Künstliche Intelligenz

Wenn Computer Probleme lösen

Ute Schmid

`http://www.inf.uos.de/schmid`

KI erforscht,
wie man Computer Dinge machen lassen könnte,
die Menschen im Moment noch besser erledigen.

Elaine Rich

Ziele der KI

Ingenieurswissenschaftlich: Methoden zur maschinellen Lösung von Problemen (Techniken der Wissensrepräsentation, Inferenz, Suchalgorithmen, Klassifikation, Lernen, ...)

Wissenschaftliche Erkenntnis: Objektivierung und Modellierung von kognitiven Prozessen (Wahrnehmung, Denken, Schlussfolgern, Sprachverstehen, Wissenserwerb, ...)

Formalwissenschaftlich: (Weiter-)Entwicklung von Formalismen zur Beschreibung und Bewertung von Problemen und Algorithmen (Logik-Kalküle, Graphtheorie, Lernbarkeitstheorie, ...)

$\hookrightarrow$ KI ist notwendigerweise interdisziplinär!

```
++ Mathematik ++
++ Informatik ++
++ Neurobiologie ++
++ Kognitive Psychologie ++
++ Kognitive/formale Linguistik ++
++++ Philosophie der Erkenntnis ++++
```

Themengebiete der KI

Problemlösen und Planen: Suchstrategien und Techniken der Problemreduktion (Teilzielbildung)

Anwendung: Formelmanipulation, Konfigurieren, Ablaufplanung, Spiele

Inferenz: logische Deduktion, Theorembeweis, Alltagsschliessen (unscharfes/statistisches Schliessen)

Anwendung: Frage-/Antwort-Systeme, Expertensysteme

Wissensrepräsentation: Semantische Netze, Schemata, Graphen

Grundlegend für alle KI-Anwendungen, die auf Wissensbasen aufsetzen

Maschinelles Lernen: Begriffslernen, Entdeckungslernen, Regellernen

Anwendung: Objekterkennung, Klassifikation, *data mining*, Prozess-Steuerung

Automatisches Beweisen/Programmieren: Generierung oder Überprüfung von Beweisen/Programmen

Anwendung: Verifikation von Programmen

↪ Problemlösen

Was ist ein Problem?

