Technische Universität Berlin

Einführung in die
Kognitive Modellierung –

Alternative Architekturen Soar & 4CAPS

REFERENT_INNEN:

JETTE BEIßER, NIKLAS FASCHING, MIRJA HOLLMANN, NIKOLAI V. SEYDLITZ

Ablauf

1. Einführung
2. Soar
   a. Künstliche Intelligenz
   b. Entwicklung
   c. Struktur und Processing Cycle
   d. Visual Soar / Soar Java Debugger
   e. Generell
   f. Working Memory
   g. Rules
   h. Beispiel
3. 4CAPS
   a. Einführung
   b. Historie
   c. Operating Principles: Theorie und Implementierung
   d. Zusammenfassung
   e. Framework
   f. Modell-Beispiel
   g. Anwendungsgebiete

Ablauf

1. **Einführung**
2. Soar
   a. Künstliche Intelligenz
   b. Entwicklung
   c. Struktur und Processing Cycle
   d. Visual Soar / Soar Java Debugger
   e. Generell
   f. Working Memory
   g. Rules
   h. Beispiel
3. 4CAPS
   a. Einführung
   b. Historie
   c. Operating Principles: Theorie und Implementierung
   d. Zusammenfassung
   e. Framework
   f. Modell-Beispiel
   g. Anwendungsgebiete

Einführung - Kognitive Architekturen

Ziel

- Modellierung von menschlicher Kognition
- nicht AGI (**A**rtificial **G**eneral **I**ntelligence)

Grundbausteine

- Memory
- Learning

Kategorisierung

- Symbolic
- Sub-Symbolic / Emergent
- Hybrid

Einführung - Kategorisierung

Grundlegende Funktionsweise

- Input => Processing => Output

Symbol

- Abbildung eines Konzeptes (z.B. eine Zahl)
- Hat für den Mensch Bedeutung
- Der Computer erreicht kein echtes Verständnis
- Manipulation über vom Menschen einprogrammierte Algorithmen

Einführung - Kategorisierung

Symbolic

- High-Level: Arbeit mit Symbolen
- Fokus: Informationsverarbeitung => Was passiert nach der Identifizierung eines Stimulus
- Weniger autonom, mehr vom Programmierer bestimmt
- Weniger realistisch
- Komplexe kognitive Funkionen (Planung, bedachte Handlung)
- Intuitiv

➢ Wie kommt man von Wahrnehmung zu Symbolen?

➢ Arbeitet unser Gehirn überhaupt mit Symbolen?

Einführung - Kategorisierung

Sub-Symbolic / Emergent

- Low-Level: Arbeit mit … Sub-Symbolen – z.B. Aktivierung
- Fokus: Identifizierung von Stimuli
- Schwerer komplexe kognitive Funktionen umzusetzen
- Autonomer
- Realistischer
- Wenig intuitiv

- Ist es überhaupt möglich emergent komplexe kognitive Funktionen zu modellieren?

Einführung - Kategorisierung

Hybrid

- Kombination von Symbolic & Sub-Symbolic
- Realistisch & Komplex

Ablauf

1. Einführung
2. **Soar**
   a. **Künstliche Intelligenz**
   b. **Entwicklung**
   c. **Struktur und Processing Cycle**
   d. Visual Soar / Soar Java Debugger
   e. Generell
   f. Working Memory
   g. Rules
   h. Beispiel
3. 4CAPS
   a. Einführung
   b. Historie
   c. Operating Principles: Theorie und Implementierung
   d. Zusammenfassung
   e. Framework
   f. Modell-Beispiel
   g. Anwendungsgebiete

Soar

John E. Laird, University of Michigan: „Vater“ von Soar

Seit 1982 als Experimentalsoftware in Benutzung

Erste Veröffentlichung 1983

Erste umfangreiche Präsentation Dezember 1986: Soar: An Architecture for General Intelligence (Laird, J.E., Newell, A. & Rosenbloom, P.S.)

- Version Soar 4
- Idee: reine symbolische Verarbeitung, LM als prozedurales Wissen, Modellierung von Verhaltens- und vor allem Lernprozessen

Laird, Newell, Rosenblooom (1986)
http://ai.eecs.umich.edu/people/laird/

Soar – Ziel: General intelligent agent

Große Bandbreite an Aufgaben lösen

- Hochüberlernte Routineaufgaben
- Schwere, komplexe, offene Aufgaben
- Problemlösemechanismen darstellen
- Interaktionsstelle mit der Außenwelt abbilden
- Lernprozesse ermöglichen

Möglichst alle entscheidenden Wissensformen des Menschen repräsentieren

- Prozedural
- Semantisch
- Episodisch
- ikonisch

Soar – Künstliche Intelligenz

Laird

- PhD in Computer Science
- Teil des Artificial Intelligence Lab an der University of Michigan

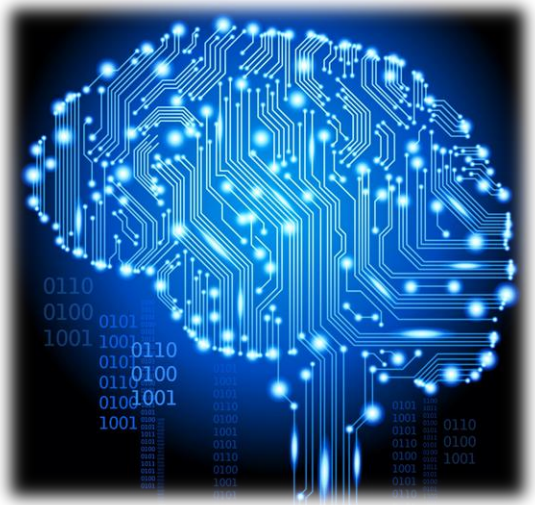
Wichtigstes Anwendungsfeld, für das Soar auch entwickelt wurde

➢Forschung mit dem Ziel und Entwicklung einer (approximierten) künstlichen Intelligenz

Ultimative KI:

- Vollständig rational
- Nutzt alle verfügbaren Informationen für jede ihr gestellte Aufgabe

➢Entspricht **nicht** der menschlichen Intelligenz!



http://soar.eecs.umich.edu/
http://ai.eecs.umich.edu/people/laird/

Soar – Künstliche Intelligenz

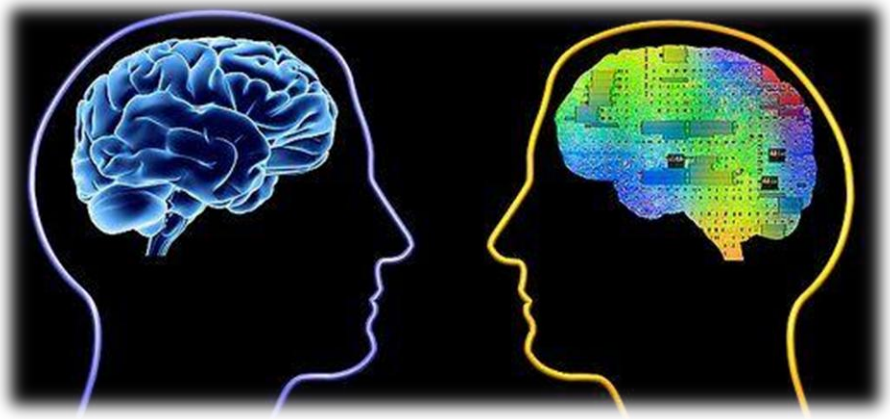
Stattdessen: Entscheidungsprozesse anhand von

- Verfügbaren sensorisch wahrgenommenen und interpretierten Informationen
- Aktuell durch vorhergehende Prozesse im WM verfügbaren Informationen
- Wichtigem aus dem LM abgerufenem Wissen

➢Entscheidungsprozesse zur Laufzeit, flexible (nicht rigide) Lösungen

➢Reaktion auf komplexe, dynamische Umwelten

Modellierung einer **menschlichen** künstlichen Intelligenz!



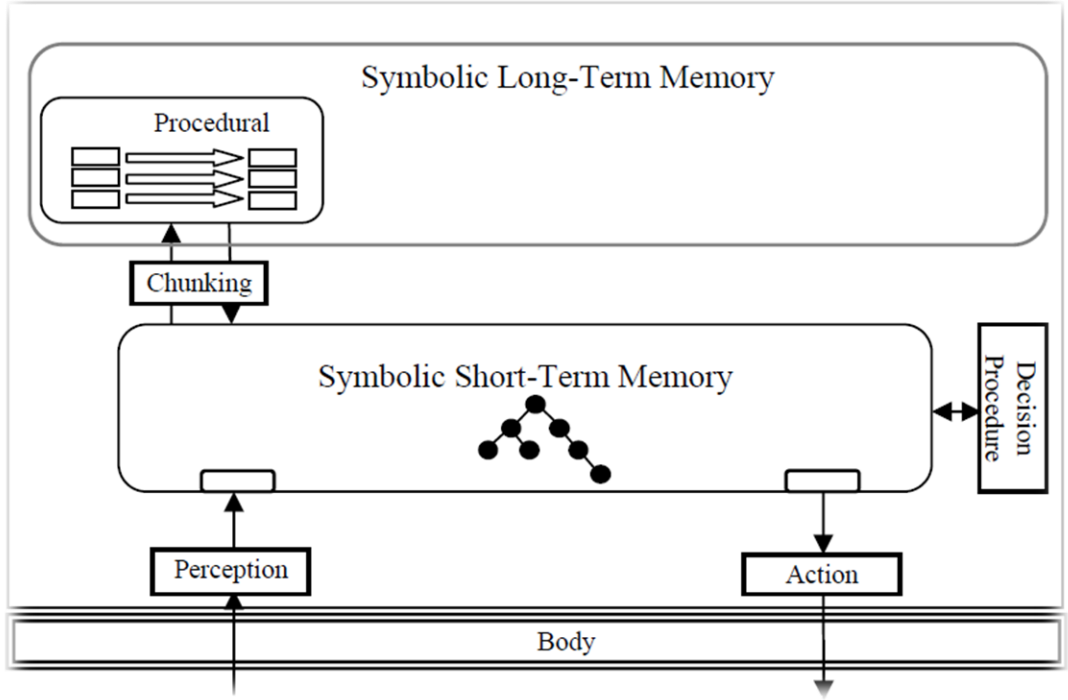
Soar – Anwendung

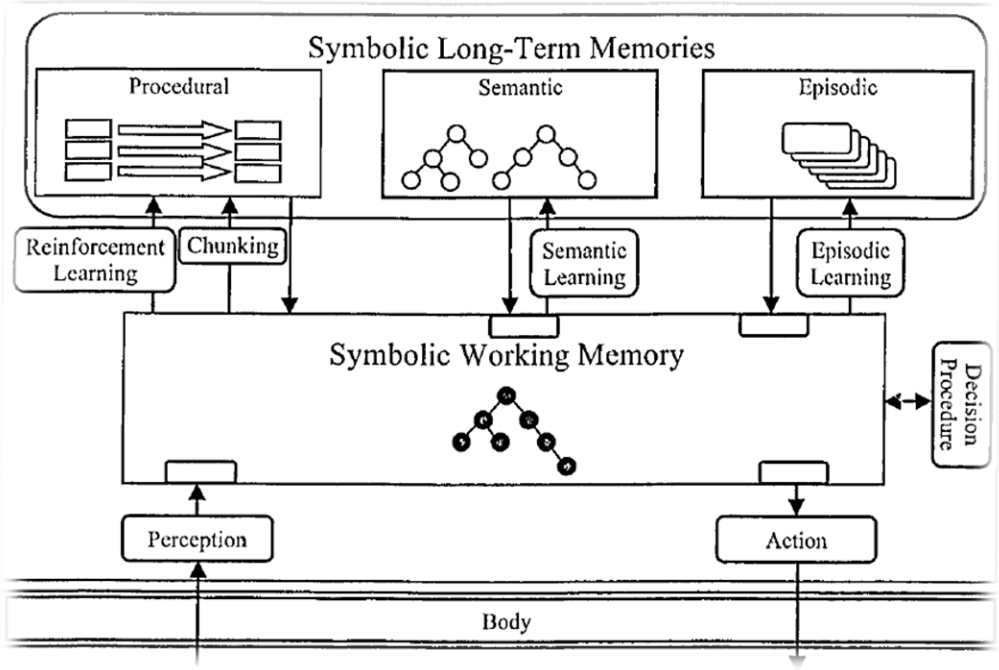
Anwendungsgebiete

- Militäranwendungen
  - Einsatzsimulation
- Spieleentwicklung
  - Intelligente, realistisch agierende Bots, adaptive Skill-Level Gestaltung
  - Z.B. Quake, Descent
- Simulationen
  - Z.B. Im Luftfahrtsektor
- KI-Forschung
  - Robo-Soar
  - Lernende Roboter
  - Modellierung von (sprachbasierten) Lernprozessen

