×2
R_4 for complete expression

Select: R_1

Apply: results in $2 \times (1 + 2)$

Arbeitsweise

(1) Working Memory: $2 \times (1 + 2)$

Match: Applicable Rules

R_2 for complete expression
R_3 for complete expression
R_4 for complete expression

Select: R_2

Apply: results in $2 \times 1 + 2 \times 2$

Arbeitsweise

(2) Working Memory: $2 \times 1 + 2 \times 2$

Match: Applicable Rules

R_3 for left part
R_3 for right part
R_4 for left part
R_4 for right part

Select: R_3

Apply: results in $1 + 1 + 2 \times 2$

Arbeitsweise

(3) Working Memory: $1 + 1 + 2 \times 2$

Match: Applicable Rules

R_3 for right part

R_4 for right part

Select: R_3

Apply: results in

(4) Working Memory: $1 + 1 + 2 + 2$

Match: no rule

Halt

Anmerkungen

- Modell des prozeduralen Wissens, das benötigt wird, ein Problem zu lösen, basierend auf einer festen Strategie zur Regelauswahl
  $\hookrightarrow$ Welche Regeln sind bekannt?
  $\hookrightarrow$ Nach welcher Strategie werden die Regeln angewendet?
- Ohne Wissen, dass $x \times 2 = x + x$ (R_3) (könnte bei einem Grundschulkind der Fall sein) könnte das Problem nicht gelöst werden
- Eine Strategie nach der $R4$ in Schritt (2) angewendet wird, würde dazu führen, dass danach $R1$ angewendet werden muss.
  Ergebnis: $2 + 2 + 2$
  $\hookrightarrow$ insgesamt eine Regelanwendung mehr (längere Lösungszeit, umständlichere Problemlösung)
- Eine Strategie, bei der immer R_4 bevorzugt würde (keine Entfernung einer Multiplikation) würde darin resultieren, dass keine Lösung gefunden wird
- **Intelligente Tutor Systeme** für elementare Mathematik basieren auf Produktionssystem-Modellen, um Fehler zu erklären, die Schüler machen (Andersons Algebra Tutor, Kurt van Lehns Arbeiten)

Das Produktionssystem ACT

- Das System ACT (*adaptive character of thought*) ist eine kognitive Architektur, die im Wesentlichen als Produktionssystem realisiert ist.
- Bekannt wurde vor allem die Systemversion ACT* (Anderson, 1983).
- Erweiterung um Komponenten für Informationsaufnahme (*perception*) und Handlungsausführung (*motor performance*)
- ACT-R besteht aus einer Rahmentheorie über Repräsentation, Anwendung und Erwerb von Wissen.
- Dabei werden zwei Arten von Wissen – deklaratives und prozedurales Wissen – angenommen.
- ACT-R ist in der Programmiersprache Lisp implementiert.
- siehe `http://act-r.psy.cmu.edu`

Deklaratives und Prozedurales Wissen

Deklaratives Wissen

- *know that*, explizit, verbalisierbar, dem Bewusstsein zugänglich
- Faktenwissen, beispielsweise, dass ein Kanarienvogel singen kann oder dass sieben plus vier elf ergibt
- Es wird in Form sogenannter *Chunks* als Strukturen, die Fakten enthalten repräsentiert (vergleiche Schemata zur Wissensrepräsentation)

Prozedurales Wissen

- *know how*, implizit, Automatismen
- korrespondiert mit Fertigkeiten, also wie deklaratives Wissen zur Lösung von Problemen angewendet werden kann
- Es wird in Form von Produktionsregeln repräsentiert.

Turm von Hanoi in ACT

- Keine Modellierung in der aktuellen Version (ACT-R 6) online. Modellierungsbeispiel in veralteter Version (mit *goal stack*)
- Das deklarative Wissen besteht aus *Chunks*, die den Anfangszustand (`current`) und den Zielzustand (`goal`) beschreiben, sowie Fakten-Wissen über die Reihenfolge der natürlichen Zahlen eins bis vier.
- Die *Chunks* sind dabei durch abstrakte Typen beschrieben.
- Beispielsweise legt der Typ `disk` fest, dass eine Scheibe durch ihre Größe und den Stift, auf dem sie steht, beschrieben wird.
- Zusätzlich wird angegeben, in welchem Problemzustand dieser Fakt gilt.