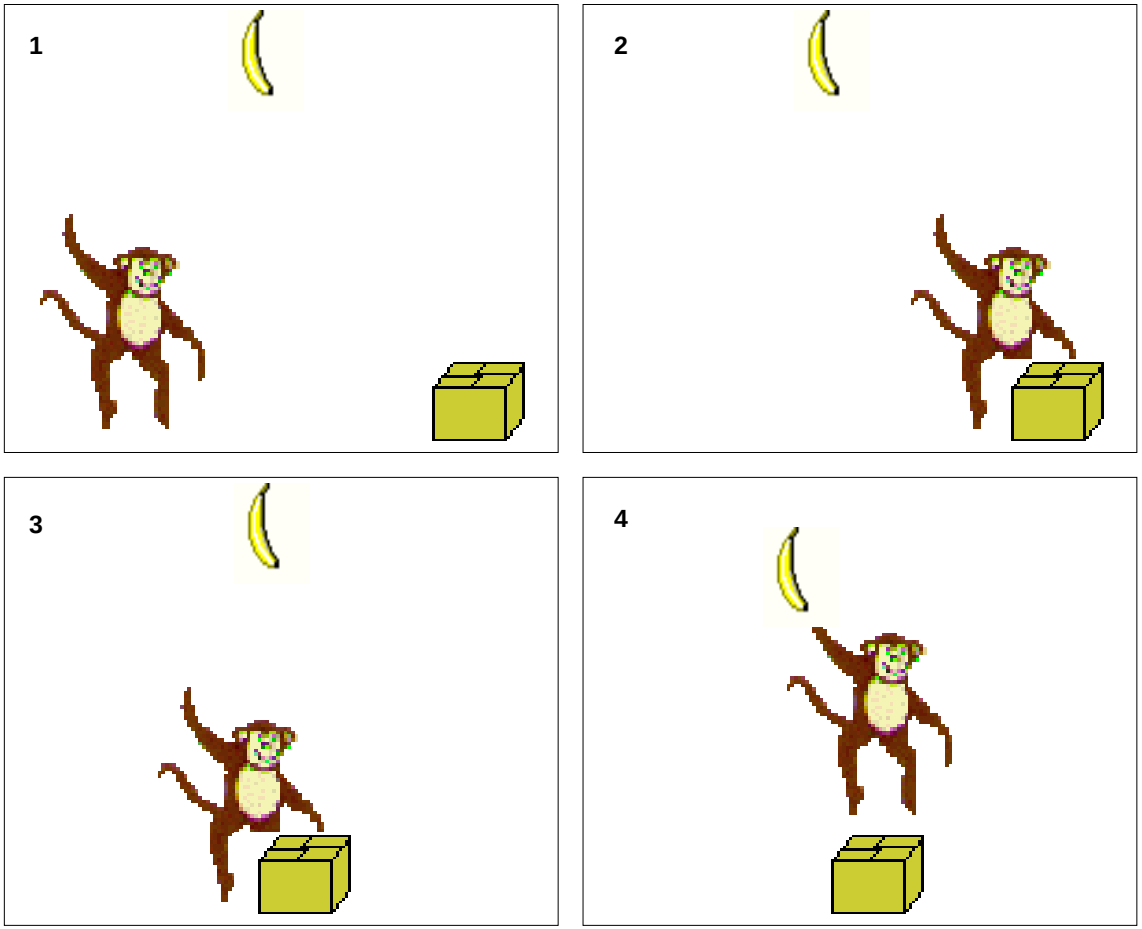
↪ Schaffe eine friedliche Welt

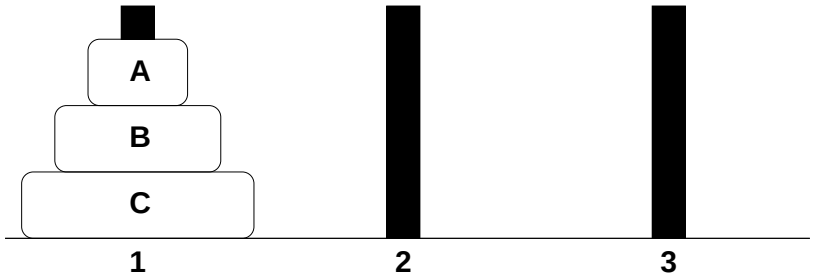
↪ Male ein schönes Bild

↪ Löse die Mathe-Hausaufgabe

↪ Gehe auf kürzestem Weg von Zuhause zur Schule und kaufe unterwegs Pausenbrot

↪ Löse eine kryptoarithmetische Aufgabe, ...






```
DONALD
GERALD
------
ROBERT
```

Definition: Problem

- Problem: Anfangszustand, Zielzustand, Operatoren
- Offenes/Geschlossenes Problem: Sind Anfangs- und Zielzustand nicht klar definiert, handelt es sich um ein offenes Problem, sonst um ein geschlossenes Problem.
- Sind die Operatoren klar definiert handelt es sich um ein Transformations-Problem, sonst um ein Synthese-Problem.

↪ Schaffe eine friedliche Welt (offenes Problem)

↪ Male ein schönes Bild (offenes Problem)

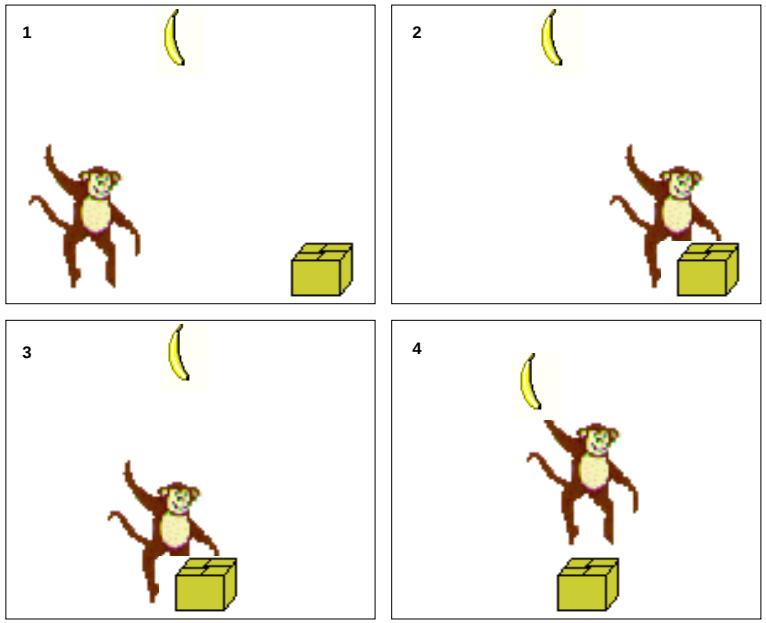
↪ Löse die Mathe-Hausaufgabe (geschlossenes Problem, Synthese-Problem)

↪ Gehe auf kürzestem Weg von Zuhause zur Schule und kaufe unterwegs Pausenbrot (geschlossenes Problem, Transformations-Problem)

↪ Löse eine kryptoarithmetische Aufgabe (geschlossenes Problem, Transformations-Problem)

Problemzustände

- **Anfangszustand:** Affe steht links auf dem Boden, Kiste steht rechts auf dem Boden, Banane hängt in der Mitte an der Decke
- **Zielzustand:** Affe hat die Banane (Kiste steht in der Mitte, Affe steht auf der Kiste)
- **Zwischenzustand:** Affe steht rechts auf dem Boden, Kiste steht rechts auf dem Boden, Banane hängt in der Mitte an der Decke



↪ Ein Zustand besteht aus einer Menge von Fakten.

↪ Es werden nur “relevante” Aspkete abgebildet (Farbe der Kiste ist irrelevant).

↪ Abgeschlossenheits-Annahme: In einem Zustand nicht genannte Fakten werden als nicht gegeben angenommen (Im Anfangszustand hat der Affe die Banane nicht).

↪ Es kann mehrere Zielzustände geben: Zustände, in denen der Affe die Bananne hat.