Soar – Entwicklung

Auf dem Weg zum Modell für KI von Soar 1 (1982) bis Soar 9 (2008)





Soar – Entwicklung

1-8 folgte ähnlichen Prinzipien

- Symbolisches LM
  - ➢Vollständig prozedural (Produktionsregeln)
- Chunking als Lernmechanismus
- WM als symbolische Graphenstruktur, repräsentiert Objekte mit Eigenschaften und Beziehungen
  - Bewertung der aktuellen Situation durch Wahrnehmung und Rückgriff auf Wissen aus dem LM
  - Auswahl von Operatoren die Handlungen repräsentieren und zur Handlungsauswahl/-initialisierung verwendet werden

➢Produktionsregeln erlauben flexibles, kontextabhängiges Wissen über Handlungen, die es unter bestimmten Bedingungen auszuführen gilt (matching and firing rules)

Soar – Entwicklung

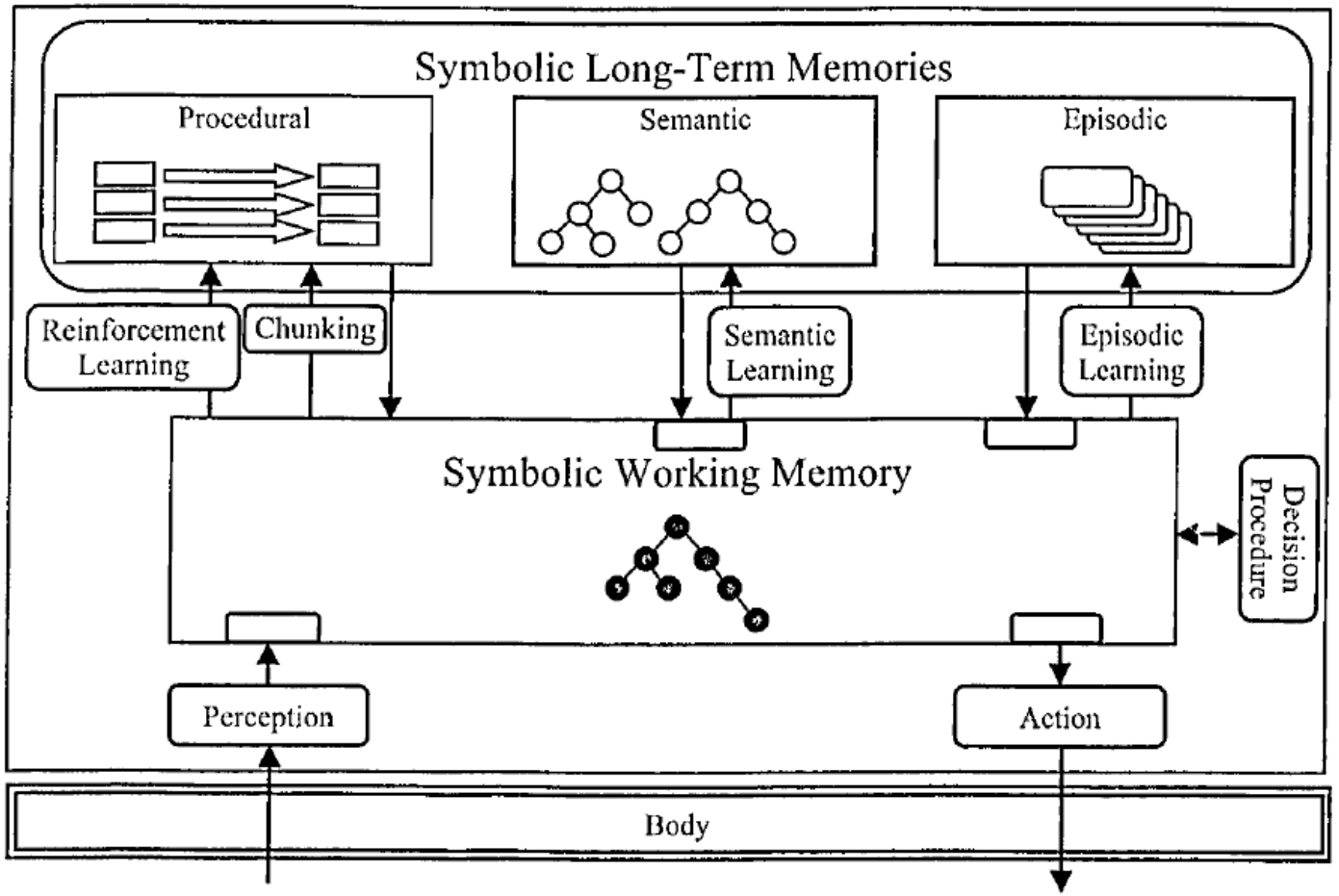
Vielzahl bedeutender Erweiterungen in Soar 9

Auswahl:

- 3 verschiedene Formen symbolischen Langzeitwissens
  - Prozedural
  - Semantisch
  - Episodisch
- 4 entsprechende Lernprozesse
  - Chunking (wie bisher)
  - Sematisches Lernen
  - Episodisches Lernen
  - Verstärkungslernen (Integration emotionaler Prozesse und Bedürfnisse)

Soar – Struktureller Aufbau

Vorgestellte Struktur: Soar 9



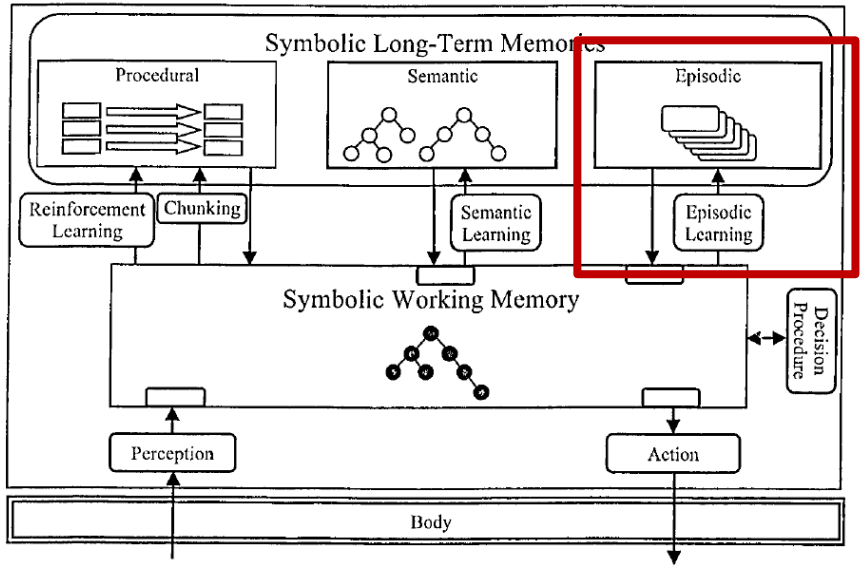
Soar – Struktureller Aufbau

Episodisches LM

- Repräsentation von Informationen/Objekten, die gleichzeitig im WM aktiv waren
- Speicherung von Kontext, zeitlichen Zusammenhängen von Objekten

➢ Retrieval wenn Teil eines kontextuellen Zusammenhangs im WM aktiviert wird

➢ Bestes Match wird gesucht und mit aktiviert



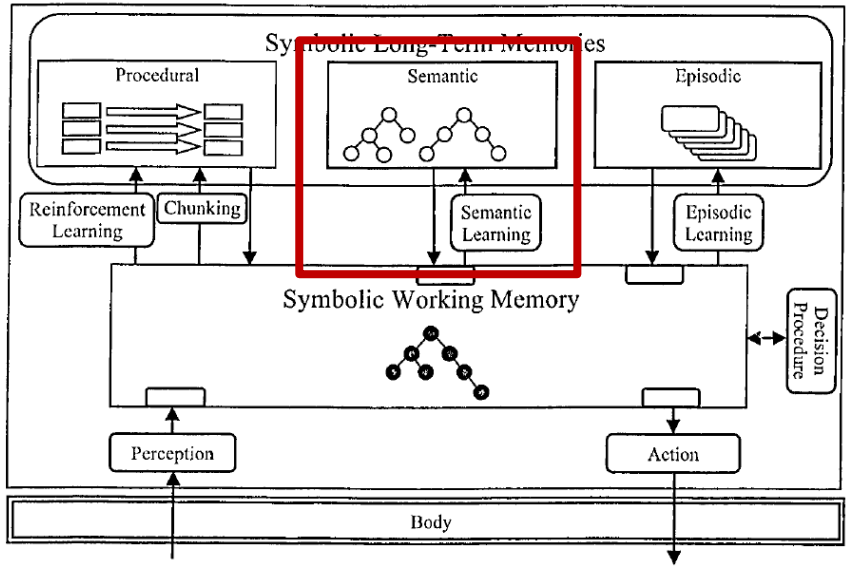
Soar – Struktureller Aufbau

Semantisches LM

- Repräsentation von deklarativem Faktenwissen über die Welt

➢Vgl. Zu Act-R

➢Erlaubt es Agenten zu schlussfolgern und Faktenwissen in Entscheidungsprozesse einzubeziehen



Soar – Struktureller Aufbau

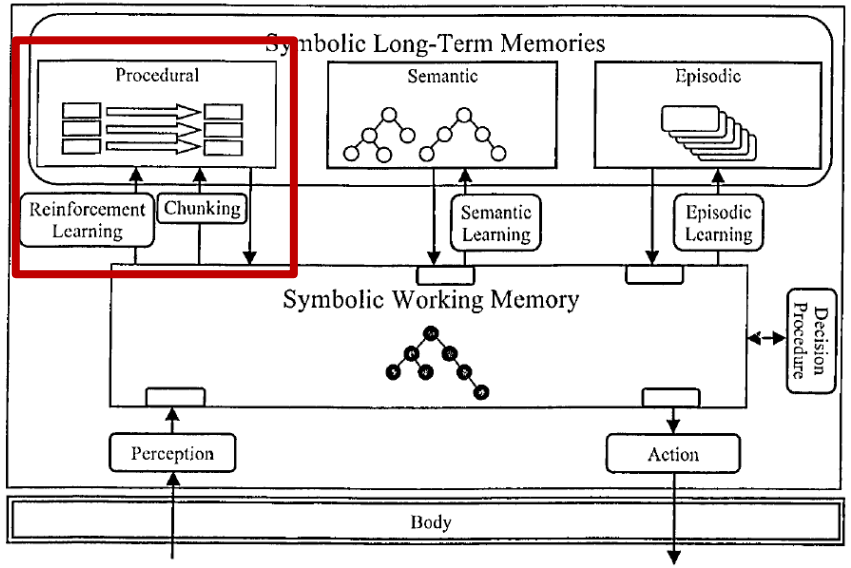
Prozedurales LM

- Repräsentation von prozeduralem Wissen
- Liste aller dem System bekannten Produktionsregeln

➢Vgl. Zu Act-R

➢Führt System von einem Zustand in einen neuen über

➢Ausgehend von aktuellen Informationen aus dem WM



Soar – Struktureller Aufbau

Aus LM-Formen resultieren zugehörige Lernformen

Älteste (ursprüngliche einzige) Lernform: Chunking

- „Erklärungsbasiertes“ Lernen
- Kreation und Speicherung einer neuen Produktionsregel

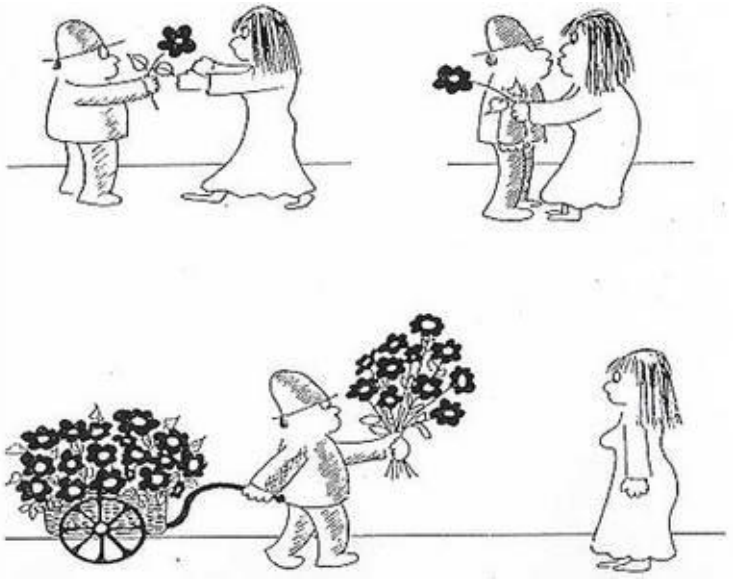
- Wird keine passende Produktionsregel zur Erfüllung eines Ziels gefunden: „Sackgasse“ (Impasse)
- Erstellung eines Subziels: Weg aus der Sackgasse finden
- Wird ein Weg gefunden: Bildung und Speicherung eines Chunks

