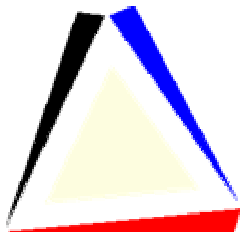


CoPsy Papers

CoPsy Paper No. 10

Tina Schorr & Peter Gerjets

Textaufgaben aus der Kombinatorik:
Eine Analyse von Bearbeitungsstrategien
im Rahmen der ACT-R-Architektur



Universität des Saarlandes
Saarbrücken

SFB 378
Ressourcenadaptive kognitive Prozesse

Fachrichtung Psychologie



CoPsy-Papers dienen der raschen Information über laufende Arbeiten sowie erste Befunde und Ergebnisse aus den Arbeitsgruppen

‚Empirische kognitive Psychologie‘ (Prof. Dr. J. Engelkamp) und

‚Kognitive Modellierung und Methodologie‘ (Prof. Dr. Werner H. Tack)

und aus den zugeordneten SFB-Teilprojekten

‚Verbales und visuelles Arbeitsgedächtnis (VEVIAG)‘ und

‚Wissenserwerb und Wissensnutzung (KnAc)‘

des Sonderforschungsbereichs 378 der Universität des Saarlandes. Sie enthalten interne Arbeitsmaterialien (wie etwa Manuale, Versuchs- und Auswertungsanweisungen), erweiterte Abstracts (etwa zu angemeldeten Vorträgen), erste zusammenfassende Darstellungen neuer Daten und Befunde (noch ohne die für eine endgültige Publikation erforderliche wissenschaftliche Einbettung) sowie Skizzen zur Vorbereitung späterer umfassenderer Darstellungen und Analysen.

CoPsy-Papers wollen Publikationen weder vorwegnehmen noch ersetzen; ihr Ziel ist lediglich eine rasche Vorab-Information.

CoPsy-Paper No. 10

April 2001

© SFB 378 ‚Ressourcenadaptive kognitive Prozesse‘
und Fachrichtung ‚Psychologie‘
der Universität des Saarlandes, Saarbrücken

Tina Schorr
Peter Gerjets

Fachrichtung Psychologie
Universität des Saarlandes
Postfach 15 11 50
D - 66041 Saarbrücken

Inhalt

Teil I: Theoretische Vorüberlegungen	3
1. Einleitung und Fragestellung	3
2. HYPERCOMB: Lernen und Problemlösen mit Kombinatorik-Textaufgaben	6
2.1 Aufgabenkategorien	6
2.2 HYPERCOMB-Experimentalumgebung	7
3. Bearbeitungsstrategien und ihre Wissensvoraussetzungen	12
3.1 Suchbasierte, beispielbasierte und schemabasierte Bearbeitungsstrategien	13
3.2 Varianten beispielbasierter Bearbeitungsstrategien	17
3.3 Bearbeitungsstrategien für Textaufgaben: Schlüsselwort-Strategie und Situationsmodell-Strategie	20
4. Grundannahmen der kognitiven Architektur ACT-R	25
4.1 Deklaratives Gedächtnis	26
4.2 Prozedurales Gedächtnis	26
4.3 ACT-R-Modelle	27
Teil II: Kognitive Modelle verschiedener Bearbeitungsstrategien	28
5. ACT-R-Modell für die Schlüsselwort-Strategie	28
5.1 Grundidee der Schlüsselwort-Strategie	28
5.2 Analyse der Teilzielstruktur der Schlüsselwort-Strategie	29
5.3 Komponenten der Schlüsselwort-Strategie	30
5.4 Darstellung eines <i>run</i> des Modells der Schlüsselwort-Strategie	36
6. ACT-R-Modell für die Situationsmodell-Strategie	38
6.1 Grundidee der Situationsmodell-Strategie	38
6.2 Analyse der Teilzielstruktur der Situationsmodell-Strategie	38
6.3 Komponenten der Situationsmodell-Strategie	39
6.4 Darstellung eines <i>run</i> des Modells der Situationsmodell-Strategie	47
7. Zusammenfassender Vergleich der Modelle und weiterführende Implikationen	49
7.1 Ressourcenanforderungen der Schlüsselwort-Strategie und der Situationsmodell-Strategie	49
7.2 Vergleich von Simulationsdaten und empirischen Daten	51
7.3 Ableitung neuer empirischer Vorhersagen	55
7.4 Implikationen für Lernmodelle und Strategiewahlmodell	56
8. Literatur	59
Teil III: Formaler Anhang	62
A. ACT-R: Definitionen und Gleichungen	62
B. ACT-R-Quellcode für die Schlüsselwort-Strategie	80
C. ACT-R-Quellcode für die Situationsmodell-Strategie	95
D. Kombinatorik-Aufgabenkategorien in HYPERCOMB	113

Teil I: Theoretische Vorüberlegungen

1. Einleitung und Fragestellung

Gegenstand des vorliegenden Arbeitsberichtes sind aufgabenanalytische Überlegungen zur Bearbeitung von Textaufgaben aus dem Bereich der Kombinatorik. Als Aufgabenanalyse bezeichnen wir den Einsatz formaler Techniken zur Beschreibung von Wissen und Strategien, die zur erfolgreichen Bearbeitung einer Aufgabe geeignet sind. Zur Formalisierung unserer Überlegungen greifen wir dabei auf die kognitive Architektur ACT-R von Anderson und Lebiere (1998) zurück, die als umfassende Theorie menschlichen Wissens sowie der Anwendung und des Erwerbs dieses Wissens konzipiert ist. ACT-R erlaubt die Erstellung lauffähiger Simulationsmodelle und ermöglicht damit ebenso detaillierte wie präzise kognitive Aufgabenanalysen.

Konkret beziehen sich die hier beschriebenen Aufgabenanalysen auf kombinatorische Textaufgaben, die in eigenen experimentellen Untersuchungen zum Lernen und Problemlösen als Experimentalmaterial verwendet wurden (Gerjets, Scheiter & Tack, 2000, 2001; Scheiter, Gerjets & Heise, 2000). In diesen Experimenten wurden Textaufgaben aus der Kombinatorik genutzt, um Prozesse des Wissenserwerbs und der Wissensanwendung zu untersuchen. Dabei ging es vor allem um die Abhängigkeit dieser Prozesse von der Verfügbarkeit verschiedener interner und externer Ressourcen. Als interne Ressourcen wurden „Vorwissen“ und „Arbeitsgedächtnis“ betrachtet, als externe Ressourcen „Zeit“ und „nutzbare Information“.

Zentrales Thema der Experimente waren dabei die folgenden beiden Forschungsfragen:

- Wie hängt die Performanz beim Lernen und Problemlösen von der jeweils vorliegenden Konfiguration interner und externer Ressourcen ab?
- Welche Bearbeitungsstrategien beim Lernen und Problemlösen werden in Abhängigkeit von der jeweils vorliegenden Ressourcenkonfiguration eingesetzt und in welchem Ausmaß findet eine ressourcenadaptive Strategiewahl statt?

Die den Untersuchungen zu Grunde liegende Rahmenvorstellung zur Rolle interner und externer Ressourcen bei Prozessen des Wissenserwerbs und der Wissensnutzung ist in Abbildung 1 schematisch dargestellt.

Aus der Verarbeitung von Lernmaterial auf Grundlage einer vorliegenden Zielsetzung resultiert verfügbares Wissen. Menschen verfügen über unterschiedliche Strategien bei einem solchen Lernprozess, deren Wahl sowohl von der gegebenen Zielsetzung als auch von der Konfiguration verfügbarer interner und externer Ressourcen abhängt. Die gewählte Wissenserwerbsstrategie bestimmt wiederum, welche Art von Wissen erworben wird. Aus der späteren Wissensnutzung resultiert eine unter verschiedenen Gesichtspunkten bewertbare Leistung. Auch bei der Aufgabenbearbeitung gibt es ver-

schiedene mögliche Strategien, deren Wahl sowohl vom erworbenen Wissen als auch von verfügbaren weiteren Ressourcen abhängt.