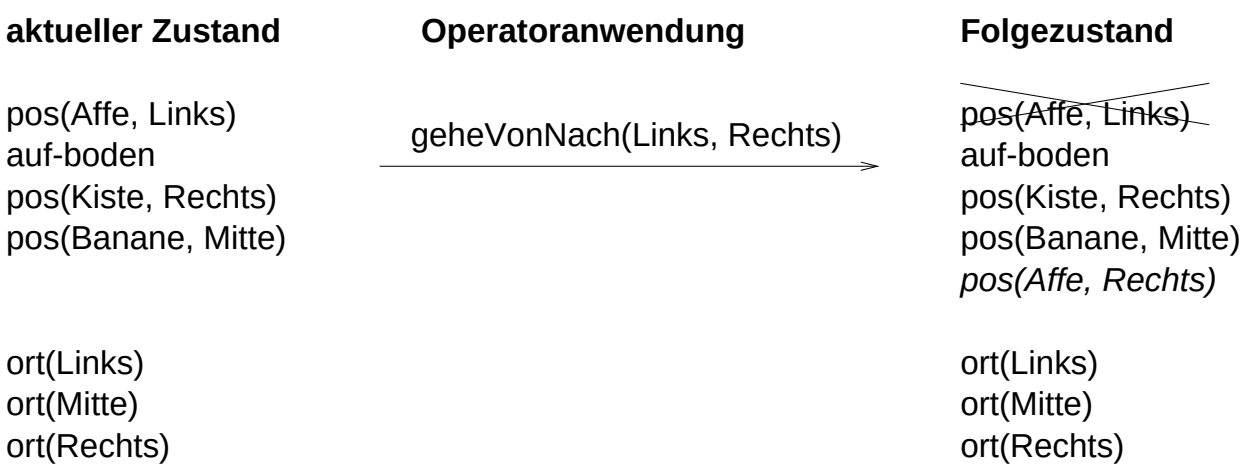
```
pos(Affe, Rechts), aufBoden, pos(Kiste, Rechts), pos(Banane, Mitte)
ort(Links), ort(Mitte), ort(Rechts)
```

Problemlöse-Operatoren

- Allgemeine Form: WENN <Anwendungsbedingung> DANN <Aktion>
- Anwendungsbedingung: Fakten, die erfüllt sein müssen
- Aktion: Handlung, die ausgeführt wird, definiert über ihre Auswirkung
  "Addiere" neue Fakten, die nach Aktion gelten, "Lösche" Fakten, die nach Aktion nicht mehr gelten
- Anmerkung: hier Operator-*Schema* mit Variablen (allgemeinste Form, um Operatoren zu repräsentieren)

GeheVonNach(x,y):

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



Das Affe-Banane Problem

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

Zielzustand: pos(Affe, Mitte), auf-kiste, pos(Kiste, Mitte), hat-banane

Raumpositionen: ort(Links), ort(Mitte), ort(Rechts) [statisch]

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):

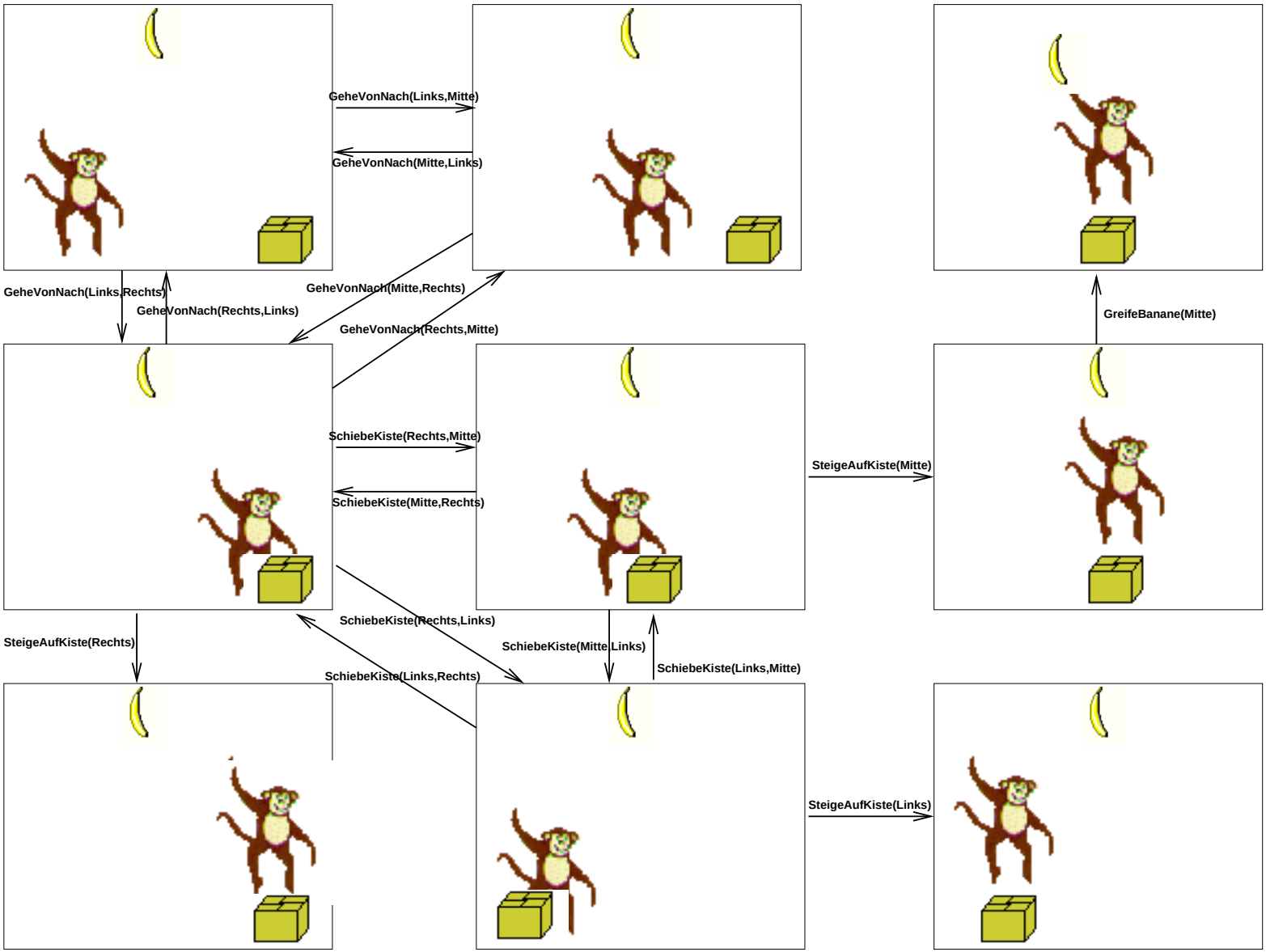
Anwendungsbedingung: ort(x), pos(Affe, x), pos(Banane, x), auf-kiste