Soar – Struktureller Aufbau

Neu u.a.: Verstärkungslernen

- Aktionsauswahl anpassen zur Gewinnmaximierung (Belohnung, positiver Outcome)

- Symbolisches Wissen über die Ergebnisse einer Handlungsoption konnte nicht angepasst werden
  - ➢ Numerische Repräsentation von Präferenzen mit maximiertem erwarteten Wert eines Operators eingeführt
- Ergebnis einer Handlung führt zu Update des erwarteten Wertes eines Operators

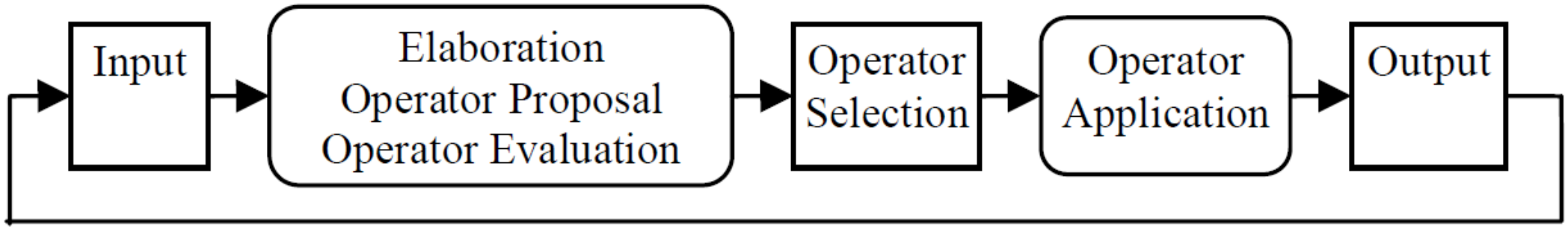


Soar – Struktureller Aufbau

Verstärkungslernen komplementär zu Chunking

- keine Produktion neuer Regeln/Operatoren
- erfahrungsbasierte Anpassung des vorhandenen prozeduralen Wissens

Soar – Processing Cycle



Laird (2008)
Laird, Newell, Rosenbloom (1987)

Soar – Processing Cycle

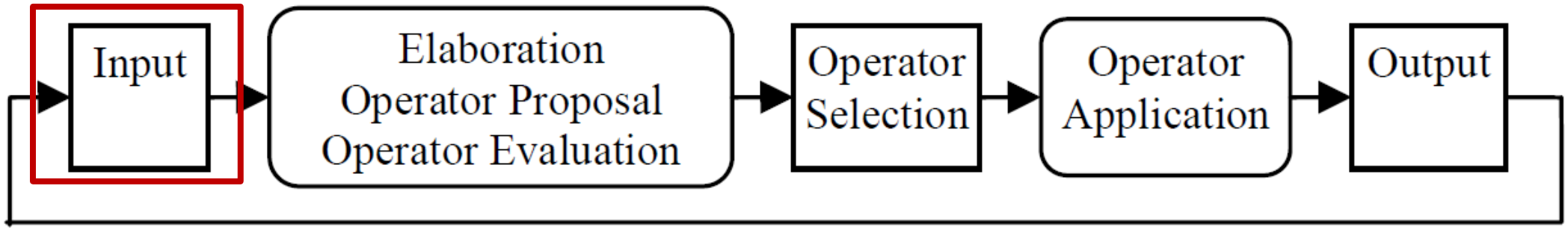
Produktionsregeln schlagen Operatoren vor

- ➢Imitation menschlicher Gedankengänge
- Aktueller Zustand des WM matched auf Produktionsregeln
- Schlagen Operator vor
- ➢Frage: Welcher Operator wird tatsächlich ausgelöst?

Laird (2008)
Laird, Newell, Rosenbloom (1987)

Soar – Processing Cycle

Wahrnehmung wird verarbeitet → Verändert Inhalt im WM

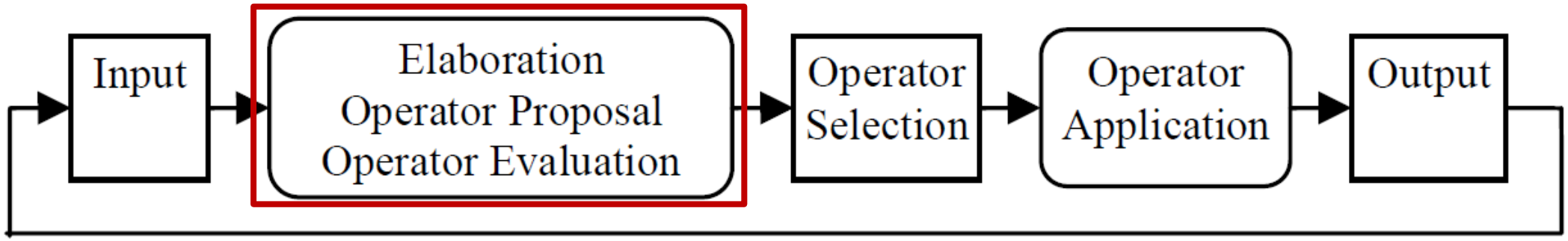


Laird (2008)
Laird, Newell, Rosenbloom (1987)

Soar – Processing Cycle

Produktionsregeln suchen zur Wahrnehmung passende Operatoren/Handlungsoptionen

- Vorschlag von situationsbedingt passenden Operatoren
- Evaluation, welcher Operator in Situation präferiert werden soll (Vgl. ACT-R?)



Laird (2008)
Laird, Newell, Rosenbloom (1987)

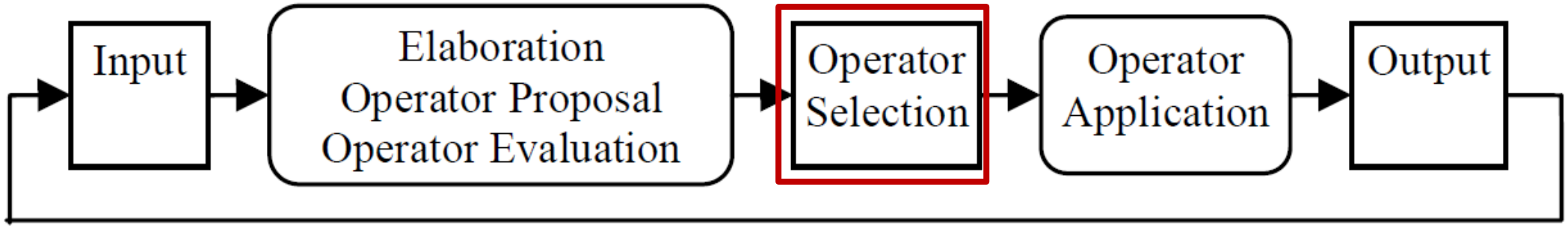
Soar – Processing Cycle

Operatorauswahl

- Entscheidungsprozedur wählt Operator als Kombination aller Präferenzen

➢Mehrere Opertoren gleich geeignet?

➢Impasse („Sackgasse“)



Laird (2008)
Laird, Newell, Rosenbloom (1987)

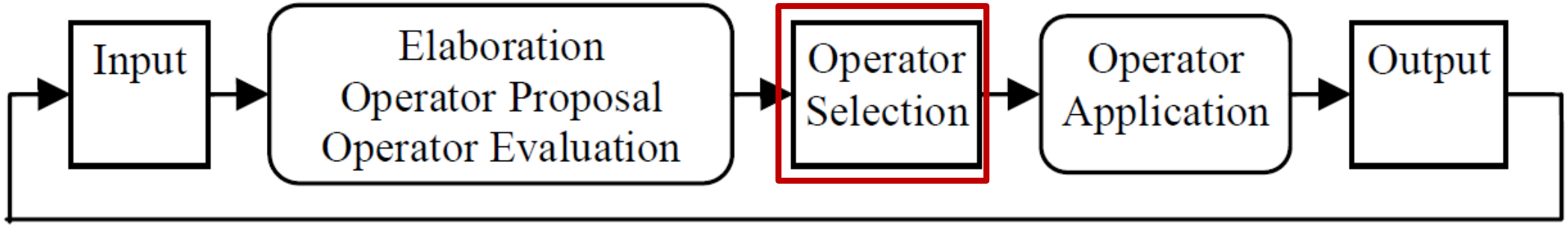
Soar – Processing Cycle

Impasse genereiert Subgoal

- Impasse beseitigen

➢Rekursive Erzeugung immer kleinerer Subgoals, bis eines gelöst werden kann

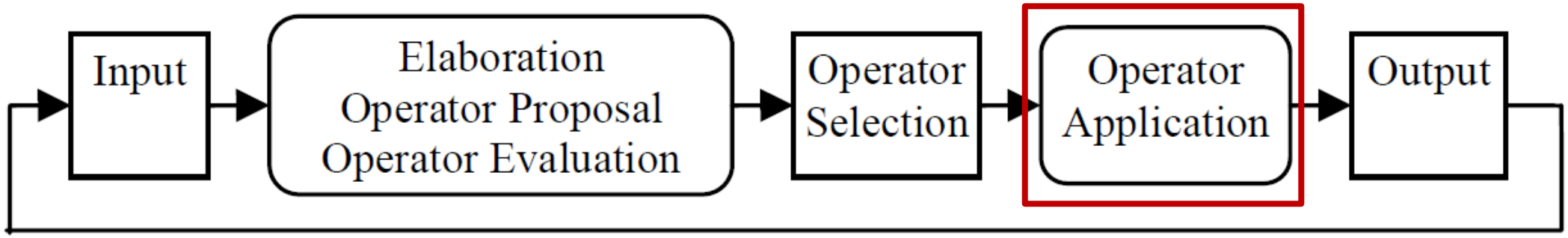
➢Dann Stufenweises „hocharbeiten“ bis Haupt-Goal gelöst wurde



Laird (2008)
Laird, Newell, Rosenbloom (1987)

Soar – Processing Cycle

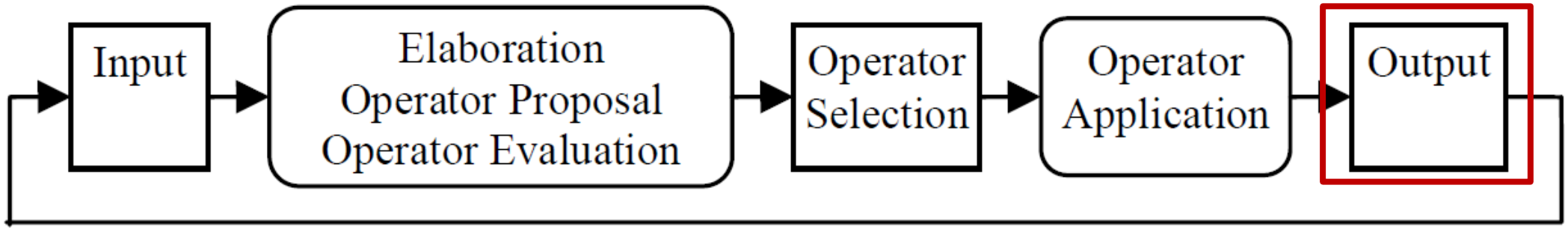
Anwendung der Produktionsregel, die auf aktuelle Situation in WM und den ausgewählten Operator matched



Laird (2008)
Laird, Newell, Rosenbloom (1987)

Soar – Processing Cycle

Output → Weitergabe von Kommandos an motorisches System



Laird (2008)
Laird, Newell, Rosenbloom (1987)

Soar - Fragen

Ablauf

1. Einführung
2. **Soar**
   a. Künstliche Intelligenz
   b. Entwicklung
   c. Struktur und Processing Cycle
   d. **Visual Soar / Soar Java Debugger**
   e. **Generell**
   f. **Working Memory**
   g. **Rules**
   h. **Beispiel**
3. 4CAPS
   a. Einführung
   b. Historie
   c. Operating Principles: Theorie und Implementierung
   d. Zusammenfassung
   e. Framework
   f. Modell-Beispiel
   g. Anwendungsgebiete