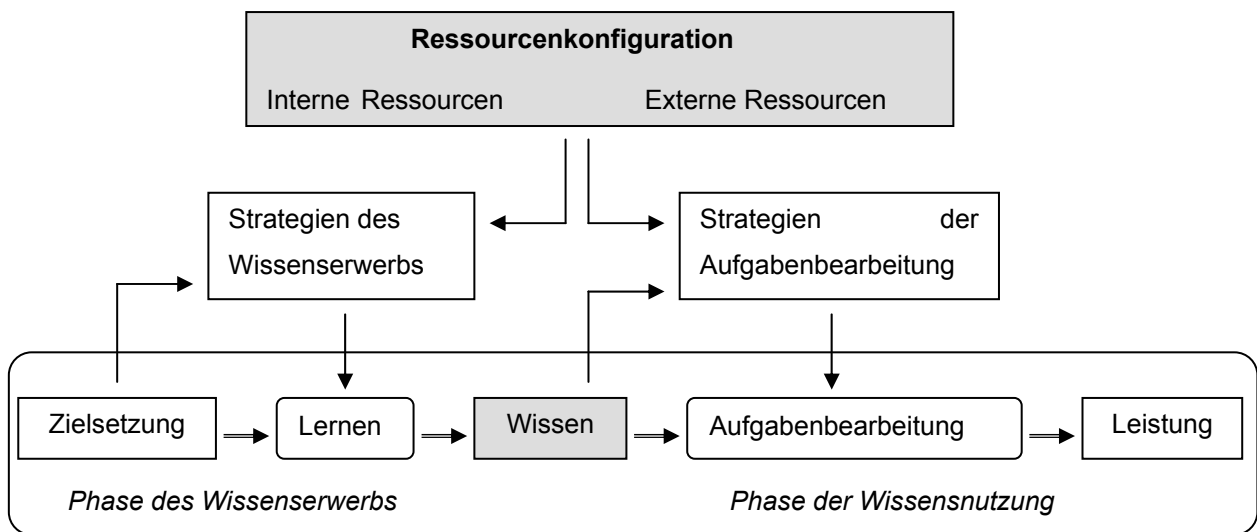


Abbildung 1: Ressourceneffekte bei Wissenserwerb und Wissensnutzung

Zur experimentellen Manipulation interner und externer Ressourcen und zur Erfassung von Performanz- und Strategiemaßen haben wir in unseren Untersuchungen eine selbstentwickelte hypertextbasierte Lern- und Problemlöseumgebung eingesetzt (HYPERCOMB), in der anhand von ausgearbeiteten Beispielaufgaben Wissen über verschiedene Kategorien von Kombinatorikaufgaben erworben werden kann. Im Anschluß an eine Wissenserwerbsphase folgt dabei eine Klausurphase, in der dieses Wissen zur Bearbeitung von Testaufgaben eingesetzt werden soll. Sowohl die Beispielaufgaben zur Illustration von Aufgabenkategorien als auch die Testaufgaben in der Klausurphase sind in Form von Textaufgaben vorgegeben, in denen es um die Berechnung der Wahrscheinlichkeit bestimmter Ereignisse geht.

Ein Überblick über zentrale empirische Befunde, die aus Untersuchungen mit HYPERCOMB resultieren, findet sich in Gerjets, Scheiter und Tack (2000, 2001) sowie Scheiter, Gerjets und Heise (2000). In diesem Arbeitsbericht stehen jedoch nicht die empirischen Details verschiedener HYPERCOMB-Experimente im Vordergrund, sondern vielmehr die vertiefte theoretische Beschäftigung mit verschiedenen Bearbeitungsstrategien für die in HYPERCOMB eingesetzten Textaufgaben aus der Kombinatorik. Dabei ziehen wir die ACT-R-Architektur von Anderson und Lebiere (1998) heran, um langfristig detaillierte Aufschlüsse über folgende Aspekte der Aufgabenbearbeitung zu erhalten:

- Welche Strategien der Bearbeitung von Kombinatorikaufgaben sind möglich und aus welchen Komponenten setzen sich diese Strategien zusammen?

- Welche Anforderungen an interne und externe Ressourcen wie Wissen, nutzbare Information, Arbeitsgedächtnis oder Zeit gehen mit verschiedenen Bearbeitungsstrategien einher?
- Welche Bearbeitungsleistungen beim Lösen von (isomorphen und neuartigen) Testproblemen gehen mit verschiedenen Bearbeitungsstrategien einher?

Die kognitive Aufgabenanalyse auf Basis der ACT-R-Architektur, über die im Folgenden berichtet wird, ist dabei mit einer Reihe von Vorteilen verbunden, die für unsere weiteren Forschungsarbeiten zentral sind:

- ACT-R-Analysen führen zu lauffähigen Simulationsmodellen, in denen Bearbeitungsstrategien und ihre einzelnen Komponenten im Detail expliziert sind. Damit ist sichergestellt, dass formale Spezifikationen von Strategien der Aufgabenbearbeitung vollständig, präzise und logisch konsistent sind.
- Durch die Verwendung einer umfassenden kognitiven Architektur zur Aufgabenanalyse ist gewährleistet, dass nur solche elementare Mechanismen der Informationsverarbeitung postuliert werden, die sich auch in anderen Gegenstandsbereichen als nützlich erwiesen haben.
- Lauffähige Simulationsmodelle erlauben die Ableitung spezifischer Vorhersagen für unterschiedliche experimentelle Bedingungen. Insbesondere lässt sich im Rahmen der ACT-R-Architektur beschreiben, welche Auswirkungen verschiedene Konfigurationen der Ressourcen Zeit, Wissen, externe Information und Arbeitsgedächtnis auf Aspekte der Strategiewahl und Performanz besitzen sollten.
- Eine kognitive Analyse verschiedener Bearbeitungsstrategien auf der Ebene einzelner deklarativer und prozeduraler Wissens Elemente ermöglicht die präzise Spezifikation von Wissensvoraussetzungen bei der Aufgabenbearbeitung. Diese Spezifikationen können zur genaueren Analyse und zur optimalen Gestaltung von Wissenserwerbssituationen genutzt werden.
- ACT-R erlaubt nicht nur die Erstellung von Performanzmodellen, sondern postuliert auch eine Reihe von Mechanismen des Erwerbs von prozeduralem und deklarativem Wissen. Über die Analyse von Bearbeitungsstrategien hinaus können daher in einem zweiten Schritt auch die verschiedenen Lernprozesse analysiert werden, die erforderlich sind, um die mit unterschiedlichen Bearbeitungsstrategien von Aufgaben einhergehenden Wissensvoraussetzungen zu erfüllen.
- Die Darstellung von Bearbeitungsstrategien in ACT-R als bedingte Sequenzen von Teilzielen und Produktionsregeln erlaubt in Kombination mit den in der Architektur vorgesehenen Mechanismen der Auswahl zwischen verschiedenen konkurrierenden Produktionsregeln die Modellierung von Strategiewahlprozessen.

Bevor in Teil II dieser Arbeit konkrete kognitive Modelle von zwei Bearbeitungsstrategien für Textaufgaben aus der Kombinatorik vorgestellt werden, sollen in Abschnitt 2 bis 4 verschiedene theoretische Vorüberlegungen eingeführt werden, die zum Verständnis der dargestellten Modelle nützlich sind.

- In Abschnitt 2 wird zunächst die Inhaltsdomäne genauer erläutert, indem an Beispielen aus der HYPERCOMB-Umgebung illustriert wird, wie Textaufgaben aus der Kombinatorik aufgebaut sind

und wie diese Aufgaben gelöst werden können. Die genaue Erklärung von Kombinatorik-Aufgabenkategorien in HYPERCOMB findet sich in Anhang D.