Auswirkung: ADD hat-banane

Problemlösen

- **Problemlösung**: Folge von Aktionen, um einen Anfangszustand in einen Zielzustand zu transformieren. Alternativ: Optimale Folge (möglichst wenig Aktionen, möglichst "billig" bei Aktionen mit unterschiedlichen Kosten)
- **Problemlösen**: *Suche* nach einer Folge von Aktionen, um einen Anfangszustand in einen Zielzustand zu transformieren.
- Wenn die Lösung bereits bekannt ist, ist Problemlösen unnötig.
  Beispiel: rekursive Berechnungsvorschrift für Turm von Hanoi
- **Problemraum**: Alle möglichen Zustände eines Problems bilden einen Problemraum – Graph mit Knoten als Zuständen und Aktionen als Kanten.
- Ein Problemraum ist nicht explizit gegeben (weder beim menschlichen Problemlöser noch im Rechner): Beim Problemlösen werden Teile des Problemraums in Form eines **Suchbaums** erzeugt.

Problemraum für das Affe-Banane-Problem



Uninformierte Suche

- Einfachste Strategie: Versuch-und-Irrtum (jeweils zufällig eine Aktion ausführen).
- Systematische Strategien: Tiefensuche, Breitensuche
- Grundidee des Tiefensuchalgorithmus:
  - Gegeben ist eine Ordnung auf den Operatoren (Präferenz).
  - Für den aktuellen Zustand wird diejenige Aktion ausgewählt, die anwendbar ist und die höchste Präferenz besitzt.
  - Der Folgezustand kann ein bereits besuchter Zustand sein (**Zyklus**):
    Affe könnte unendlich oft von Links zur Mitte und wieder zurück laufen.
    $\hookrightarrow$ Zyklen müssen erkannt und vermieden werden.
  - Der Folgezustand kann ein Zustand sein auf den keine weiteren Aktionen annwendbar sind (**Sackgasse**):
    $\hookrightarrow$ Die letzte Aktion (die zur Sackgasse führte) muss zurückgenommen werden (*backtracking*).

Tiefensuch-Algorithmus

1. Aktueller Zustand = Anfangszustand
2. **Solange** Aktueller Zustand $\neq$ Zielzustand **Und** noch unbesuchte Zustände vorhanden:
   (a) Wähle nächste anwendbare Aktion aus.
   (b) **Wenn** keine Aktion anwendbar ist,
   Dann nimm die letzte Aktion zurück (markiere sie als bereits versucht) und setze Aktueller Zustand auf den vorherigen Zustand,
   Sonst wende Aktion an und setze Aktueller Zustand auf den resultierenden Zustand.
   (c) **Wenn** Aktueller Zustand bereits auf dem Lösungsweg liegt,
   Dann nimm die letzte Aktion zurück (markiere sie als bereits versucht) und setze Aktueller Zustand auf den vorherigen Zustand.
3. **Wenn** Aktueller Zustand $=$ Zielzustand
   Dann Gib Lösung aus, **Sonst** Melde ‘‘Problem nicht lösbar’’.