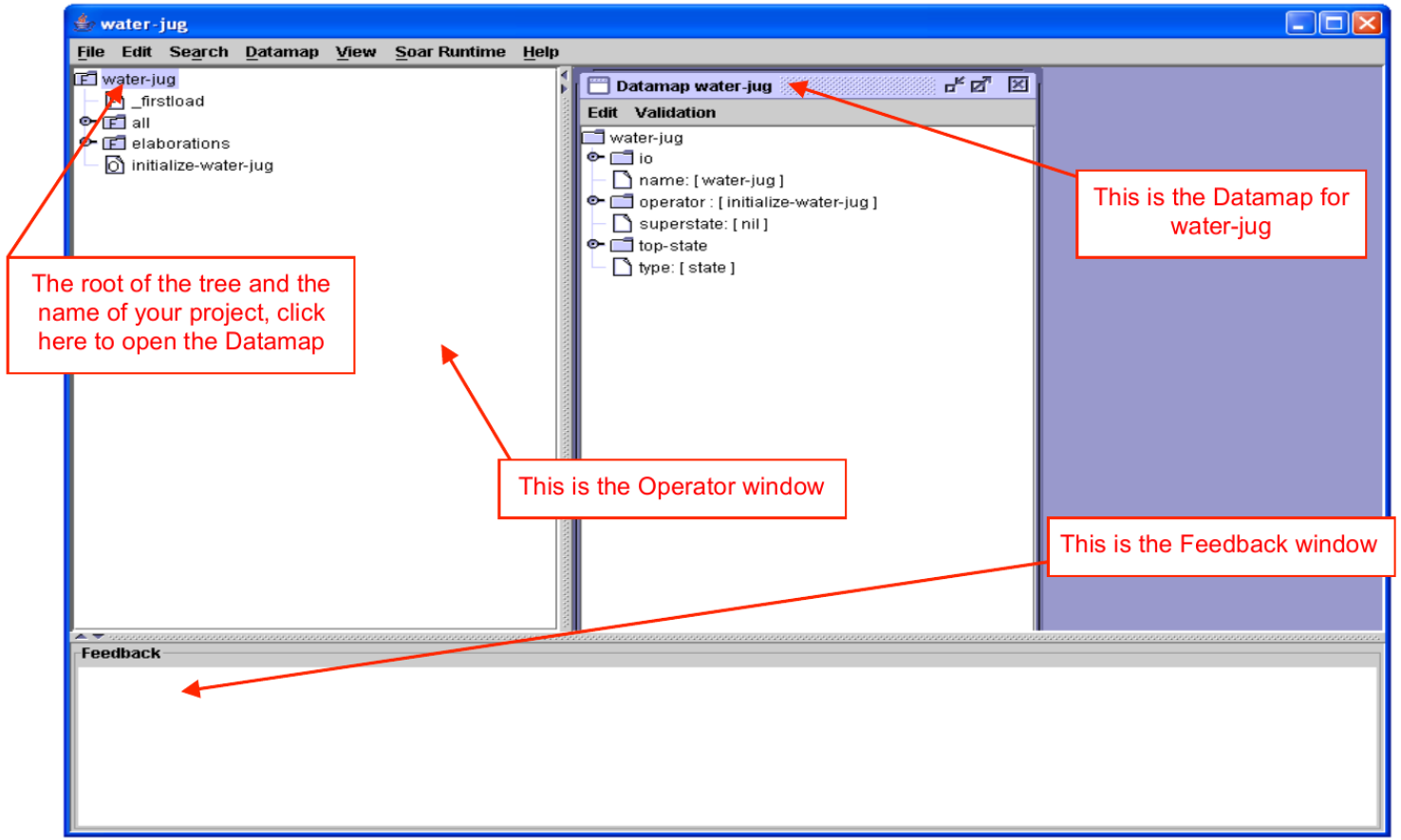
Visual Soar / Soar Java Debugger

Wie in ACT-R

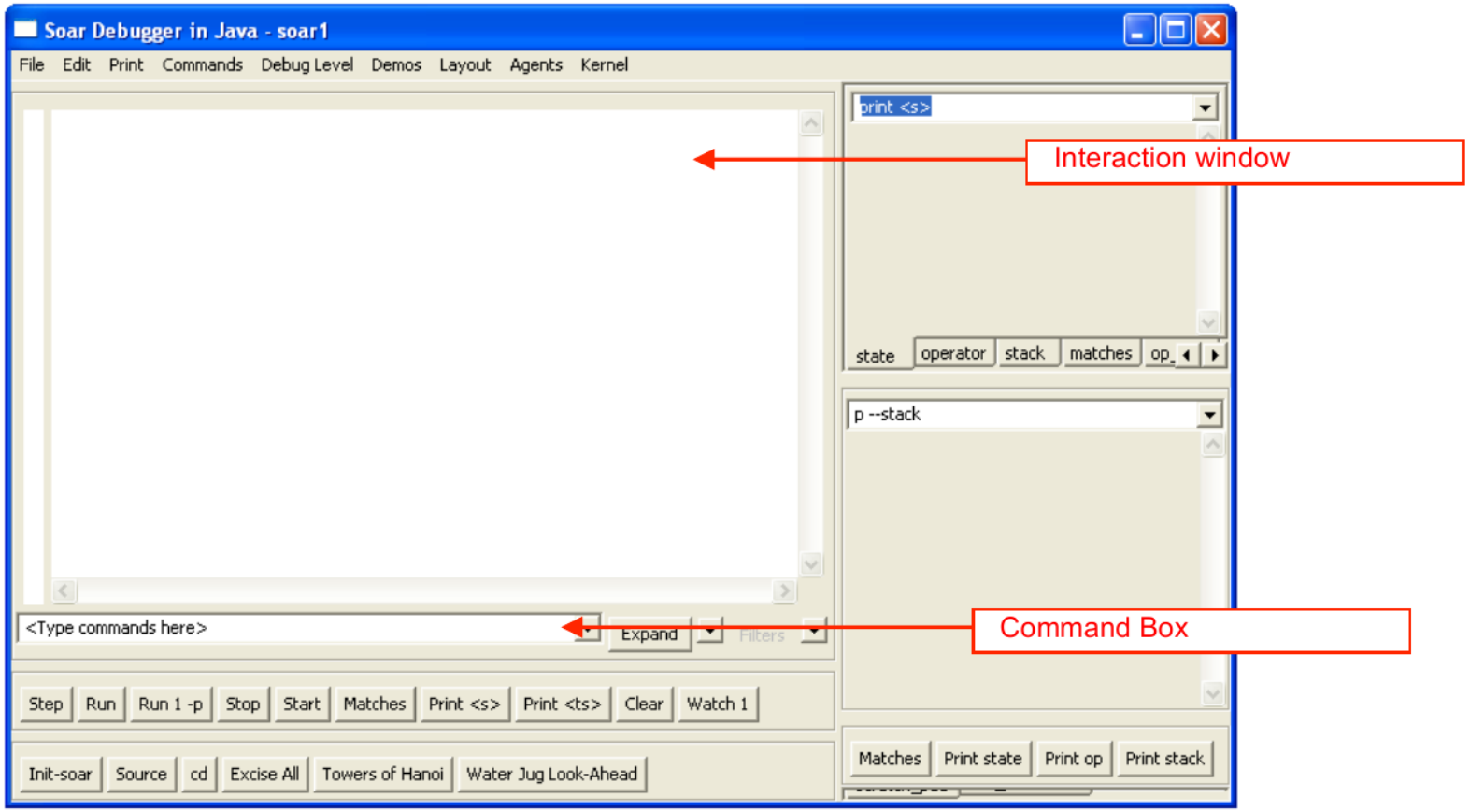
Konvention: Rules in mehrere Dateien aufgeteilt

- Kein Einfluss auf Programm, lediglich für Übersichtlichkeit

Visual Soar


water-jug
File Edit Search Datamap View Soar Runtime Help
water-jug
_firstload
all
elaborations
initialize-water-jug
Datamap water-jug
Edit Validation
water-jug
io
name: [water-jug]
operator : [initialize-water-jug]
superstate: [nil]
top-state
type: [state]
Feedback
The root of the tree and the name of your project, click here to open the Datamap
This is the Operator window
This is the Datamap for water-jug
This is the Feedback window

Soar Debugger



Soar - Generell

Syntax

- # für Kommentare
- || für Strings

Soar - Rules

```
sp {rule*name
   (condition)
   (condition)
   …
   -->
   (action)
   (action)
   …}
```

… means additional conditions can follow.

… means additional actions can follow.

Soar - Rules

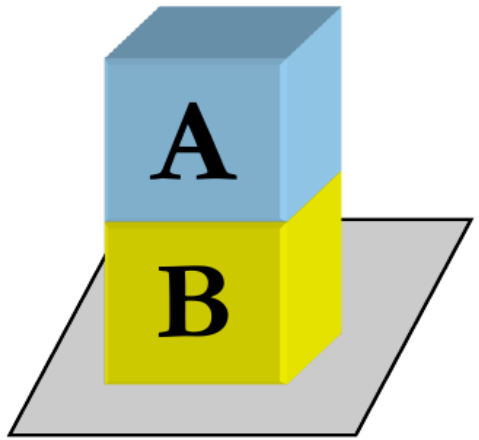
Benennungs Konvention

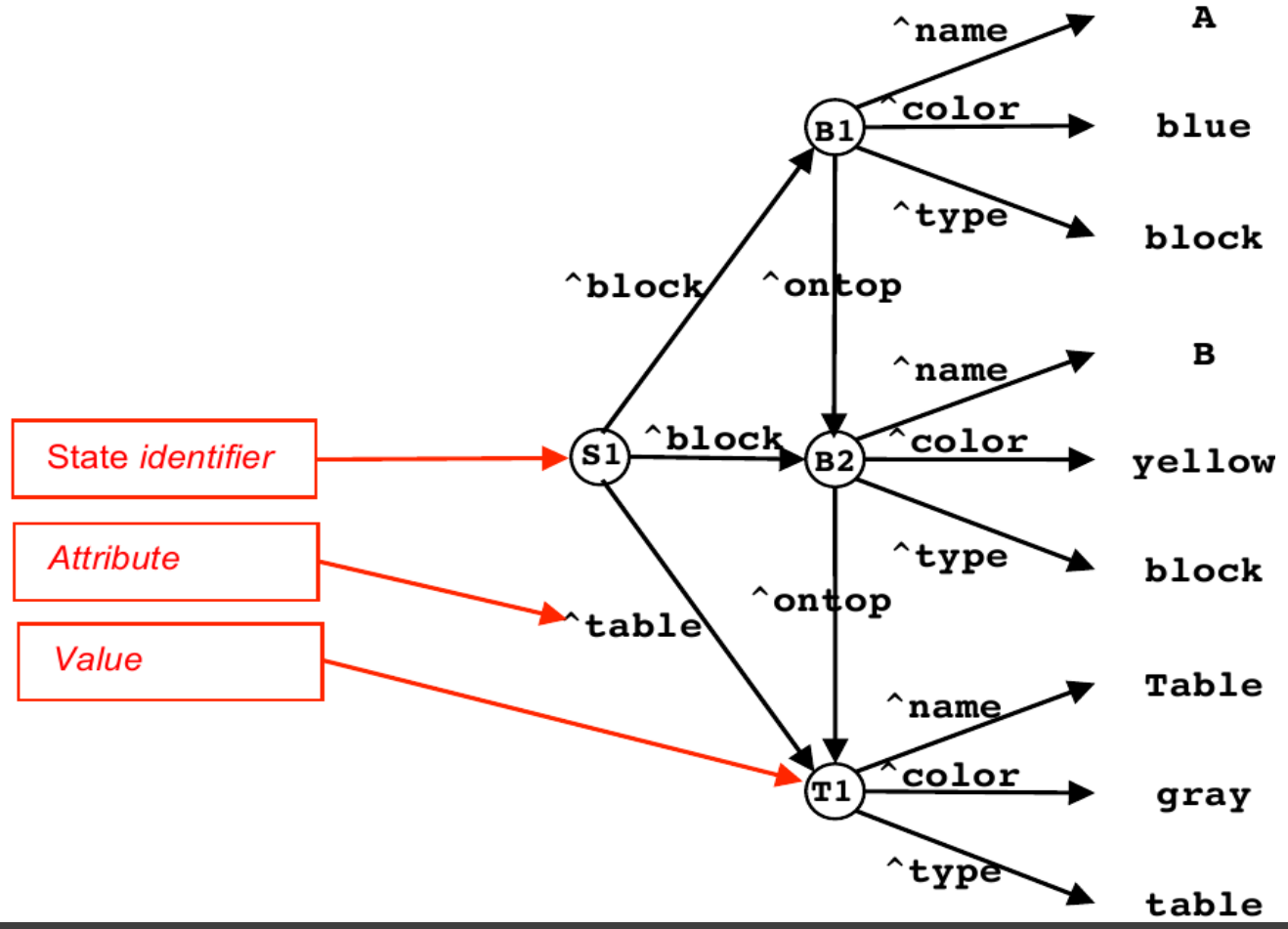
- task*function*name*details

Variablen

- <var>
- Mehrere Referenzen zur gleichen Variable -> gleiches Symbol
- Tests
  - Conditional: -, <, <=, >, >=, <>