- In Abschnitt 3 werden unterschiedliche Klassen von Problemlösestrategien und ihre Wissensvoraussetzungen eingeführt. Die *Schlüsselwort-Strategie* und die *Situationsmodell-Strategie*, deren Modellierungen in Teil II im Detail dargestellt werden, sind als Spezialfälle dieser allgemeinen Bearbeitungsstrategien aufzufassen.
- In Abschnitt 4 werden die grundlegenden Annahmen der kognitiven Architektur ACT-R eingeführt. Diese Annahmen werden in Anhang A vertieft und formal präzisiert dargestellt. Die Anhänge B und C enthalten die ACT-R Quellcodes der in Teil II im Detail vorgestellten kognitiven Modelle.

2. HYPERCOMB: Lernen und Problemlösen mit Kombinatorik-Textaufgaben

Das mathematische Gebiet der Kombinatorik dient der Lösung von Abzählproblemen. Insbesondere haben Problemstellungen aus dem Bereich der Kombinatorik die Berechnung der Anzahl möglicher Anordnungen und Auswahlen von Elementen zum Gegenstand. Diese Berechnungen werden z.B. benötigt, um in Zufallsexperimenten die Wahrscheinlichkeiten bestimmter Anordnungen und Auswahlen zu bestimmen. Wahrscheinlichkeiten werden dabei als Quotient aus günstigen Fällen und möglichen Fällen dargestellt.

2.1 Aufgabenkategorien

Elementare Kombinatorik-Probleme lassen sich in sechs Aufgabenkategorien einteilen. Man unterscheidet *Permutationsprobleme*, *Variationsprobleme* und *Kombinationsprobleme*, die jeweils mit und ohne Wiederholungen auftreten können. Die Frage der Wiederholung steht in Abhängigkeit davon, ob die betrachteten Elemente nur einmal oder aber mehrmals in einer Anordnung oder Auswahl auftreten können. Während bei Permutationen und Variationen die Reihenfolge der angeordneten oder ausgewählten Elemente wesentlich ist, ist die Reihenfolge bei Kombinationen unwesentlich. Permutationsprobleme betreffen die vollständige Anordnung von Elementen, während Variationen und Kombinationen sich auf die Anzahl möglicher Auswahlen von Elementen beziehen. Die zur Klassifikation dieser Kombinatorik-Probleme wesentlichen strukturellen Aufgabenmerkmale sind damit *Anordnung/Auswahl*, *mit/ohne Wiederholung* und *Reihenfolge relevant/irrelevant*. Eine übliche Vorgehensweise zur Lösung der resultierenden sechs Aufgabenkategorien besteht darin, jeweils eine bestimmte Formel anzuwenden.

- *Permutation ohne Wiederholung (Aufgabenkategorie 1):* $P_{ow} = n!$
Es gibt $n!$ (n Fakultät) Möglichkeiten, n Objekte in eine Anordnung zu bringen. Diese Formel benötigt man z.B. zur Lösung folgenden Problems: Wie groß ist die Wahrscheinlichkeit, im Untertest

„Bilderlegen“ des Hamburg-Wechsler-Intelligenztests eine Gruppe von sechs Bildern zufällig in die richtige Reihenfolge zu bringen?

- *Permutation mit Wiederholung (Aufgabenkategorie 2):* $P_{mw} = n! / (k_1! k_2! \dots k_n!)$
Wenn eine Menge von n Objekten aus k Gruppen mit jeweils k_1, k_2, \dots, k_n nicht-unterscheidbaren Elementen besteht, gibt es $n! / (k_1! k_2! \dots k_n!)$ verschiedene Möglichkeiten, die n Objekte in unterscheidbarer Reihenfolge anzuordnen. Diese Formel benötigt man z.B. zur Lösung folgenden Problems: Wie groß ist die Wahrscheinlichkeit, fünf rote und fünf schwarze Kugeln so aus einer Urne zu ziehen, dass zuerst alle roten und dann alle schwarzen Kugeln gezogen werden?
- *Variation ohne Wiederholung (Aufgabenkategorie 3):* $V_{ow} = n! / (n-k)!$
Es gibt $n! / (n-k)!$ Möglichkeiten, aus n Objekten k Objekte in einer bestimmten Reihenfolge auszuwählen, wenn jedes Objekt nur einmal ausgewählt werden kann. Beispiel: Wie groß ist die Wahrscheinlichkeit, aus einem Teich mit fünf verschiedenen Fischen zuerst eine Forelle und dann einen Karpfen zu angeln?
- *Variation mit Wiederholung (Aufgabenkategorie 4):* $V_{mw} = n^k$
Es gibt n^k Möglichkeiten, aus n Objekten k Objekte in einer bestimmten Reihenfolge auszuwählen, wenn die Objekte jeweils mehrfach ausgewählt werden können. Beispiel: Wie groß ist die Wahrscheinlichkeit, mit einem Würfel hintereinander die Zahlen 3, 6 und noch einmal die 3 zu würfeln?
- *Kombination ohne Wiederholung (Aufgabenkategorie 5):* $K_{ow} = n! / ((n-k)! k!)$
Es gibt $n! / ((n-k)! k!)$ Möglichkeiten, aus n Objekten k Objekte auszuwählen, wenn die Reihenfolge der Objekte unwesentlich ist und wenn gewählte Objekte jeweils nicht noch einmal ausgewählt werden können. Beispiel: Wie groß ist die Wahrscheinlichkeit, beim Lottospiel sechs richtige Zahlen zu tippen?
- *Kombination mit Wiederholung (Aufgabenkategorie 6):* $K_{mw} = (n+k-1)! / ((n-1)! k!)$
Es gibt $(n+k-1)! / ((n-1)! k!)$ Möglichkeiten, aus n Objekten k Objekte auszuwählen, wenn die Reihenfolge der Objekte unwesentlich ist und wenn die Objekte jeweils mehrfach ausgewählt werden können. Beispiel: Wie groß ist die Wahrscheinlichkeit, aus einem Kartenspiel hintereinander vier Könige zu ziehen, wenn die gezogenen Karten jeweils zurückgelegt werden?

2.2 HYPERCOMB-Experimentalumgebung

Zur Vermittlung des für die Bearbeitung von Kombinatorik-Textaufgaben erforderlichen Wissens wurde die webbasierte Hypertext-Umgebungen HYPERCOMB eingesetzt. Diese Umgebung soll im folgenden anhand von Screenshots der Benutzeroberfläche kurz genauer vorgestellt werden.

Zu Beginn eines HYPERCOMB-Experimentes erhalten alle Versuchspersonen eine etwa 10-seitige Einführung in die Kombinatorik, d.h. es wird ihnen erläutert, worum es in der Kombinatorik geht, was z.B. Auswahlen und Anordnungen von Elementen sind, wie Wahrscheinlichkeiten berechnet werden usw. (siehe Abbildung 2).