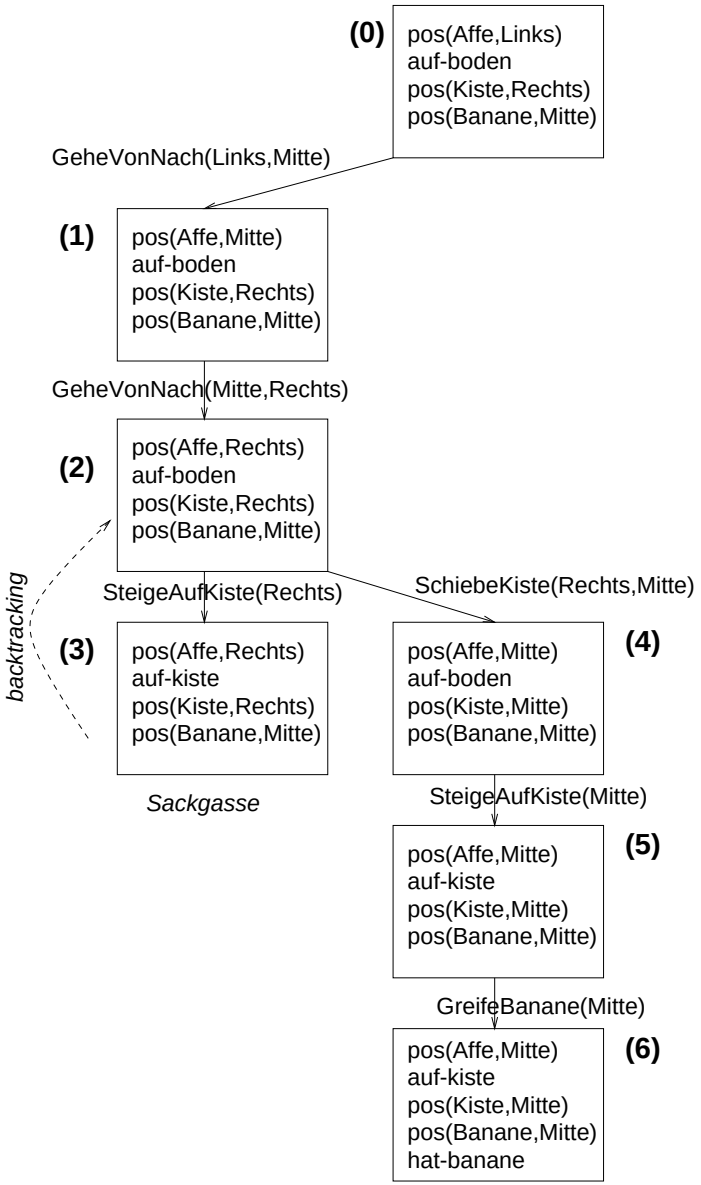
Tiefensuche beim Affe-Banane Problem

Ordnung auf den Operatoren:

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

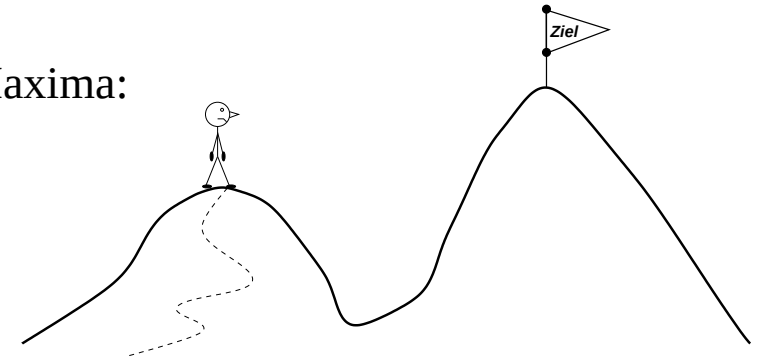
Bevorzugung von Orten:
(1) Mitte,
(2) Rechts,
(3) Links

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



Heuristische Suche

- Menschen suchen im Allgemeinen nicht systematisch nach einer Problemlösung. Sie haben häufig Vorannahmen/Vermutungen, welche Aktionen eher zum Erfolg führen als andere: **Heuristik**, Erfahrung, Hintergrundwissen, ...
- In der Informatik gibt es eine grosse Klasse von Problemen, die nicht effizient mit systematischen Suchverfahren gelöst werden können (riesige Suchbäume, exponentieller Aufwand).
- KI-Algorithmen: Verwendung von Heuristiken
  z. B. Schachprogramme: Mass zur Bewertung einer Spielstellung nach ihren Chancen, zum Sieg zu führen
- Einfachste heuristische Strategie: **Hill Climbing**
- Idee der Unterschiedsreduktion: Wähle jeweils die Aktion, die den Unterschied zwischen dem aktuellen Zustand und dem Zielzustand am stärksten verringert.
- Problem bei heuristischen Strategien: Wenn man Pech hat, findet man die Lösung nicht (man “verrennt sich”).
- Problem der lokalen Maxima:



Heuristiken

- Das Aufstellen einer Heuristik bedarf einer Problemanalyse.
- Naive Bewertungsfunktionen:
  - Affe-Banane Problem: ein Punkt für Affe bei der Kiste, zwei Punkt für Kiste in Mitte, drei Punkte für Affe und Kiste in Mitte, vier Punkte für Affe auf der Kiste in der Mitte.
  - Hobbits & Orcs: Zahl der Personen auf dem Zielufer (Problem: Es müssen einmal zwei Personen zurück transportiert werden, um das Problem zu lösenn)
  - Achter-Puzzle: Zahl der korrekt platzierten Plättchen (korrekte Sequenz ist wichtiger)
- Es ist nicht einfach, sinnvolle Heuristiken zu entwickeln!

möglicher Anfangszustand

2	1	6
4		8
7	5	3

Zielzustand

1	2	3
4	5	6
7	8	

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

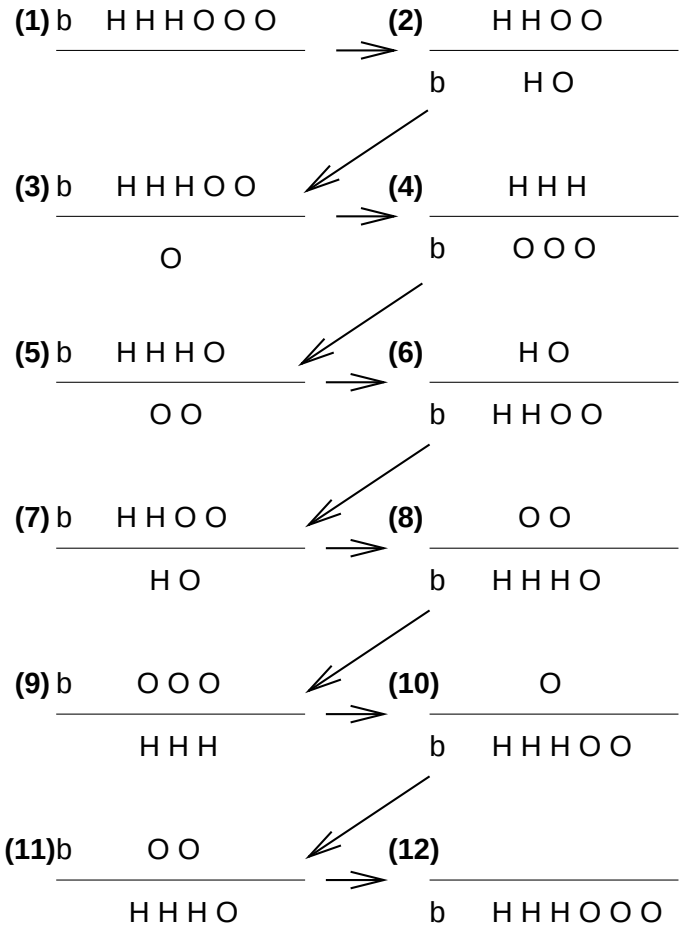
Das Hobbits & Orcs Problem

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:



Anmerkungen:

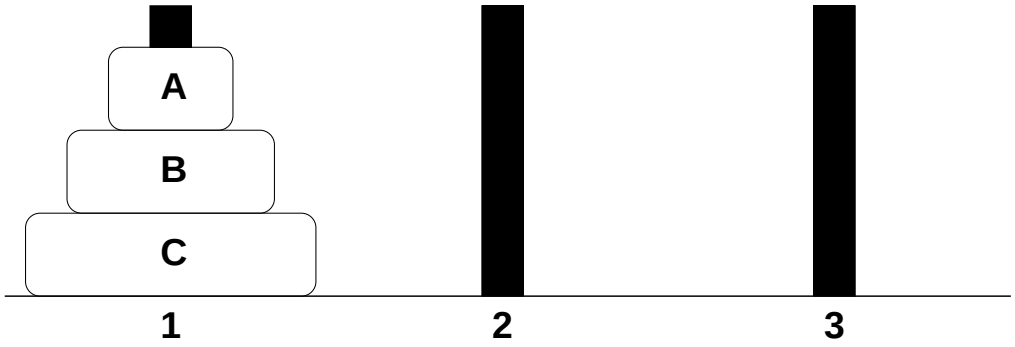
- Bester heuristischer Algorithmus: A^* (Nils Nilsson)
- Nur wenn die heuristische Funktion die tatsächliche Distanz zum Ziel *unter*schätzt sind solche Algorithmen effizient (mathematisch beweisbar).
- KI-Planung: Allgemeine Sprache zur Repräsentation von Problemen, effiziente Algorithmen (Graphplan, HSP, ...)
- Zweijährlicher Wettbewerb von Planern:
  `http://www.dur.ac.uk/d.p.long/competition.html`
- Aktuelle Forschungsbereiche: Fallbasierte Strategien (Anpassung von Lösungen bekannnter Probleme), Lernen aus Problemlöseerfahrung

Und was hat das mit KI zu tun?

- Was, KI ist so langweilig? Programmieren von Suchalgorithmen soll intelligentes Verhalten abbilden?
- KI befasst sich damit, Probleme auf Computern lösbar zu machen, die bislang nicht autmatisch gelöst werden können.
- Teilgebiet der KI: Kognitive Modellierung – Versuch der Beschreibung menschlichen Problemlöseverhaltens basierend auf empirischen Ergebnissen der kognitiven Psychologie.
- Häufig führt die erfolgreiche Konstruktion von Algorithmen zur “Demystifizierung” eines Problembereichs!

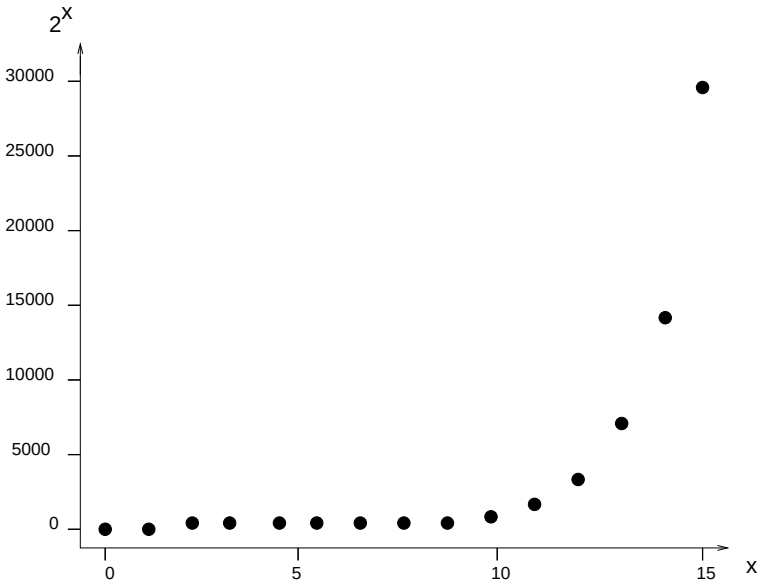
Die Türme von Hanoi

- Der Legende nach sind die Mönche von Hanoi seit Jahrhunderten damit beschäftigt, einen Turm aus 64 Scheiben von einer Position auf eine andere zu versetzen. Nach Meinung der Mönche soll die Welt untergehen, wenn der Turm komplett versetzt ist.
- Es gibt drei Stäbe.
- Ursprünglich steht der Turm auf einem dieser Stäbe.
- Ziel ist es, den Turm auf einem anderen Stab zu haben.
- Der dritte Stab darf als Zwischenablage genutzt werden.
- Jede Scheibe hat eine unterschiedliche Grösse.
- Eine Scheibe darf nur auf eine grössere Scheibe oder auf einen leeren Stab gelegt werden.
- Das 3-Scheiben Problem:



Problemanalyse für die Türme von Hanoi

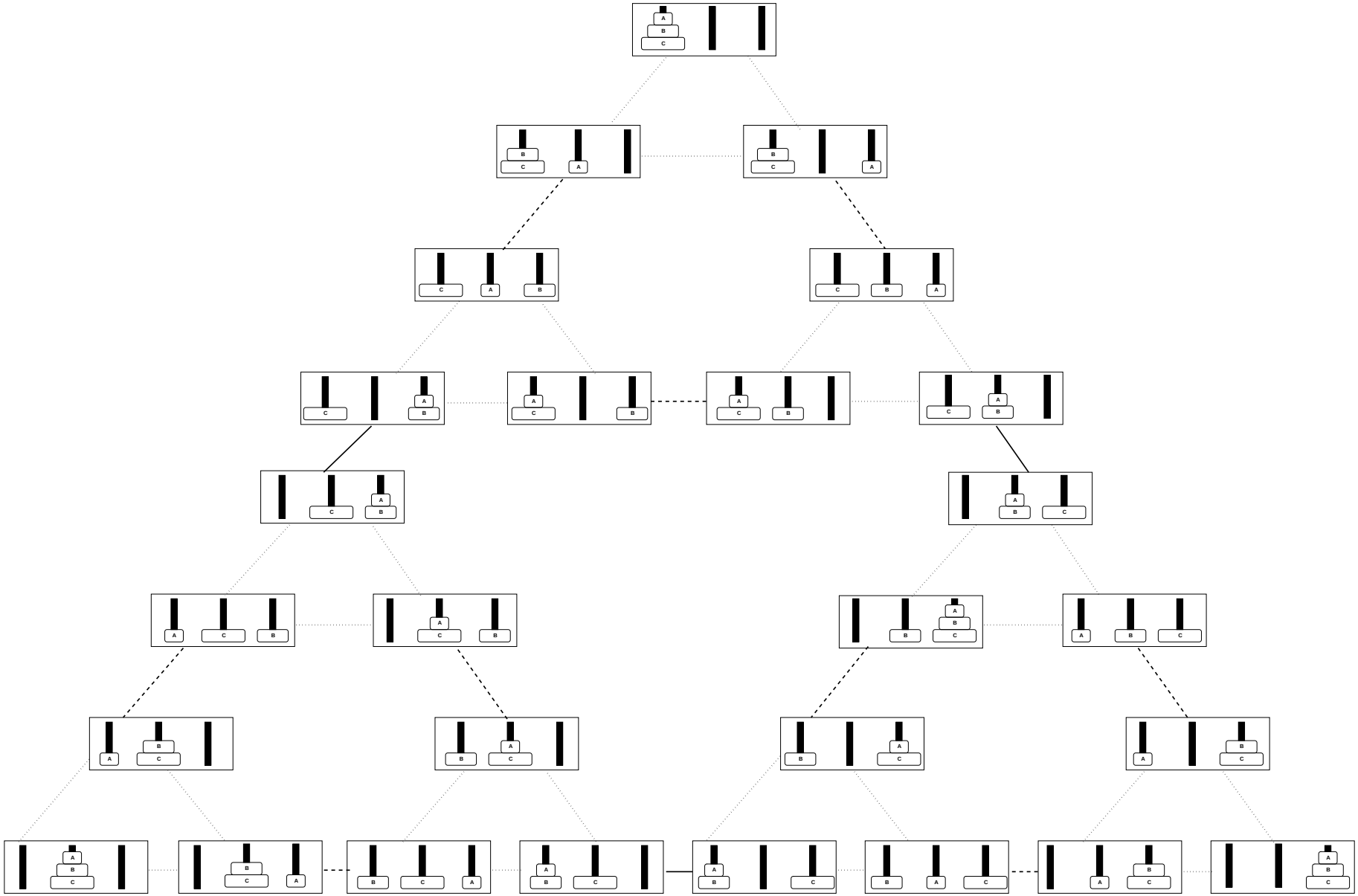
- Anzahl der Zustände: 3^n bei drei Stäben und n Scheiben
  (Wahrscheinlichkeitstheoretisches Gesetz der *Variation*)
- Minimale Anzahl von Aktionen zur Lösung des n-Scheiben Problems: $2^n - 1$
  - 1-Scheiben Problem: eine Bewegung
  - 2-Scheiben Problem: eine Bewegung für die grosse, zwei Bewegungen für die kleine Scheibe
  - 3-Scheiben Problem: eine Bewegung für die grosse, zwei für die mittlere, vier für die kleine
  - ... $2^0 + 2^1 + 2^2 + \ldots + 2^{n-1}$
- 64-Scheiben Problem: fünf Trillionen Jahre, wenn alle 10 Sekunden eine Scheibe bewegt wird.



Exponentielles Wachstum

Scheiben	Zustände	Aktionen
1	3	1
2	9	3
3	27	7
4	81	15
5	243	31
6	729	63
7	2187	127
8	6561	255
9	19683	511
10	59049	1023

Problemraum für das 3-Scheiben Problem



(Bewegungen von *A* gepunktet, Bewegungen von *B* gestrichelt, Bewegungen von *C* durchgezogen)

... Die Türme von Hanoi

- das Türme von Hanoi-Problem ist nicht mit systematischer Suche lösbar: exponentielles Wachstum!
- Heuristische Suchverfahren sind notwendig.
- Alternativ: Identifizieren eines speziellen Lösungsalgorithmus (Psychologische Studien zeigen, dass vor allem mathematisch begabte Schüler dazu in der Lage sind.)
- Das Türme von Hanoi Problem hat eine sehr regelmässige Struktur: Problemraum als “ineinander verschachtelte Dreiecke”.
  $\hookrightarrow$ Rekursive Berechnungsvorschrift

```
hanoi(n, von, via, nach) =
WENN n = 0
DANN tue nichts
SONST
    hanoi(n-1, von, nach, via)
    bewegeScheibe(von, nach)
    hanoi(n-1, via, von, nach).
```