Soar - Working Memory





Soar - WM Elements

Element

- Triple (identifier ^attribute value)
- Identifier: non-terminal (hat Attribute)
- Constant: terminal, (keine Attribute, ein Wert)
  - z.B. 42

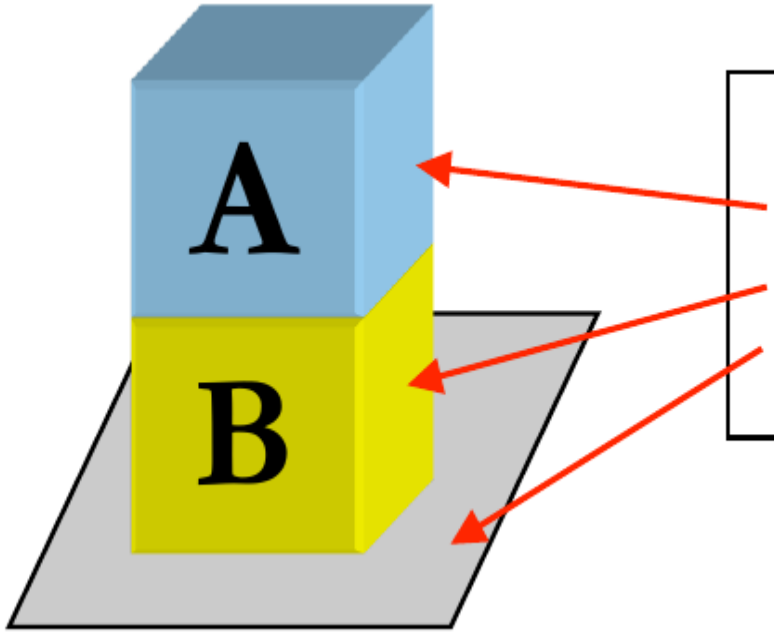
- Verändern eines Elements
  - Altes löschen, neues erstellen
  - Elemente können nicht verändert werden
  - (<var> ^attribute value -): - als 4. Item

Soar - WM Objects

Object

- Mehrere Elemente mit dem gleichen identifier
  - Element: (identifier ^attribute value)
  - Object: (identifier ^attribute value ^attribute value ….)

Soar - WM Objects




```
(s1 ^block b1 ^block b2 ^table t1)
(b1 ^color blue ^name A ^ontop b2 ^type block)
(b2 ^color yellow ^name B ^ontop t1 ^type block)
(t1 ^color gray ^name Table ^type table)
```

Soar - Proposal

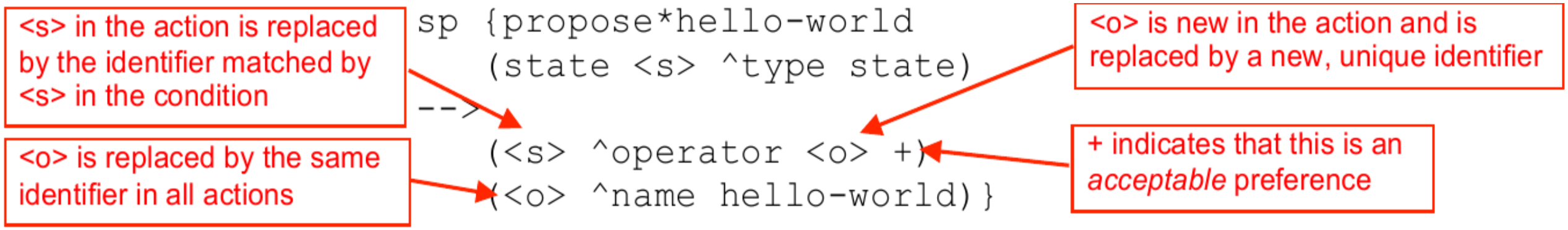
Proposal Rule

- Non Persistent changes (instantiation-support)
- Schlägt Operator als Kandidat vor: Preference erstellen
- (<s> ^operator <o> +)
- (<s> ^operator <o> + =)
- Nur die Decision Procedure kann den Operator auswählen/erstellen
  - (<s> ^operator <o> +) vs (<s> ^operator <o>)

Preference

- + acceptable
- = indifferent

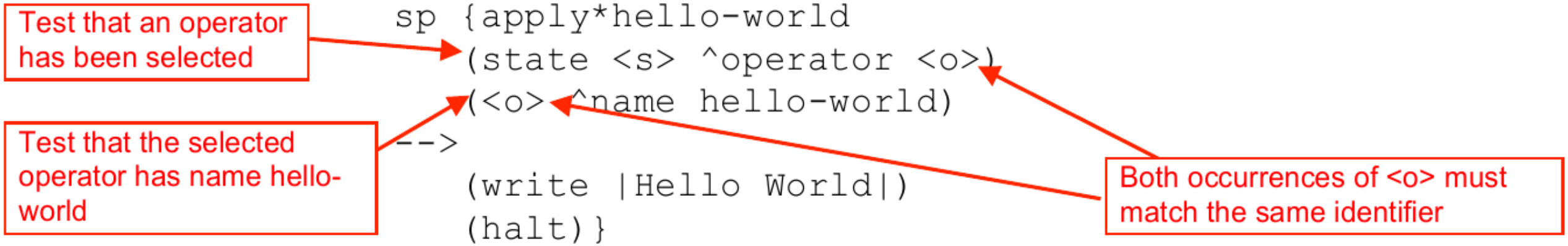
Soar - Proposal



Soar - Application

Application Rule

- Persistent Changes (operator-support)
- Muss den State verändern

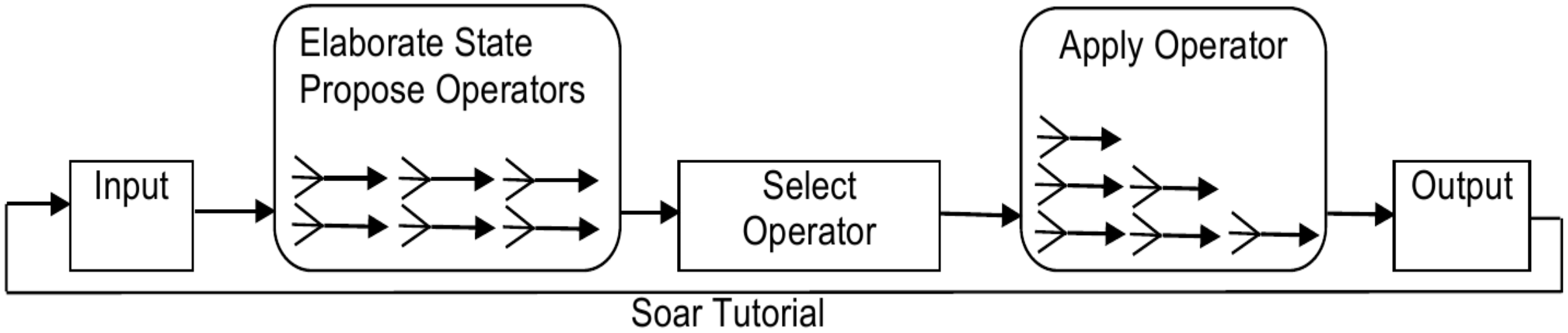


Soar - Elaboration

Elaboration Rule

- Non persistent changes (instantiation-support)
- Berechnung nützlicher Werte, die Rules einfacher zu schreiben machen
  - In Rule Conditions dürfen keine Berechnungen ausgeführt werden

Soar - Cycle


Elaborate State
Propose Operators
Apply Operator
Input
Select
Operator
Output
Soar Tutorial

Soar - Beispiel

Water Jug Task

- 5l Jug
- 3l Jug
- Ziel: 1l in 3l Jug füllen

Soar - Fragen

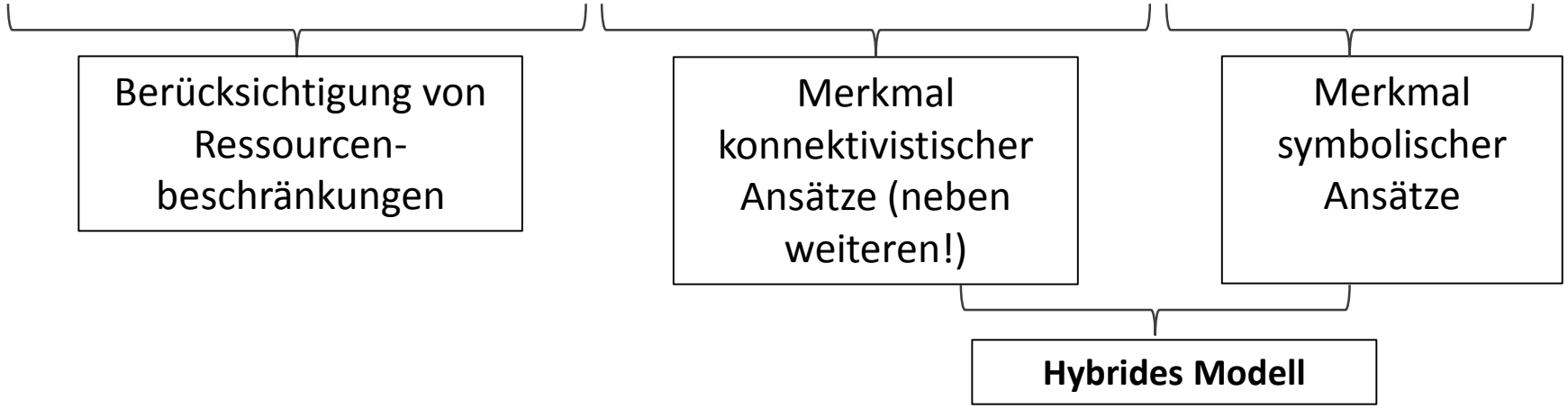
Ablauf

1. Einführung
2. Soar
   a. Künstliche Intelligenz
   b. Entwicklung
   c. Struktur und Processing Cycle
   d. Visual Soar / Soar Java Debugger
   e. Generell
   f. Working Memory
   g. Rules
   h. Beispiel
3. **4CAPS**
   a. **Einführung**
   b. **Historie**
   c. **Operating Principles: Theorie und Implementierung**
   d. **Zusammenfassung**
   e. Framework
   f. Modell-Beispiel
   g. Anwendungsgebiete

Das Wichtigste zuerst

- Fokus: komplexe Kognitionen (Sprachverstehen, spatial rotation, Problemlösen)
- Hintergrund: **Artificial Intelligence** (mathematische Formalisierung der Informationsverarbeitung) und **Cognitive Science** (→ Neuroimaging)
- Kernmerkmale:

Cortical Capacity-Constrained Concurrent Activation-based Production System



4CAPS - Historie und Architectural Family

CAPS (Thibadeau et al.,1982)

- synthetisiert symbolic und activation-based processing

➢ Modellierung von **high-level cognition** (behavioural data)

3CAPS (Just & Carpenter, 1992)