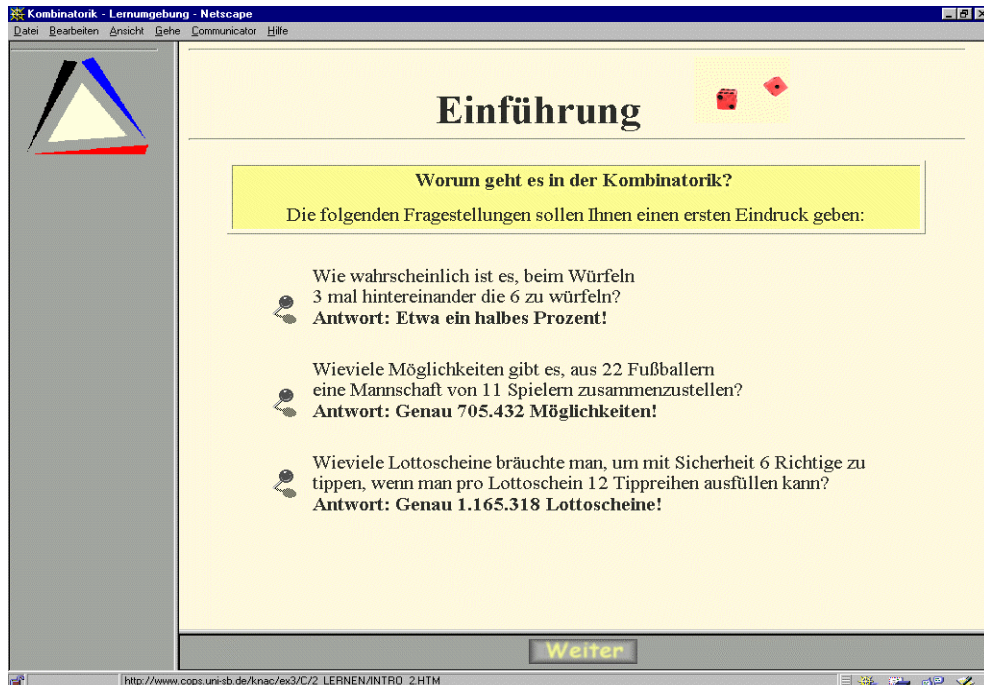


Abbildung 2: Einführung in die Kombinatorik (screenshot)

Im Anschluss an diese Einführung präsentieren wir den Versuchspersonen eine Lernumgebung, in der sie die sechs Aufgabenkategorien aus der Kombinatorik erlernen sollen, und zwar die Formeln für Permutation, Kombination und Variation, jeweils mit und ohne Wiederholung. Diese Kategorien können in der Navigationsleiste am linken Bildschirmrand aufgerufen werden und werden dann zunächst allgemein erklärt, indem die entsprechende Aufgabenkategorie abstrakt definiert und die benötigte Formel vorgestellt wird. Als Zusatzinformation werden je nach Versuchsbedingung verschiedene Beispielaufgaben mit Musterlösungen angeboten. Diese Beispielaufgaben können z.B. einfache Urnenbeispiele, einfache Alltagsbeispiele oder komplexere Alltagsbeispiele sein (siehe Abbildung 3).

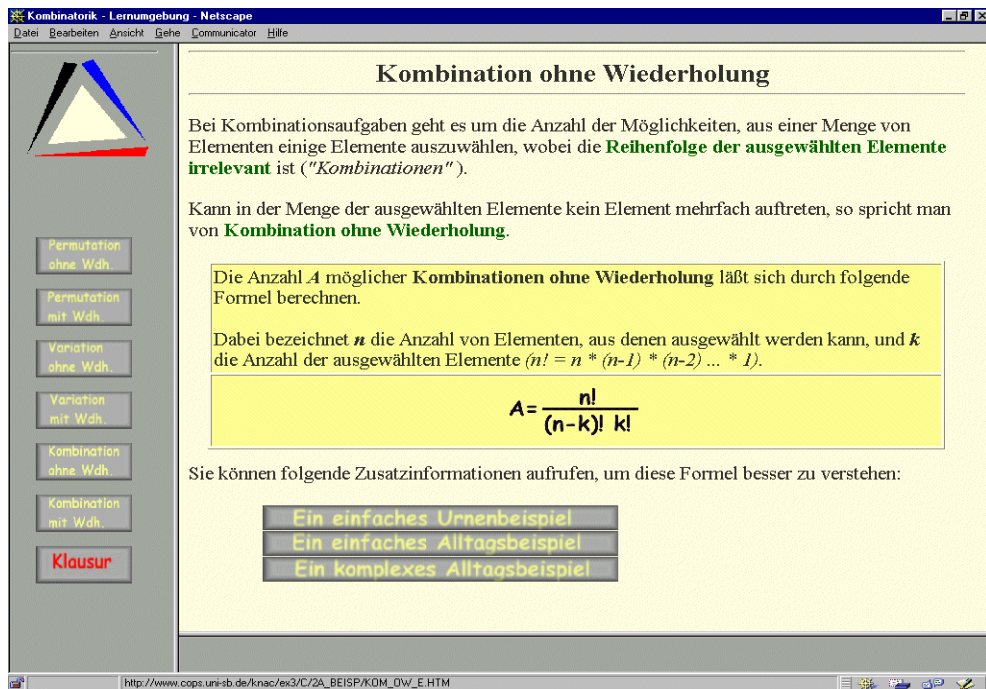


Abbildung 3: Abstrakte Definition einer Aufgabenkategorie (screenshot)

Diese ausgearbeiteten Beispielaufgaben werden dann auf den entsprechenden Bildschirmseiten Schritt für Schritt gelöst und erklärt (siehe Abbildung 4). Wenn die Versuchsperson der Meinung ist, die verschiedenen Aufgabenkategorien und ihre Lösung verstanden zu haben, kann sie die Lernphase beenden und in eine Hypertext-Klausur wechseln, um selbstständig Testaufgaben zu bearbeiten.

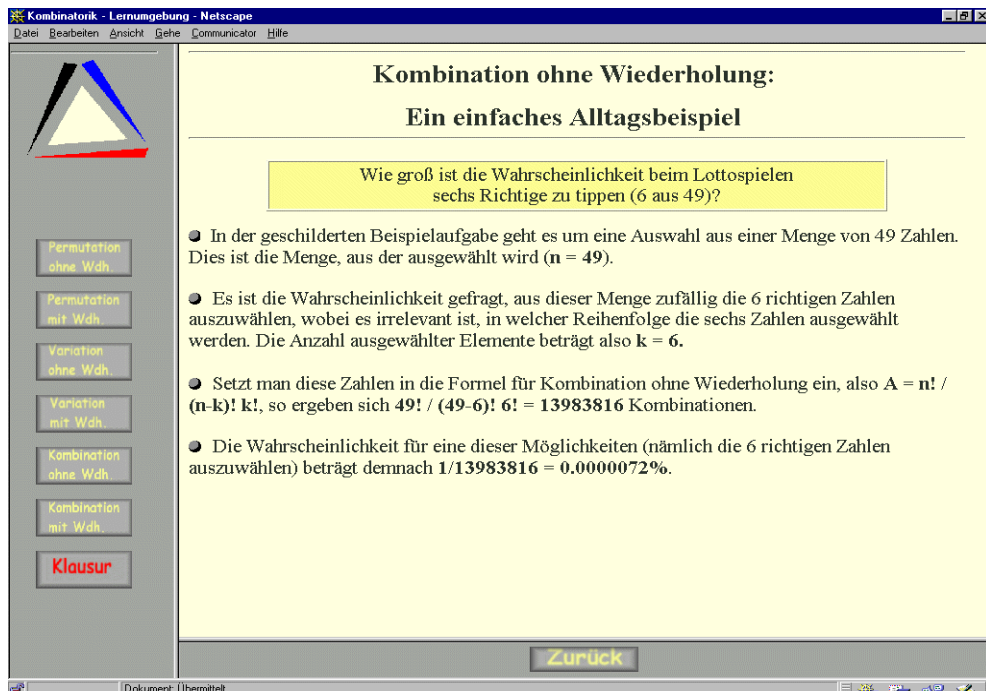


Abbildung 4: Ausgearbeitete Beispielaufgabe (screenshot)

Die Klausur enthält drei Textaufgaben, die jeweils gelöst werden müssen, indem die benötigte Formel und die Ausprägungen der auftretenden Variablen angegeben werden (siehe Abbildung 5).