Rekursive Probleme

- Rekursion meint Selbstbezüglichkeit, also die Definition eines Problems unter Bezug auf sich selbst.
- Die wohl bekannteste rekursive Definition einer mathematischen Funktion ist die der Fakultät:

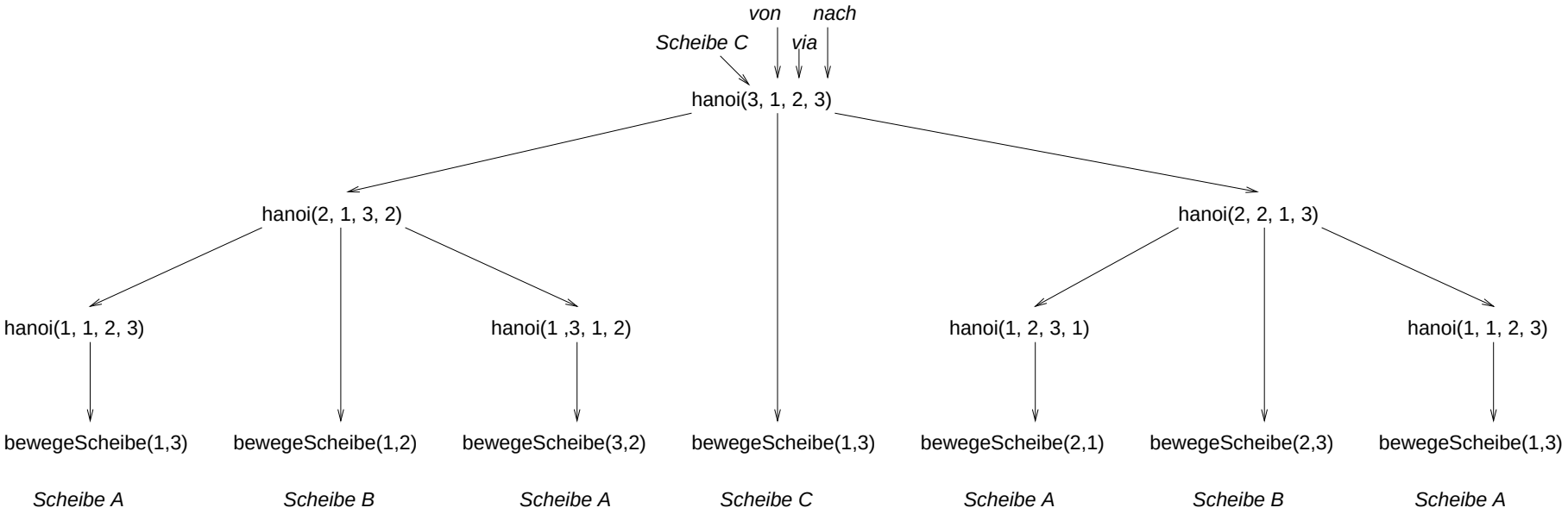
$$faku(x) = \begin{cases} 1 & wenn\ x = 0 \\ x \cdot faku(x-1) & sonst. \end{cases}$$

Die Fakultät einer natürlichen Zahl x berechnet sich durch Multiplikation von x mit der Fakultät der Zahl $x - 1$. Beispielsweise gilt:

$$\begin{aligned} faku(0) &= 1 \\ faku(1) &= 1 \cdot faku(0) = 1 \cdot 1 = 1 \\ faku(2) &= 2 \cdot faku(1) = 2 \cdot 1 \cdot 1 = 2 \\ faku(3) &= 3 \cdot faku(2) = 3 \cdot 2 \cdot 1 \cdot 1 = 6 \\ faku(4) &= 4 \cdot faku(3) = 4 \cdot 3 \cdot 2 \cdot 1 \cdot 1 = 24. \end{aligned}$$

- Um also die Fakultät der Zahl 4 auszurechnen, muss man auf das Ergebnis *faku(3)* zurückgreifen. Ist dieses Ergebnis nicht bekannt, so muss nun zunächst *faku(3)* berechnet werden, etc.
- Für *faku(0)* ist das Ergebnis direkt definiert.
- Das Ergebnis 1 kann an die wartende Funktion *faku(1)* zurückgegeben werden, die nun berechnet werden kann und ihr Ergebnis an *faku(2)* weitergibt, etc.
- Es ist wichtig, dass es einen Fall gibt, für den das Ergebnis *direkt* ermittelt werden kann. Ansonsten würde man bei der Berechnung endlos “in die Tiefe” steigen (Nicht-Termination).

Abarbeitung der rekursiven Funktion ‘hanoi’



Literatur für den Einstieg

R. Kurzweil: Das Zeitalter der Künstlichen Intelligenz. Hanser. (schöner Bildband, deckt alle Bereiche der KI anschaulich ab)

P. Winston. Künstliche Intelligenz. Addison-Wesley. (alle Suchalgorithmen in einfacher Form; klassisches KI-Lehrbuch)

U. Schmid. Computermodelle des Problemlösens. In J. Müsseler und W. Prinz (Hrsg.), Lehrbuch Allgemeine Psychologie, Spektrum Verlag. (2003, siehe `http://www.inf.uos.de/schmid/pub-ps/lb-end.pdf`)

U. Schmid, U. & M. Kindsmüller. Kognitive Modellierung: Eine Einführung in die logischen und algorithmischen Grundlagen. Spektrum Verlag. (einfacher Einstieg in KI-Formalismen)