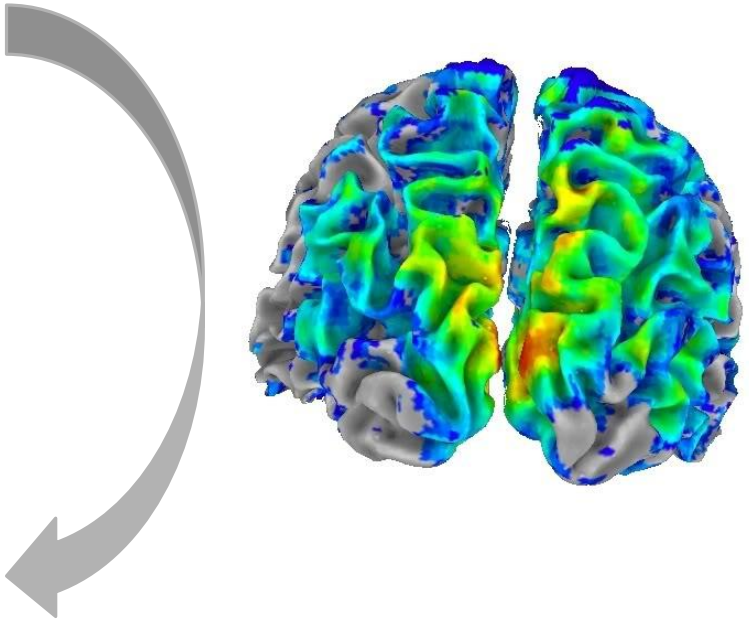
- integriert Kapazitätsbeschränkungen

➢ Untersuchung **individueller Unterschiede**

4CAPS (Just & Varma)

- plausible neuronale Grundlage

➢ **Integrative Neuroarchitektur**



4CAPS - Operating Principles

Theoretische Grundlage: **6 Operating Principles**

(1) Informationsverarbeitung ist netzwerkbasiert

(2) Kortikale Areale haben multiple Spezialisierungen

(3) Ressourcenbeschränkungen begrenzen die Aktivität von kortikalen Arealen

(4) Die Netzwerktopologie verändert sich dynamisch

(5) Die Kommunikationsinfrastruktur ist kapazitätsbeschränkt

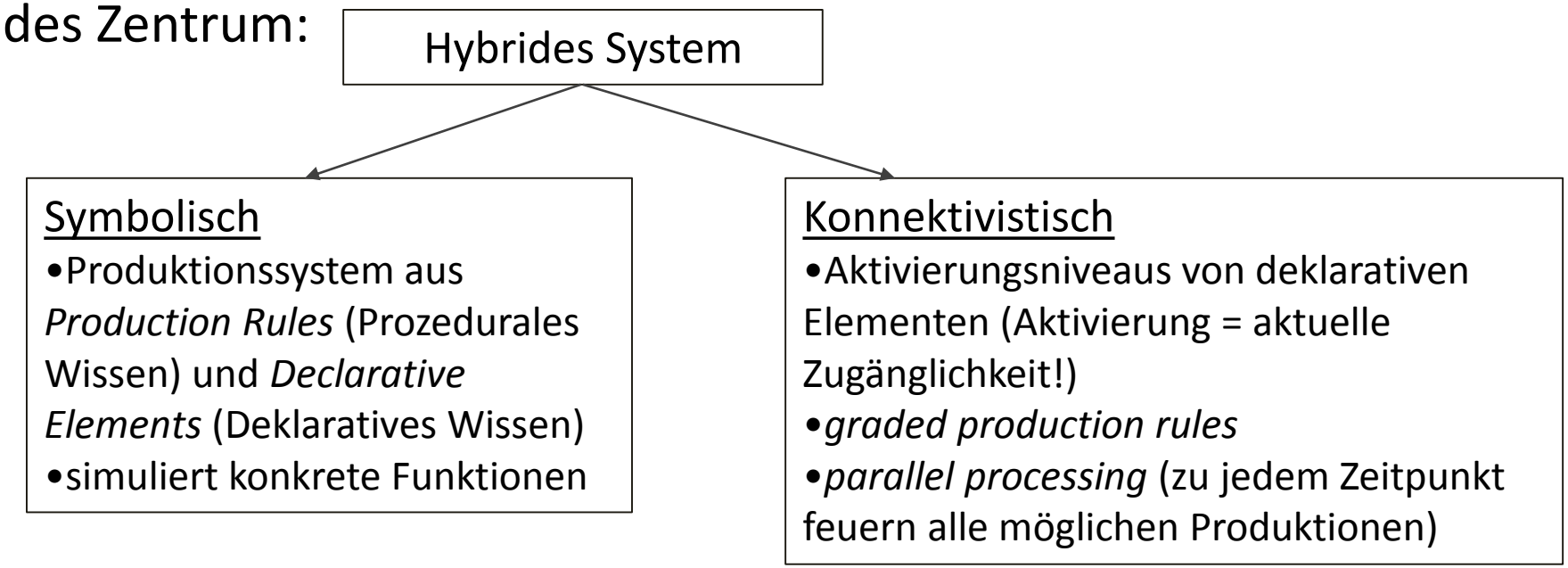
(6) Die Aktivierung kortikaler Areale variiert mit dem Workload

Theorie → Implementierung

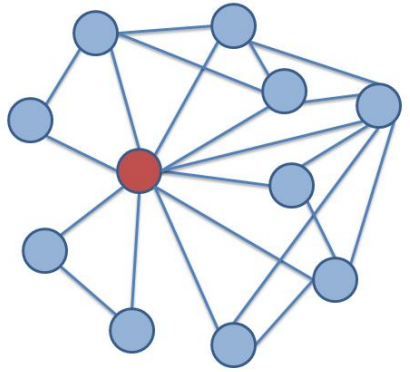
4CAPS - Struktur der Architektur

kortikal: Gehirnareale ⟷ kognitiv: Zentren

jedes Zentrum:



4CAPS - Netzwerkbasierte Informationsverarbeitung

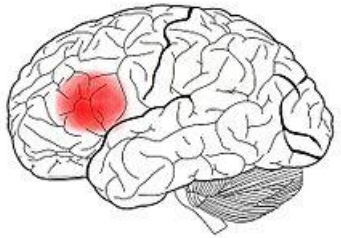


Theorie

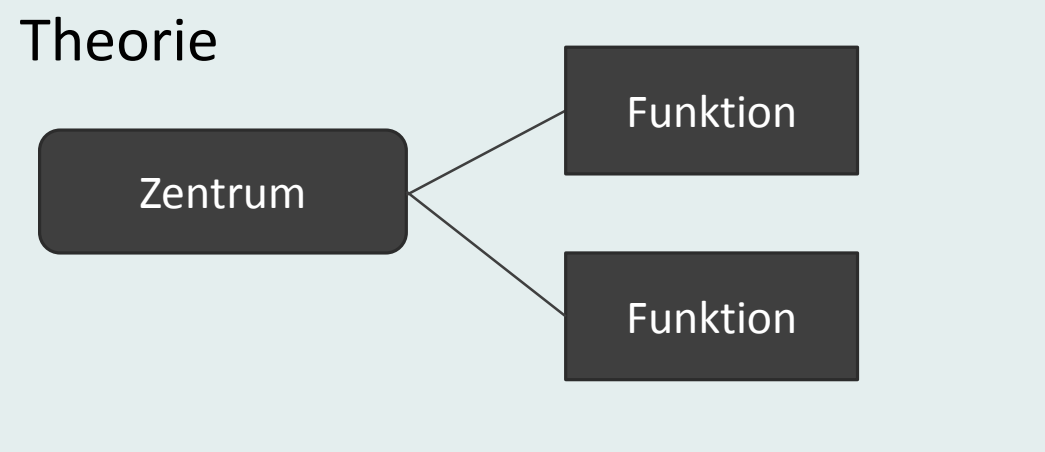
- Zusammenwirken diverser kortikaler Areale bei Aufgabenerledigung → Integration in ein Netzwerk
- funktionale Konnektivität als Koordinationsmaß

Implementierung

- Distinkte kortikale Areale entsprechen Zentren (10 - 20 potentielle Zentren pro Hemisphäre)
- Zusammenarbeit as peers oder hierarchisch

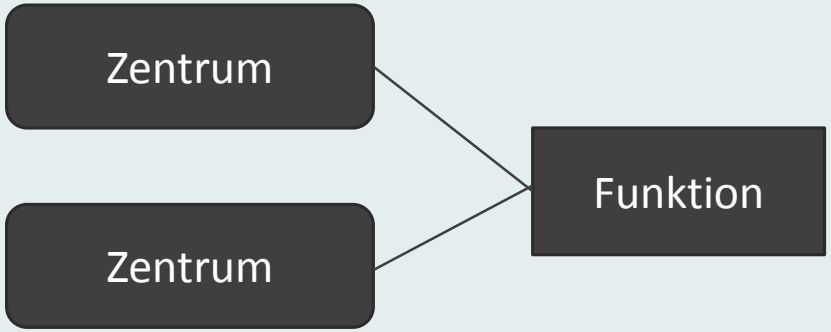


4CAPS - Multiple Spezialisierung von kortikalen Zentren



4CAPS - Multiple Spezialisierung von kortikalen Zentren

Theorie



- ≠ Modularitätsannahme
- ≠ Equipotentiality

Implementierung

- gewisse Redundanz der Zentren jedoch unterschiedlich ausgeprägte **Spezialisierung**
- Zentren können Funktionen, die sich qualitativ (i. e. im processing style) ähneln durchführen
- hohe Spezialisierung ≙ hohe Effizienz (geringer Ressourcenverbrauch)

➢ Auswahl der Zentren für Funktionen nach **relativer Spezialisierung**

- Menge von kognitiver Funktion j, die durch Zentrum i ausgeführt wird: Aij

Aktivitätsbeschränkung von kortikalen Zentren durch begrenzte Ressourcen

Theorie

- Denken = biologische Arbeit
- Ressourcen: Neurotransmitter, Stoffwechselprodukte...

Implementierung

- Ressourcen ≙ Aktivierung
- Mindestaktivierung nötig zum Feuern aber: Ressourcenknappheit

4CAPS - Dynamische Änderung der Netzwerktopologie

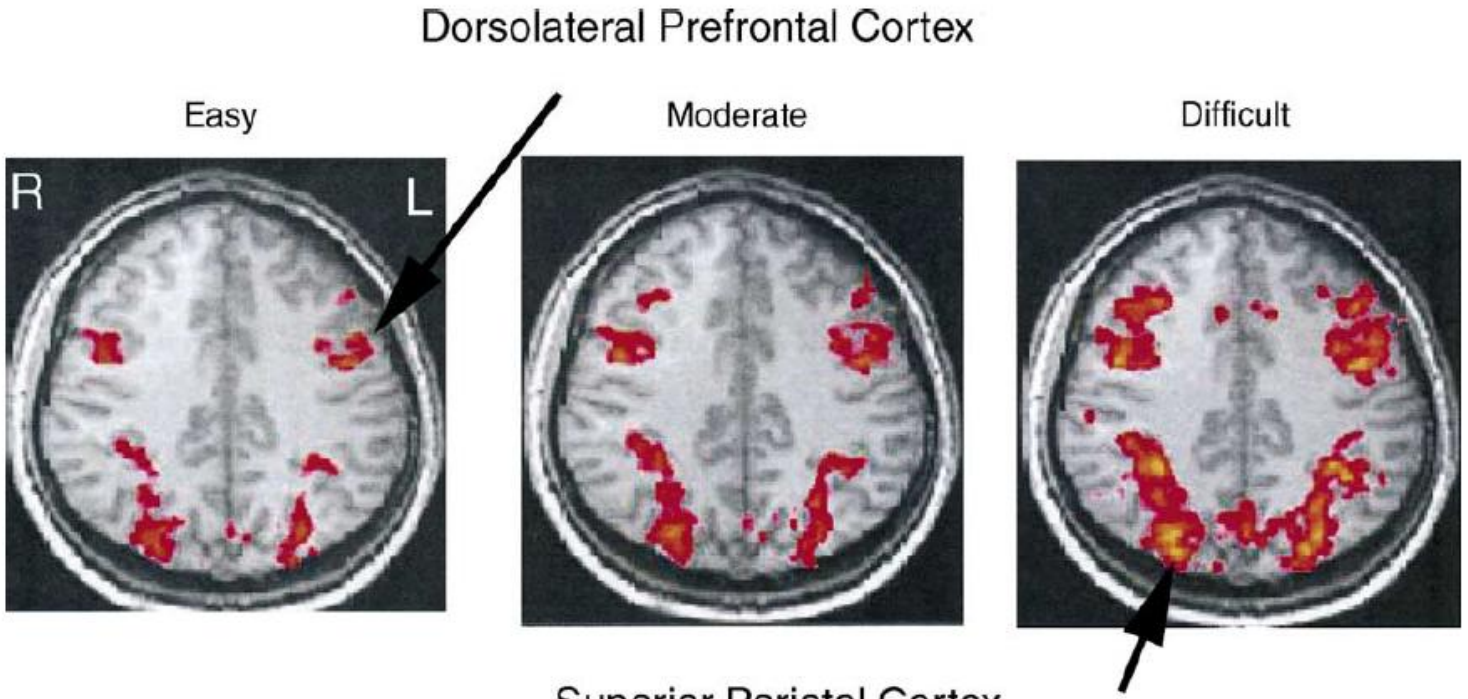
Theorie

- Topologie = Zugehörigkeit und Konnektivität von Zentren im Netzwerk
- Netzwerkkonfiguration **nicht** statisch
- Shortfall (Überbeanspruchung) → Bedarf nach adaptiver Anpassung
- quantitative (Aktivierung von mehr Arealen) und qualitative (Aktivierung von Arealen mit anderem Fokus) Veränderungen