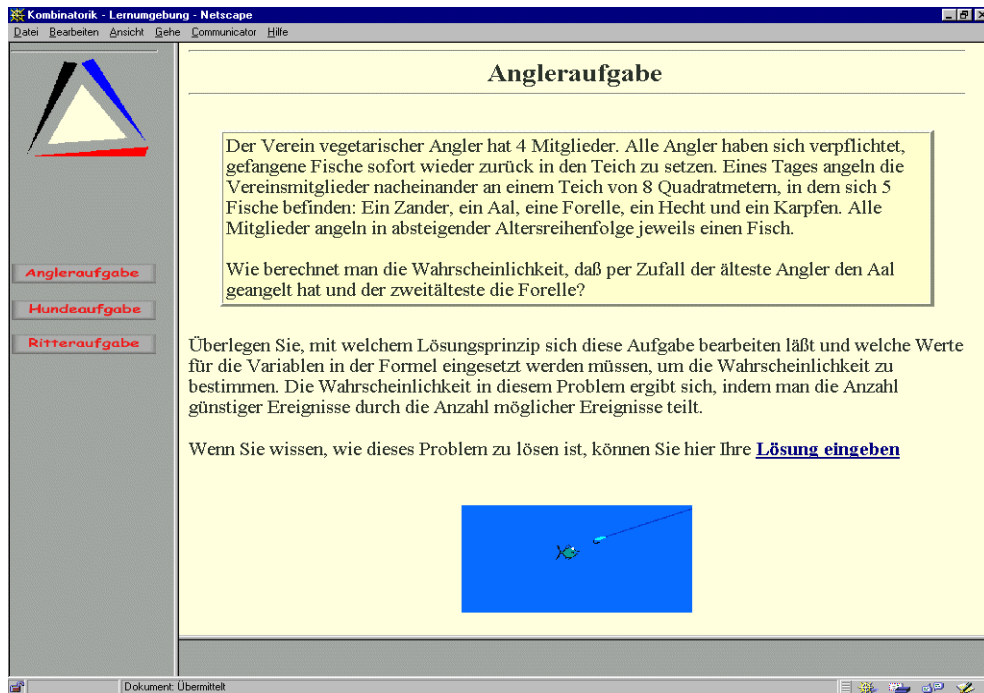


Abbildung 5: Klausuraufgabe (screenshot)

Diese Angaben müssen in ein Lösungsformular eingegeben werden, indem eine Formel und zwei Variablenausprägungen angeklickt werden (siehe Abbildung 6). Danach kann die Lösung abgeschickt und die nächste Klausuraufgabe bearbeitet werden.

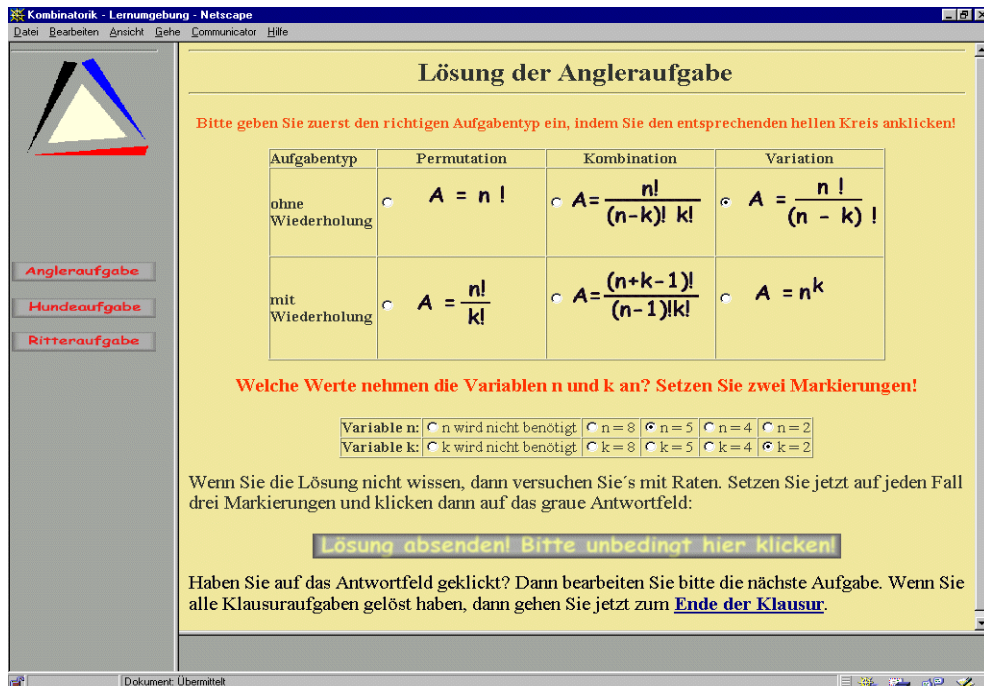


Abbildung 6: Lösungsformular (screenshot)

Für die kognitiven Aufgabenanalysen, die in diesem Arbeitsbericht vorgestellt werden, beziehen wir uns auf eine spezifische Experimentalbedingung, in der die Versuchspersonen für jede der sechs Aufgabenkategorien ein einfaches Urnenbeispiel als Instruktionsmaterial abrufen können (siehe Anhang D). In dieser Experimentalbedingung wurde z.B. die folgende ausgearbeitete Beispielaufgabe als Illustration der Aufgabenkategorie *Variation mit Wiederholung* verwendet:

Beispielaufgabe: Variation mit Wiederholung

In einer Urne befindet sich eine weiße, eine gelbe, eine rote, eine grüne und eine blaue Kugel. Nacheinander werden drei Kugeln gezogen, die jeweils sofort wieder in die Urne zurückgelegt werden. Wie groß ist die Wahrscheinlichkeit zuerst die rote, dann die blaue und dann wieder die rote Kugel zu ziehen?

- In der geschilderten Situation stehen 5 Kugeln zur Auswahl. Dies ist die Menge, aus der ausgewählt wird ($n = 5$).
- Es werden insgesamt drei Kugeln gezogen, also geht es um eine Auswahl von 3 Kugeln aus 5 Kugeln. Die Anzahl ausgewählter Kugeln beträgt also $k = 3$. Da die Wahrscheinlichkeit gefragt ist, das erste Mal die rote, das zweite Mal die blaue und dann wieder die rote Kugeln zu ziehen, ist die Reihenfolge der Auswahl von Bedeutung.
- Setzt man diese Zahlen in die Formel für Variation mit Wiederholung ein, also $A = n^k$, so ergeben sich 53 Variationen.
- Die Wahrscheinlichkeit für eine dieser Möglichkeiten (erst rote, dann blaue, dann wieder rote Kugel) beträgt $1/125 = 0,8\%$.

In der Klausurphase wurden die folgenden drei Textaufgaben zur Bearbeitung vorgegeben:

Klausuraufgabe 1: Angleraufgabe

- Der Verein vegetarischer Angler hat 4 Mitglieder. Alle Angler haben sich verpflichtet, gefangene Fische sofort wieder zurück in den Teich zu setzen. Eines Tages angeln die Vereinsmitglieder nacheinander an einem Teich von 8 Quadratmetern, in dem sich 5 Fische befinden: Ein Zander, ein Aal, eine Forelle, ein Hecht und ein Karpfen. Alle Mitglieder angeln in absteigender Altersreihenfolge jeweils einen Fisch. Wie berechnet sich die Wahrscheinlichkeit, dass per Zufall der älteste Angler den Aal geangelt hat und der zweitälteste die Forelle?
- Lösung: Variation mit Wiederholung, $n = 5$, $k = 2$.

Klausuraufgabe 2: Hundeaufgabe

- Ein Tierheim sucht für 11 Hunde ein Zuhause. 4 der Hunde sind Terrier, die restlichen 7 sind Mischlinge. Es melden sich 2 blonde und 4 schwarzhaarige Kinder, die ein Haustier suchen. Um Streit zu vermeiden, werden die Hunde per Zufall ausgelost. Zuerst ziehen die schwarzhaarigen Kinder jeweils ein Los. Wie berechnet sich die Wahrscheinlichkeit, dass jedes schwarzhaarige Kind einen Terrier bekommt?
- Lösung: Kombination ohne Wiederholung, $n = 11$, $k = 4$.