Deklaratives Wissen beim TvH

```
(chunk-type disk size peg state)
(chunk-type peg name)
(chunk-type tower-task largest current goal)
(chunk-type move-disk disk to from other test at)
(chunk-type countfact first then)
(chunk-type encode-configuration size state)
(add-dm (disk1c isa disk size 1 peg c state current)
        (disk2c isa disk size 2 peg a state current)
        (disk3c isa disk size 3 peg b state current)
        (disk4c isa disk size 4 peg b state current)
        (disk1g isa disk size 1 peg b state goal)
        (disk2g isa disk size 2 peg a state goal)
        (disk3g isa disk size 3 peg c state goal)
        (disk4g isa disk size 4 peg c state goal)
        (fact01 isa countfact first 0 then 1)
        (fact12 isa countfact first 1 then 2)
        (fact23 isa countfact first 2 then 3)
        (fact34 isa countfact first 3 then 4)
        (fact45 isa countfact first 4 then 5)
        (a isa peg name a)
        (b isa peg name b)
        (c isa peg name c)
        (goal isa tower-task largest 4)
        (current isa chunk))
```

Ausschnitt des prozeduralen Wissens beim TvH

```
(p start-tower
"IF the goal is to solve a tower task of size =size and =size is greater than 1
 THEN set a subgoal to move disk =size checking disk =new and
      change the goal to solve a tower task of size =new"
   =goal>
      isa tower-task
      largest =size
    - largest 1
      current t
      goal t
   =fact>
      isa countfact
      first =new
      then =size
==>
   =goal>
      goal nil
      largest =new
   =newgoal>
      isa move-disk
      disk =size
      test =new
   !push! =newgoal
)
```

Ausschnitt des prozeduralen Wissens beim TvH

```
(p move
"IF the goal is move disk of size n to peg x
    and all smaller disks have been checked
 THEN move disk n to peg x and pop the goal"
   =goal>
      isa move-disk
      test 0
      from =from
      to =to
      disk =size
   =disk>
      isa disk
      size =size
==>
   =disk>
      peg =to
   !pop!
)
```

Anmerkungen

- Durch Anwendung von Produktionsregeln wird der aktuelle Anfangszustand in einen Folgezustand – der dann `current` ist – überführt, solange bis der Zielzustand erreicht ist.
- ACT-R ist als **zielgesteuertes Produktionssystem** realisiert.
- Im Bedingungsteil der Regel wird jeweils das aktuelle Problemlöseziel und die im aktuellen Zustand gültigen Fakten geprüft, im Aktionsteil werden Teilziele als neue aktuelle Ziele gesetzt und Fakten verändert.
- Die Spezialanweisungen `!push!` und `!pop!` regeln den Auf- und Abbau des Ziel-*Stacks*
- Die Regel `start-tower` kommt beispielsweise zur Anwendung, wenn das aktuelle Ziel ist, ein Problem `tower-task` mit vier Scheiben zu lösen.
- Die Variable `=size` kann durch Musterverleich mit dem *Chunk* `(goal isa tower-task largest 4)` belegt werden.
- Über den Fakt `(fact34 isa countfact first 3 then 4)` wird die Variable `=new` mit 3 belegt.

Sub-Symbolische Ebene in ACT-R

- Im deklarativen sowie im prozeduralen Gedächtnis werden Parameter verwendet.
- Diese modellieren das **rationale** Verhalten.
- Jeder *Chunk* hat einen Aktivierungswert, der die Wahrscheinlichkeit angibt, mit der er im nächsten Verarbeitungszyklus verwendet wird.
- Während eines Problemlöseprozesses werden Chunks über Aktivationsausbreitung aktiviert (und gelangen damit ins Arbeitsgedächtnis). vgl. Fan-Effekt
- Produktionsregeln haben drei Parameter: Einen Stärkewert, der die Wahrscheinlichkeit angibt, mit der diese Regel zur Anwendung kommt, eine Wahrscheinlichkeit dafür, dass die Produktion einen bestimmten Effekt erzielt, und einen Wert für die Kosten der Anwendung.
- Die Parameter werden aufgrund von Problemlöseerfahrungen verändert (Lernen).

Weitere Kognitive Architekturen

- **ACT-R** ist in der Psychologie am stärksten verbreitet.
- Ein System, das als direkter Nachfolger des General Problem Solver im Umfeld der Forschergruppe um Newell (1990) wurde, ist **SOAR** (*State, Operator And Result*). SOAR ist wie ACT ein zielgesteuertes Produktionssystem. Lernen wird als *chunking* von Produktionsregeln modelliert. SOAR wird seit einiger Zeit eher in kommerziellen Bereichen, etwa zur Simulation von Flugzeugsteuerungen, eingesetzt.
- Eine in Deutschland entwickelte kognitive Architektur ist das **Psi**-System von Dörner. Psi ist ein auf einer Theorie des Handelns basierendes System, das die Interaktion eines Agenten mit einer komplexen Umgebung realisiert und motivationale und emotionale Aspekte berücksichtigt. Es existiert auch eine Multiagenten-Version (Mäuse)
- Das System **COGENT** (Cooper, 1998) ist ein graphisches Werkzeug für Entwurf und Analyse kognitiver Modelle.