Implementierung

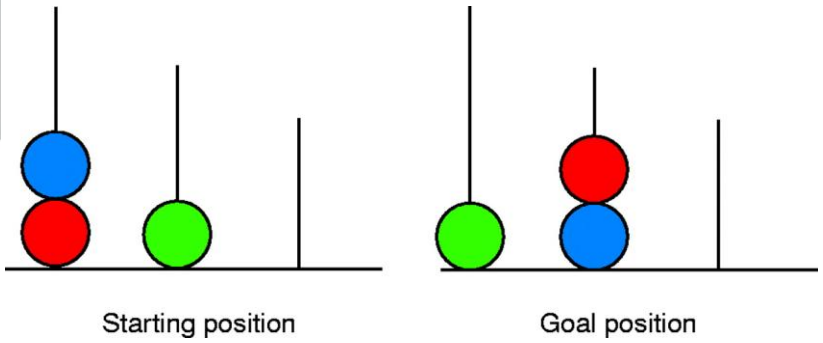
- bei Überbeanspruchung eines Zentrums:
  - **Spillover** (Integration weiterer Zentren)
  - **De-/Reallocation** („Übertragung“ von Aktivierung von nicht mehr benötigten DM-Elementen)

4CAPS - Dynamische Änderung der Netzwerktopologie - Beispiel



(Newman et al., 2003)

- Differentielle Aktivierungsmuster (fMRI) nach Aufgabenschwierigkeit bei Tower of London -Aufgabe



4CAPS - Dynamische Änderung der Netzwerktopologie

Ziel

- Zuordnung der Funktionen zu Zentren, sodass kognitiver Durchlauf maximiert wird
- unter Berücksichtigung aller einschränkenden Faktoren (Ressourcen der Zentren, Aktivierungsbedarf der Funktionen)
- bei Minimierung des Ressourceneinsatzes

Problem

- Zuordnung von Funktionen zu Zentren

Lösung

- Optimale Allokation durch Lineare Optimierung (Simplex-Algorithmus) **zu jeden Zyklus**

Exkurs: Lineare Optimierung (grafisch)

Klassischer Anwendungsfall: Produktionsplanung

- Wie werden **Fertigungsaufgaben** optimal auf **Produktionsstätten** verteilt?

Äquivalent (4CAPS):

- Wie werden **Funktionen** optimal auf **Zentren** verteilt?

Exkurs: Lineare Optimierung (grafisch)

Beispiel:

- eine Funktion mit Aktivierungsbedarf 3
- zwei Zentren je mit Kapazitätsgrenze 6
- Zentrum 1 ist doppelt so effektiv wie Zentrum 2

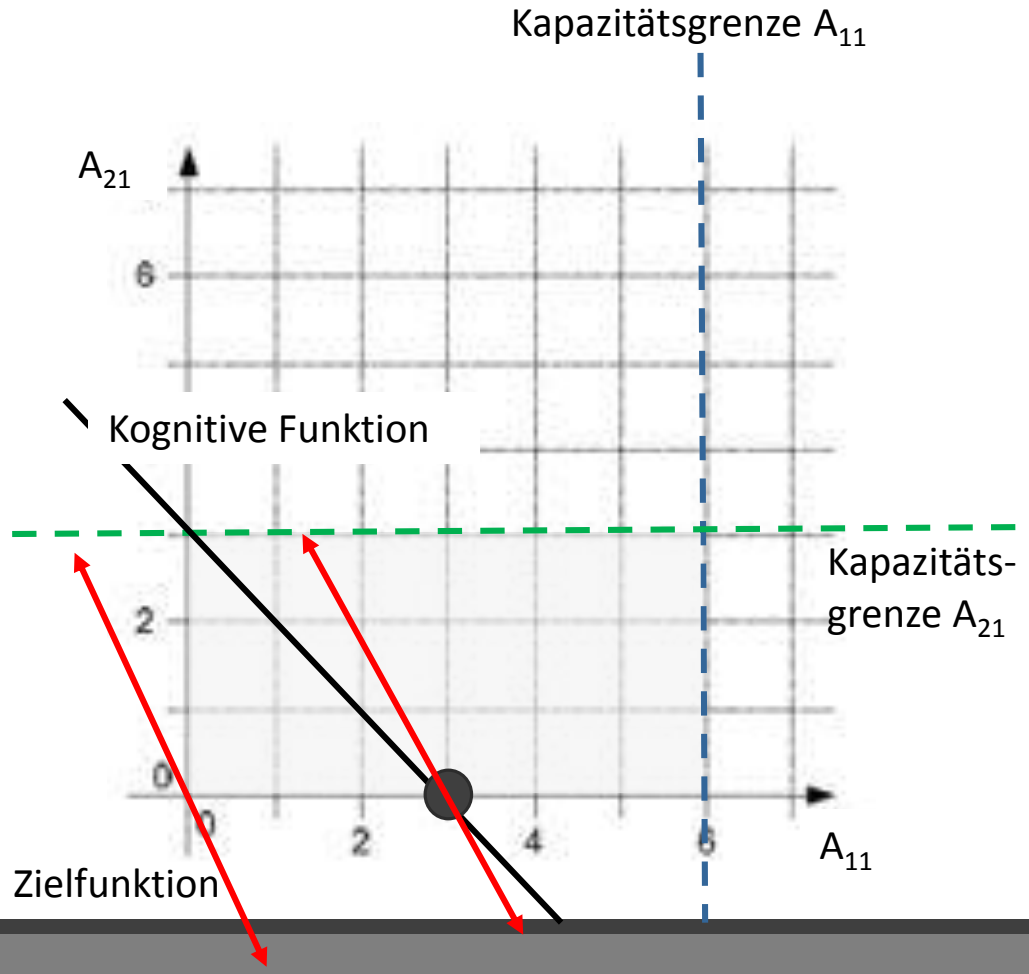
Nebenbedingungen: $A_{11} \leq 6$

$2A_{21} \leq 6$

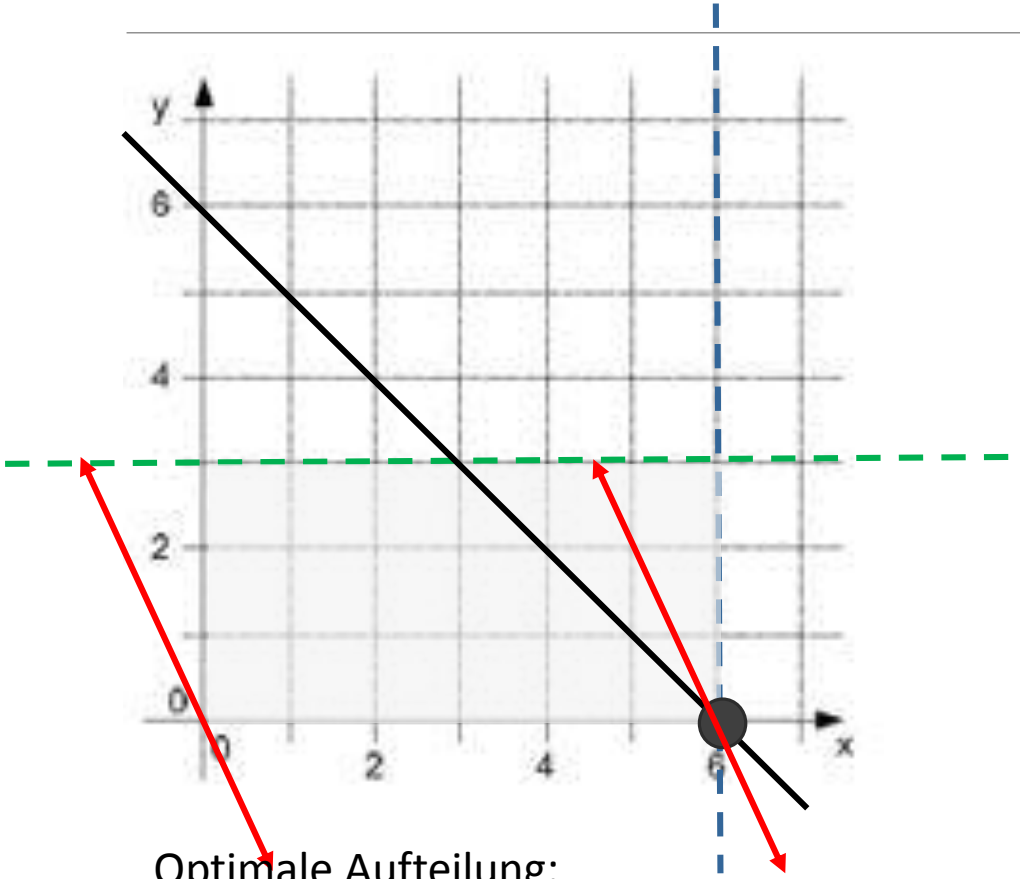
$A_{11} + A_{21} \leq 3$

$A_{11},\ A_{21} \geq 0$

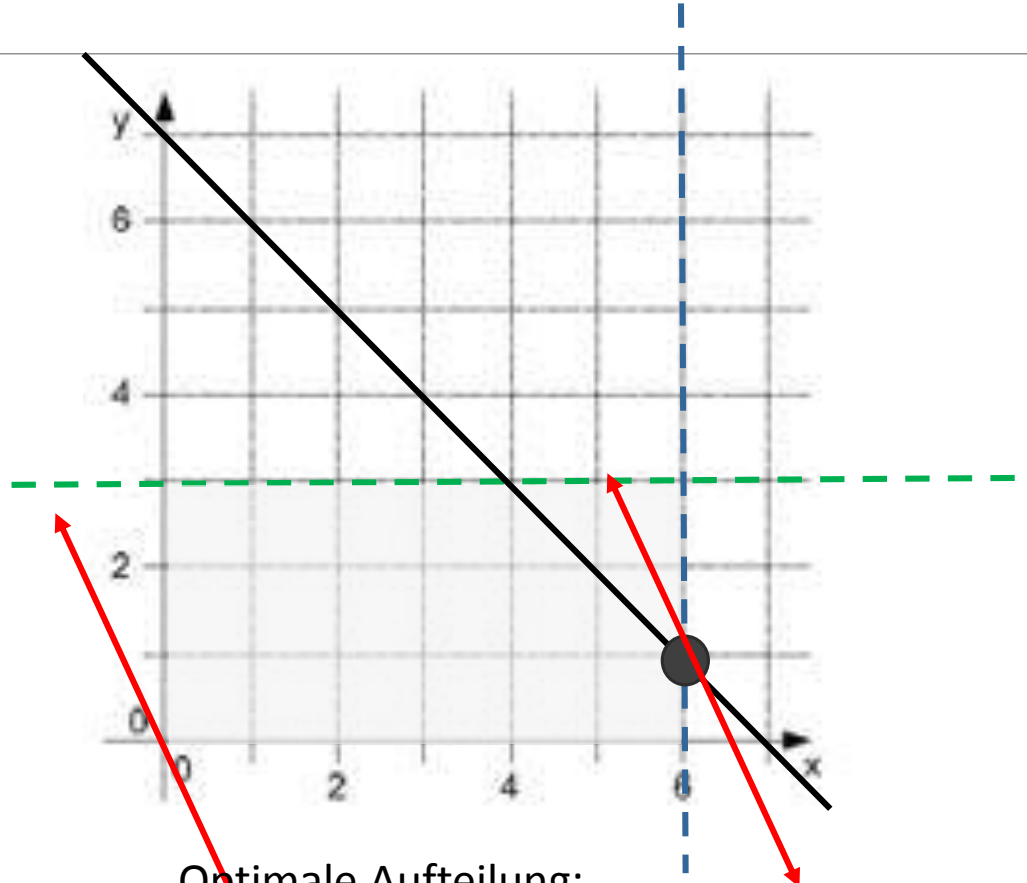
Zielfunktion: $\max\ A_{11} + \frac{1}{2} A_{21}$



Standardsituationen



Optimale Aufteilung:
A11 = 6, A21 = 0



Optimale Aufteilung:
A11 = 6, A21 = 1 → Spillover

4CAPS in Kürze

Neuroarchitektur mit Fokus auf komplexe Kognitionen

Denken als Netzwerkphänomen

Begrenzung von Ressourcen auf kortikaler und kognitiver Ebene

Dynamische, adaptive Veränderung der Netzwerktopologie nach ökonomischen Maßstäben

4CAPS - Fragen

FYI - Operating Principles

0. Thinking is the product of the concurrent activity of multiple brain areas that collaborate in a large-scale cortical network.