Klausuraufgabe 3: Ritteraufgabe

- Beim 9. Königsturnier beteiligen sich 10 Ritter. Der König stellt für das Turnier 12 Pferde. Die Ritter beginnen mit verbundenen Augen, sich per Zufall ein Pferd auszuwählen. Zuerst wählt der schwerste Ritter, dann der zweitschwerste, usw. Wie berechnet sich die Wahrscheinlichkeit, dass der schwerste Ritter das größte Pferd, der zweitschwerste das zweitgrößte und der drittschwerste Ritter das drittgrößte Pferd bekommt?
- Lösung: Variation ohne Wiederholung, $n = 12$, $k = 3$.

Die kognitive Aufgabenanalyse in Teil II dieses Arbeitsberichtes bezieht sich damit auf Versuchspersonen, welche die Klausuraufgaben 1 - 3 auf der Basis von Wissen bearbeiten, das sie aus dem in Anhang D wiedergegebenen Instruktionmaterial erworben haben.

3. Bearbeitungsstrategien und ihre Wissensvoraussetzungen

Bei der Bearbeitung von Textaufgaben aus der Kombinatorik können verschiedene Strategien oder Vorgehensweisen unterschieden werden. Bearbeitungsstrategien können als Abfolge von Teilzielen und den jeweils zugehörigen Schritten zur Zielerreichung definiert werden. Unterschiedliche Strategien unterscheiden sich einerseits in ihrer Effektivität und andererseits in ihren Ressourcenanforderungen. Ressourcenanforderungen, die für den erfolgreichen Einsatz einer bestimmten Bearbeitungsstrategie gegeben sein müssen, betreffen z.B. Zeitanforderungen, Arbeitsgedächtnisanforderungen, Anforderungen an die verfügbare externe Information und Anforderungen an das vorliegende Wissen. Wir betrachten hier vor allem die Wissensvoraussetzungen von Strategien, da ein zentraler Fokus kognitiver Aufgabenanalysen darin besteht, das zur Bearbeitung einer Aufgabe mit Hilfe einer bestimmten Strategie nötige Wissen formal zu explizieren.

Strategien lassen sich auf unterschiedlichen Auflösungs niveaus und mit unterschiedlich engem Bezug zur jeweiligen Aufgabendomäne beschreiben. In diesem Abschnitt werden Bearbeitungsstrategien für Kombinatorikaufgaben auf drei unterschiedlichen Auflösungs ebenen dargestellt:

- Zunächst wird die in der Psychologie des Problemlösens verbreitete Unterscheidung zwischen *suchbasierten, beispielbasierten und schemabasierten Bearbeitungsstrategien* eingeführt.
- Da wir uns auf eine experimentelle Anordnung beziehen, in der vorwiegend ausgearbeitete Beispielaufgaben zur Wissensvermittlung eingesetzt werden, sollen im zweiten Schritt vier *verschiedene beispielbasierte Bearbeitungsstrategien* unterschieden werden (*structure mapping, analogical replay, principle-cueing, principle-interpretation*). Obwohl alle beispielbasierten Strategien auf Wissen über Beispielaufgaben zurückgreifen, unterscheiden sich diese Strategien darin, wie dieses Wissen eingesetzt wird, um eine Lösung für eine Testaufgabe zu finden.
- Im dritten Schritt werden die zuvor eingeführten Unterscheidungen konkretisiert und auf die Bearbeitung von Kombinatorikaufgaben angewendet. Es wird eine konkrete beispielbasierte Bearbei-

tungsstrategie vorgestellt, die auf principle-cueing beruht („*Schlüsselwort-Strategie*“), und es wird eine konkrete schematische Bearbeitungsstrategie vorgestellt („*Situationsmodell-Strategie*“).

3.1 Suchbasierte, beispielbasierte und schemabasierte Bearbeitungsstrategien

In der Psychologie des Problemlösens wird die Bearbeitung von Aufgaben typischerweise als Suche in einem *Problemraum* beschrieben (Anderson, 1993a; Newell & Simon, 1972; VanLehn, 1989). Dieser Raum wird durch den *Ausgangszustand* und den *Zielzustand* eines Problems sowie die verfügbaren *Operatoren* zur Transformation von Problemzuständen festgelegt. Ein Problemraum kann als abstrakter Zustandsraum betrachtet werden, der sich aus der Menge aller Zustände ergibt, die durch erlaubte Operatoranwendungen aus dem Ausgangszustand eines Problems hervorgehen können. Erfolgreiches Problemlösen beruht in dieser Konzeption auf zwei Leistungen: Erstens muss für ein Problem ein adäquater Problemraum konstruiert werden (*Problemrepräsentation*) und zweitens muss in diesem Problemraum ein Weg vom Ausgangszustand zum Zielzustand gefunden werden (*Problemraumsuche*). Da Problemräume oft bereits für Probleme geringer Komplexität so groß werden, dass eine erschöpfende Suche nach zielführenden Operatorsequenzen nicht durchführbar ist, muss die Suche im Problemraum auf der Basis vorhandenen Wissens eingeschränkt werden. Zur Vereinfachung und zur Kontrolle des Suchprozesses können eine Reihe von *Problemlösestrategien* eingesetzt werden. In Abhängigkeit von den notwendigen Wissensvoraussetzungen können nach Ansicht vieler Autoren drei Arten übergeordneter Problemlösestrategien unterschieden werden, die man als *suchbasiertes, beispielbasiertes und schematisches Problemlösen* bezeichnen kann (Carbonell, 1986; Novick, 1988).

Suchbasiertes Problemlösen beruht auf bereichsübergreifenden Suchheuristiken wie der *Mittel-Ziel-Analyse*, die es ermöglichen, die Lösungssuche in einem Problemraum zu steuern. Bei der Mittel-Ziel-Analyse wird z.B. vom aktuellen Problemzustand ausgehend derjenige Operator ausgewählt, der die Distanz zum Zielzustand maximal reduziert (Differenzreduktion). Ist die Anwendung dieses Operators nicht möglich, weil Anwendungsvoraussetzungen nicht erfüllt sind, so wird zunächst das Teilziel gebildet, einen Zustand herbeizuführen, der die Voraussetzungen des Operators erfüllt (*subgoal*). Sobald dieses Teilziel erreicht ist, wird der ursprüngliche Operator ausgeführt und versucht, die verbleibende Differenz zum Zielzustand zu reduzieren (Anderson, 1993a). Bereichsübergreifende Suchheuristiken wie die Mittel-Ziel-Analyse erlauben eine vorwissensarme Problembearbeitung. Erforderliche Kenntnisse beschränken sich auf erlaubte Operatoren mit ihren Effekten und Anwendungsvoraussetzungen und auf Wissen darüber, welcher von zwei Problemraumzuständen eine geringere Distanz zum Zielzustand aufweist. Suchbasiertes Problemlösen wurde vorwiegend an einfachen Transformationsproblemen wie z.B. dem Turm von Hanoi untersucht.

Betrachtet man *realitätsnähere und komplexere Probleme* wie mathematische Textaufgaben, so ist über allgemeine Suchheuristiken hinaus meist ein *domänenspezifisches Wissen* zur erfolgreichen Problembearbeitung erforderlich. Dabei kann es sich um *Fallwissen* über konkrete Aufgaben und ihre

Lösungen handeln, aber auch um *generalisiertes Problemlösewissen* in Form von *Problemlöseschemata*. Dieses domänenspezifische Wissen ist die Voraussetzung für den Einsatz einer beispielbasierten oder schematischen Problemlösestrategie. Fallwissen über Problemsituationen und ihre Lösungen, Beispielwissen über Instanzen von Kategorien sowie episodisches Wissen über einzelne Ereignisse lassen sich zusammenfassend als konkretes Wissen bezeichnen (Strube & Janetzko, 1990). Im Unterschied dazu kann Wissen, das durch Abstraktion über konkrete Instanzen, Probleme oder Episoden gewonnen wurde und das sich auf Kategorien von Objekten, Aufgabenstellungen oder Vorgehensweisen bezieht, als abstraktes Wissen bezeichnet werden (Spiro & Jheng, 1990). Im Sinne dieser Unterscheidung beruht eine beispielbasierte Vorgehensweise im Wesentlichen auf konkretem Wissen, während eine schemabasierte Strategie vorwiegend auf abstraktem Wissen basiert (es sind allerdings Mischformen möglich, die sowohl konkretes als auch abstraktes Wissen voraussetzen).

Beispielbasiertes Problemlösen beruht im Unterschied zum suchbasierten Problemlösen darauf, dass eine aktuelle Problemstellung (*Zielproblem*) auf der Grundlage von konkretem Beispielwissen über eine ähnliche Problemsituation und deren Lösung (*Quellproblem*) bearbeitet wird. Die bekannteste beispielbasierte Bearbeitungsstrategie ist das analoge Problemlösen im Sinne eines *structure mapping*. Dabei wird eine bekannte Quellproblemlösung als Muster für die Konstruktion einer analogen Zielproblemlösung verwendet (Holyoak, Novick & Melz, 1994). Voraussetzung dafür ist es, dass ein entsprechendes Quellproblem bemerkt, gesucht oder erinnert wird (*reminders* im Sinne von Ross, 1984). Analoges Problemlösen im Sinne eines *structure mapping* besteht aus einer Sequenz von Schritten. Im Anschluss an die Auswahl eines Quellproblems werden dabei strukturelle Korrespondenzen zwischen Quell- und Zielproblem hergestellt, die es erlauben, eine hypothetische Zielproblemlösung zu inferieren, die schließlich an die Details des Zielproblems adaptiert werden kann (Holyoak, Novick & Melz, 1994; Kedar-Cabelli, 1988). Grundlegend für analoges Problemlösen sind sogenannte *strukturelle Ähnlichkeiten* zwischen Quell- und Zielproblem. Zwei Probleme sind strukturell ähnlich (oder isomorph), wenn sie durch die gleiche Lösungsprozedur bearbeitet werden können. Wissensvoraussetzungen der analogen Problemlösestrategie bestehen damit im Wesentlichen in Fallwissen über Quellproblemlösungen und in Wissen über strukturelle Ähnlichkeiten von Problemen (Carbonell, 1986).

Beispielbasierte Bearbeitungsstrategien werden auch im Bereich der Künstlichen Intelligenz beschrieben, und zwar unter der Bezeichnung „fallbasiertes Schließen“ (*case-based reasoning*; Kolodner, 1993; Riesbeck & Schank, 1989; Strube & Janetzko, 1990; Weber, 1994). Theorien des fallbasierten Schließens postulieren entsprechend ihrer Herkunft aus der KI und im Unterschied zu psychologischen Analogietheorien technisch gut realisierbare Mechanismen zur Nutzung von Fallwissen (Carbonell, 1986; Veloso, 1994). Ein weiterer Unterschied zu Analogietheorien besteht darin, dass Theorien des fallbasierten Schließens sich ausschließlich mit Analogien zwischen *konkreten episodischen Problemfällen innerhalb einer Domäne* befassen, während analoges Problemlösen weder auf eine einzelne Domäne noch auf episodische Probleminstanzen beschränkt ist (Nelson, Thagard & Hardy, 1994; Thagard, 1996).

Schemabasiertes Problemlösen unterscheidet sich von beispielbasiertem Problemlösen darin, dass nicht auf konkretes Wissen über einzelne Quellprobleme und ihre Lösungen, sondern auf generalisiertes Problemlösewissen in Form von *Problemschemata* oder *abstrakten Regeln* zurückgegriffen wird (Anderson, 1993b; Cummins, 1992; Gick & Holyoak, 1983; Marshall, 1995; Schoenfeld, 1985). Problemschemata werden durch Abstraktion über analoge Problemsituationen gebildet und repräsentieren unterschiedlich spezifische Klassen oder Kategorien von Problemen und die dazugehörigen Lösungsmuster (Anderson, 1993b; VanLehn, 1989). „A schema is defined as a cognitive construct that permits problem solvers to recognize problems as belonging to a particular category requiring particular moves for solutions“ (Sweller, 1989, S. 458).

Auf der Basis von Problemschemata sind neue Probleme damit unmittelbar als Instanzen von Problemkategorien encodierbar, so dass Lösungsprozeduren ohne explizite Lösungssuche direkt aus dem Gedächtnis abrufbar sind (Anderson, 1993a). Ein wesentliches Merkmal von Problemschemata besteht darin, dass Probleme nach sogenannten *strukturellen Aufgabenmerkmalen* kategorisiert werden, die für die Anwendbarkeit einer bestimmten Lösungsprozedur wesentlich sind. Oberflächliche Aufgabenmerkmale sind demgegenüber irrelevant in Bezug auf die Aufgabenlösung und beschreiben lediglich den Kontext, in den eine Aufgabe eingebettet ist. Man kann spezifische Schemata für umgrenzte Problemsituationen und unspezifische Schemata für einen breiteren Bereich von Anwendungsfällen unterscheiden. In der Problemraumkonzeption sind spezifische Schemata als *Makrooperatoren* darstellbar (Koedinger & Anderson, 1990), die es erlauben, Lösungssequenzen für konkrete Probleme direkt aus dem Gedächtnis abzurufen. Unspezifische Schemata geben hingegen oft nur Lösungsmuster an, durch die ein Problem in Teilziele („Inseln“ im Problemraum) zerlegt werden kann. Diese Zerlegung schränkt die Menge möglicher Lösungswege im Problemraum ein (Carbonell, 1986).

Beide Klassen wissensintensiver Problemlösestrategien (beispielbasiertes und schemabasiertes Problemlösen) setzen mehr oder weniger stark die Nutzung von Analogien voraus. Während z.B. beim analogen Problemlösen ein Lösungsvorschlag für ein aktuelles Zielproblem auf Grund bestimmter Korrespondenzen zu einem bekannten Quellproblem analog inferiert wird, beruht die schematische Problemlösestrategie auf der Anwendung von Problemschemata, die aus dem Vergleich analoger Probleme induktiv inferierbar sind. Es kann angenommen werden, dass die Abstraktion über Beispielprobleme und die damit verbundene Identifikation struktureller Aufgabenmerkmale der einzige Weg zum Erwerb abstrakten schematischen Problemlösewissens ist (vgl. Cummins, 1992; Anderson, 1993b). Da mathematisches Wissen nach Michener (1978) zu einem großen Teil aus Beispielproblemen und Problemschemata besteht, sind beide Klassen von Bearbeitungsstrategien für mathematische Problemlöseprozesse bedeutsam.

Vor allem im Bereich der Expertiseforschung gelten jedoch schemabasierte Bearbeitungsstrategien als grundlegend für die überragenden Problemlöseleistungen von Experten im Vergleich zu Novizen. Viele Befunde deuten darauf hin, dass Experten über eine Vielzahl von Problemschemata verfügen, die jeweils Klassen von Problemen mit gleichen Lösungsmustern abstrakt repräsentieren (vgl. Cummins, 1992, Reimann & Chi, 1989). Da Problemschemata hierarchisch organisierbar sind, können sie

mehr oder weniger umfangreiche Problemklassen repräsentieren und mit unterschiedlich konkreten Lösungsmustern verknüpft sein. Eine Überlegenheit von Experten wird vor allem im Bereich spezifischer Schemata für umgrenzte Problemsituationen und deren Lösungen angenommen. Diese Problemschemata erlauben es Experten, neue Probleme als Instanzen von bekannten Problemklassen wahrzunehmen und entsprechende Lösungsansätze direkt aus dem Gedächtnis abzurufen. Van Lehn (1989) fasst diese Auffassung plakativ zusammen, wenn er postuliert: „expertise allows one to substitute recognition for search“ (S. 529) und „schemas are the key to understanding expertise“ (S. 569).