1. Each cortical area can perform multiple cognitive functions, and conversely, many cognitive functions can be performed by more than one area.
2. Each cortical area has a limited capacity of computational resources, constraining its activity.
3. The topology of a large-scale cortical network changes dynamically during cognition, adapting itself to the resource limitations of different cortical areas and to the functional demands of the task at hand.
4. The communications infrastructure that supports collaborative processing is also subject to resource constraints, construed here as bandwidth limitations.

5. The activation of a cortical area as measured by imaging techniques such as fMRI and PET varies as a function of its cognitive workload.

Ablauf

1. Einführung
2. Soar
   a. Künstliche Intelligenz
   b. Entwicklung
   c. Struktur und Processing Cycle
   d. Visual Soar / Soar Java Debugger
   e. Generell
   f. Working Memory
   g. Rules
   h. Beispiel

3. 4CAPS

   a. Einführung
   b. Historie
   c. Operating Principles: Theorie und Implementierung
   d. Zusammenfassung
   e. Framework
   f. Modell-Beispiel
   g. Anwendungsgebiete

4CAPS – Framework

working memory elements (wme‘s)

- analog zu den chunks bei ACT-R
- besteht aus **Klassentyp** und **Slots** sowie ggf. einer **Oberklasse**
- es können alle Slots der Oberklasse übernommen werden

`(defwmclass person ()`
`name`
`alter)`
	Klassentyp *person* wird definiert mit
	Slot *name* und
	Slot *alter*

`(defwmclass student (person)` `fach)`	Klassentyp *student* mit Oberklasse *person* wird definiert mit Slot *fach*
`(add(student :fach ‘hf))`	wme vom Klassentyp *student* mit Wert *hf* im Slot *fach*
`(add(person :name ‘eva :alter 23))`	…

4CAPS – Framework

Produktionen

- besteht aus **Produktionsnamen**, **wme-Spezifikation**, **Bedingungsteil (LHS)** und **Aktionsteil (RHS)**
  - **wme-Spezifikation**: Festlegen des wme-Klassentypen, der geprüft wird
  - **Bedingungsteil (LHS)**: legt zu prüfenden Slot und Prüfwert fest
  - **Aktionsteil (RHS)**: legt Operation und auf welchen Slot sie angewendet werden soll fest

```
(p fragen (a student)
(eq (fach a) ‘hf)
-->
(modify goal :done t)
```

Produktion namens *fragen*, die *student* prüft, wird erstellt

LHS: Prüfung, ob der Slot *fach* den Wert *hf* enthält

RHS: verändert Slot *done* im wme *goal* zu Wert *true*

4CAPS – Framework

Module

- Produktionen und wme‘s lassen sich zu Modulen zusammenfassen
- ein Modell kann aus mehreren Modulen bestehen
- Produktionen eines Moduls können mit dem wme‘s eines anderen Moduls interagieren
- es muss ein current-Modul benannt werden; alle folgenden Befehle beziehen sich auf dieses Modul

`(add-mod kennenlernen)`	Modul mit dem Namen *kennenlernen* wird erstellt
`(set-cmod kennenlernen)`	Modul *kennenlernen* wird als current ausgewählt

4CAPS – Framework

Aktivierungswert

- wme‘s kann ein kontinuierlicher Aktivierungswert zugeordnet werden
- der Aktivierungswert kann in der RHS von Produktionen verändert werden:
  - Verbreitung (*spreading*)
  - Hemmung (*inhibition*)
- Aktivierungswerte in wme‘s anderer Module können verändert werden

Aktivierungskapazität

- Modulen kann eine *Aktivierungskapazität* zugewiesen werden
- *Aktivierungskapazität* beschränkt die Aktivierung, die alle wme‘s innerhalb eines Moduls in der Summe haben können

Schwellenwert

- Produktionen können so eingestellt werden, dass sie erst feuern, wenn das nötige wme einen Aktivierungs*schwellenwert* überschreitet.

4CAPS – Framework

Konsequenzen

- **simultanes Feuern** von Produktionen ist möglich (concurrent production system)
  - bei ACT-R feuern die Produktionen seriell (serial production system)
- mehrere Produktionen können also gleichzeitig versuchen ein wme zu verändern
- limitierte Aktivierung durch niedrige Aktivierungskapazität in Modulen kann zu **Beeinträchtigung der Verarbeitung**
- durch das Zusammenwirken von Aktivierungswerten, Aktivierungskapazitäten und Schwellenwerten kann es zu einer Art **Aktivierungsfluss** kommen

4CAPS – Modell-Beispiel

`(del-mods)`	default-Modul wird gelöscht
`(add-mod exec)`	neues Modul mit Bezeichnung exec wird erstellt
`(set-cmod exec)`	Modul exec wird als aktuelles Modul angewählt
`(set-cap 5.0)`	Aktivierungskapazität des aktuellen Moduls wird auf 5 gesetzt
`(add-mod arith)`	neues Modul mit Bezeichnung arith wird erstellt
`(set-cap@ arith 5.0)`	Aktivierungskapazität des Moduls arith wird auf 5 gesetzt
`(trace-mods)`	speichert beim Durchlaufen Informationen zu den Modulen

4CAPS – Modell-Beispiel

16 (defwmclass goal () 17 is 18 done)	neue wme-Klasse names *goal* wird definiert mit den Slots *is* und *done*
19 (defwmclass digit () 20 value 21 op-num)	neue wme-Klasse names *digit* wird definiert mit den Slots *value* und *op-num*
22 (defwmclass answer () 23 value)	neue wme-Klasse names *answer* wird definiert mit den Slots *value*
24 (defwmclass start () 25)	neue wme-Klasse names *start* wird definiert

4CAPS – Modell-Beispiel

29 (p start-problem ((s start))	wme *s* vom Typ *start* muss präsent sein
30 (no ((~g goal)) 31 (eq (is ~g) 'get-first-digit))	es exisistiert kein wme *~g* vom Typ *goal* mit *'get first-digit* im Slot *is*
32 -->	
33 (spew s (goal :is 'get-first-digit) 1.0 34 (in arith 0.5))	verteilt 1.0 mal die Aktivierung von *s* zu *goal* mit *'get-first-digit* im Slot *is* und 0.5 mal die Aktivierung von *s* aus *arith*
35 (del s) 36)	löscht wme-Element *s*

4CAPS – Anwendungsgebiete

wissenschaftliche Anwendung

- Untersuchung von Erkenntnissen aus Studien der kognitiven Neurowissenschaften durch Integration in 4CAPS
- im Vergleich zu ACT-R ist 4CAPS mehr auf „interne“ kognitive Prozesse fokussiert, als auf der Interaktion mit der Umgebung
- Forschung zu Netzwerkeffekten der menschlichen Kognition

industrielle Anwendung

- keine bekannt (Marcel Just)

4CAPS – Fragen

Quellen

http://soar.eecs.umich.edu/

http://ai.eecs.umich.edu/people/laird/

http://ai.eecs.umich.edu/cogarch0/soar/arch/chunk.html

AI - RUG. (1995). Symbolic, subsymbolic, and analogical. Retrieved May 30, 2015, from http://www.ai.rug.nl/~lambert/projects/miami/taxonomy/node99.html

Duch, W., Oentaryo, R. J., & Pasquier, M. (2008). Cognitive Architectures: Where Do We Go from Here? In *Proceedings of the 2008 Conference on Artificial General Intelligence 2008: Proceedings of the First AGI Conference* (pp. 122–136). Amsterdam, The Netherlands, The Netherlands: IOS Press. Retrieved from http://dl.acm.org/citation.cfm?id=1566174.1566187

Gunetti, P., Dodd, T., & Thompson, H. (2013). Simulation of a Soar-Based Autonomous Mission Management System for Unmanned Aircraft. *Journal of Aerospace Information Systems*, *10*(2), 53-70.

Just, M. A., Carpenter, P. A., & Varma, S. (1999). Computational modeling of high-level cognition and brain function. Human Brain Mapping, 8, 128-136.

Just, M. A., Varma, S. (2007). The organization of thinking: What functional brain imaging reveals about the neuroarchitecture of complex cognition. Cognitive & Behavioral Neuroscience, Volume 7, Issue 3, pp 153-191.

John E. Laird. (2014, June 12). Soar Tutorial 1. Retrieved May 30, 2015, from http://soar.eecs.umich.edu/

Laird, J. E. (2008). Extending the Soar cognitive architecture. *Frontiers in Artificial Intelligence and Applications*, *171*, 224.

Laird, J. E., & Congdon, C. B. (2014). Soar Manual 9.4.0. Retrieved May 30, 2015, from http://soar.eecs.umich.edu/

Laird, J. E., Kinkade, K. R., Mohan, S., & Xu, J. Z. (2012). Cognitive robotics using the soar cognitive architecture. *Cognitive Robotics AAAI Technical Report WS-12-06. Accessed July*, *27*, 2012.

Quellen

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial intelligence*, *33*(1), 1-64.

Laird, J. E., Yager, E. S., Hucka, M., & Tuck, C. M. (1991). Robo-Soar: An integration of external interaction, planning, and learning using Soar. *Robotics and Autonomous Systems*, *8*(1), 113-129.

Kaczmarczyk, P. P. SOAR Eine Kognitive Architektur. http://www.dfki.de/~kipp/seminar_ws0607/reports/Soar.pdf

Kelley, T. D. (2003). Symbolic and Sub-Symbolic Representations in Computational Models of Human Cognition What Can be Learned from Biology? *Theory & Psychology*, *13*(6), 847–860. http://doi.org/10.1177/0959354303136005

Kirk, J., & Laird, J. (2014). Interactive task learning for simple games.*Advances in Cognitive Systems*, *3*, 11-28.

Sanner, S. P. (1999). A Quick Introduction to 4CAPS Programming. Pennsylvania.

Thibadeau, R., Just, M. A., & Carpenter, P. A. (1982). A Model of the Time Course and Content of Reading*. Cognitive Science, 6(2), 157-203.

Van Lent, M., Laird, J., Buckman, J., Hartford, J., Houchard, S., Steinkraus, K., & Tedrake, R. (1999, July). Intelligent agents in computer games. In*AAAI/IAAI* (pp. 929-930).

Varma, S. 4CAPS manual. Pennsylvania.

Varma, S., & Just, M. A. (2006). 4CAPS: An Adaptive Architecture for Human Information Processing. In AAAI Spring Symposium: Between a Rock and a Hard Place: Cognitive Science Principles Meet AI-Hard Problems (pp. 91-96).

Weng, J. (2012). Symbolic Models and Emergent Models: A Review. *IEEE Transactions on Autonomous Mental Development*, *4*(1), 29–53. http://doi.org/10.1109/TAMD.2011.2159113

Bilderverzeichnis

http://www.tamaraberg.com/teaching/Spring_14/AI.jpg

http://ai.eecs.umich.edu/people/laird/

https://englishosaca.files.wordpress.com/2012/01/f0013436-artificial_intelligence_and_cybernetics-spl.jpg

http://www.abegglen-psychologie.ch/psychologie_cms/typo3temp/pics/0f3a6f1f55.jpg

http://s120.photobucket.com/user/glahngroup/media/ctwhite.jpg.html

http://journalofdigitalhumanities.org/wp-content/uploads/2012/03/Ego_network.png

http://upload.wikimedia.org/wikipedia/commons/thumb/2/2d/Areabroca.jpg/220px-Areabroca.jpg

http://wikis.zum.de/zum/images/thumb/1/18/AufgabeA28_Koordinatensystem1.jpg/200px-AufgabeA28_Koordinatensystem1.jpg

http://www.familienbande24.de/g/fotos/fragen1.jpg

http://sites.jmu.edu/cogdevlab/files/2012/12/Tower-of-London.jpg

Newman, S. D., Carpenter, P. A., Varma, S., & Just, M. A. (2003). Frontal and parietal participation in problem solving in the Tower of London: fMRI and computational modeling of planning and high-level perception. *Neuropsychologia*, *41*(12), 1668-1682.