Eine Alternative zur Erklärung von Expertenleistungen auf der Grundlage verfügbarer Problemschemata ist der prozedurale Ansatz von Anderson (1993a,b), der auf der Unterscheidung von deklarativem und prozeduralem Wissen beruht. Deklaratives Wissen umfasst dabei sowohl konzeptuelles Wissen über Kategorien als auch Faktenwissen über konkrete Sachverhalte und ist in Form propositionaler Netzwerke oder in schemaartigen Strukturen (*chunks*) organisiert. Prozedurales Wissen über Vorgehensweisen, Methoden oder Lösungsprozeduren ist direkt in Handlungen umsetzbar und wird in Form von Produktionsregeln enkodiert. Anderson geht davon aus, dass Expertise durch den Erwerb bereichsspezifischer, automatisierter Produktionsregeln entsteht. Diese Regeln verknüpfen bestimmte deklarativ repräsentierte Sachverhalte, z.B. Merkmale von Problemen, mit bestimmten Operationen, z.B. Sequenzen von Lösungsschritten. Obwohl die Verwendung von Produktionsregeln als prozedurales Repräsentationsformat von einer eher deklarativen Schemarepräsentation von Problemkategorien unterschieden werden muss (vgl. Reimann & Chi, 1989), sind die Gemeinsamkeiten beider Ansätze zur Erklärung von Expertise größer als ihre Unterschiede.

Problemschemata sind zunächst relationale Strukturen, die komplexe Konfigurationen von Merkmalen deklarativ mit Hilfe variabilisierter Attribut-Wert-Paare repräsentieren. Damit diese Problemschemata jedoch den unmittelbaren Abruf von Lösungswissen ermöglichen, der sich in der Problembearbeitung von Experten zeigt, wird im Allgemeinen angenommen, dass Problemschemata prozedurale Bestandteile aufweisen können, d.h. mithilfe bestimmter Attribute auf Lösungspläne oder auszuführende Operationen verweisen (vgl. Marshall, 1995; Reimann & Chi, 1989).

Demgegenüber sind Produktionen zwar als prozedurale Regeln formuliert, sie nehmen in ihrem Bedingungs- und Aktionsteil aber auf deklarative Muster Bezug, die als schematische Strukturen konzipiert sind (Anderson, 1993b). Koedinger und Anderson (1990) argumentieren im Bereich geometrischer Beweise dafür, Problemschemata im Rahmen einer Produktionssystemarchitektur durch Gruppen von Produktionen umzusetzen. Auch Larkin, McDermott, Simon und Simon (1980) gehen davon aus, dass Problemsituationen in einem ersten Schritt mithilfe von Schemata repräsentiert werden, während in einem zweiten Schritt Produktionsregeln auf die schematische Repräsentation angewendet werden. Gruppen von Produktionen, die einem gemeinsamen Oberziel zugeordnet sind, bezeichnen sie als *Methoden*. Cooper und Sweller (1987) sehen den Erwerb deklarativer Problemschemata und die Automatisierung zugehöriger Produktionsregeln als zwei unterschiedliche Lernmechanismen, die nacheinander einsetzen und kombiniert die gute Performanz von Experten erklären können. Auch Anderson, Fincham und Douglass (1997) postulieren einen mehrstufigen Prozess des Fertigkeitser-

werbs, der mit der analogen Nutzung von Lösungsbeispielen beginnt und über die Entwicklung abstrakter deklarativer Repräsentationen bis zur Prozeduralisierung und Automatisierung aufgabenspezifischer Regeln reicht.

3.2 Varianten beispielbasierter Bearbeitungsstrategien

Da wir uns im Rahmen dieses Arbeitsberichts auf eine experimentelle Anordnung beziehen, in der vorwiegend ausgearbeitete Beispielaufgaben zur Wissensvermittlung eingesetzt werden und in der Lerner vermutlich nicht hinreichend lange untersucht werden, um fortgeschrittene Stadien des Fertigkeitserwerbs im Sinne einer umfangreichen Schemaabstraktion und Regelautomatisierung beobachten zu können, werden im weiteren *beispielbasierte Bearbeitungsstrategien* genauer betrachtet und ausdifferenziert. Es werden vier verschiedene Strategien unterschieden, die mit den Schlagworten *structure mapping*, *analogical replay*, *principle-cueing* und *principle-interpretation* bezeichnet werden. Alle vier Strategien greifen auf Wissen über konkrete Beispielaufgaben zurück, unterscheiden sich jedoch darin, wie dieses Wissen eingesetzt wird, um die Lösung für eine Testaufgabe zu finden. *Structure mapping* und *analogical replay* greifen sehr umfangreich auf Beispielwissen zurück und können daher zu der Strategiekategorie *maximale Analogienutzung* zusammengefasst werden. *Principle-cueing* und *principle-interpretation* greifen hingegen nur begrenzt auf Beispielwissen zurück und können daher zu der Strategiekategorie *minimale Analogienutzung* zusammengefasst werden.

Minimale und maximale Analogienutzung: Das Ausmaß und die Zielsetzung, mit der bei der Bearbeitung eines aktuellen Problems auf analoge Probleme mit bekannten Lösungen zurückgegriffen wird, ist eine wichtige strategische Entscheidung. Sowohl Chi, Bassok, Lewis, Reimann & Glaser (1989) als auch VanLehn und Jones (1993) konnten empirisch belegen, dass gute Problemlöser bei der Bearbeitung eines Problems weniger häufig auf analoge Beispiele zurückgriffen und dass sie Beispiele auf andere Weise nutzten als schlechte Problemlöser. Während erfolglose Problemlöser mehr Zeit damit verbrachten, Beispielprobleme einfach nur zu lesen und Lösungsschritte direkt zu übertragen, nutzten erfolgreiche Problemlöser Beispielprobleme häufiger, um selbst formulierte Lösungsschritte für das aktuell bearbeitete Problem mit Schritten in der Beispiellösung zu vergleichen (vgl. Reimann, 1997). VanLehn und Jones (1993, S. 338) schreiben dazu:

When solving homework exercises, human students often notice that the problem they are about to solve is similar to an example. They then deliberate over whether to refer to the example or to solve the problem without looking at the example. We present protocol analyses showing that effective human learners prefer not to use analogical problem solving for achieving the base-level goals of the problem, although they do use it occasionally for achieving meta-level goals, such as checking solutions or resolving certain kinds of impasses. On the other hand, ineffective learners use analogical problem solving in place of ordinary problem solving, and this prevents them from discovering gaps in their domain theory.

Um das Ausmaß der Analogienutzung beim Problemlösen zu beschreiben, postuliert VanLehn (1996) zwei Strategieklassen, die er minimale und maximale Analogienutzung nennt. *Minimale Analogienutzung* ist dann gegeben, wenn ein Problemlöser die Ähnlichkeit zwischen einem aktuell bearbeiteten Problem und einem Problem mit bekannter Lösung bemerkt, ohne jedoch sofort auf das bekannte Problem zurückzugreifen. Stattdessen versucht er zunächst, selbstständig eine Problemlösung zu finden. Ein Bezug auf das analoge Problem mit bekannter Lösung findet nur dann statt, wenn der Problemlöser alleine keine Lösung findet (*impasse*) und daher Hilfe benötigt. Bei der *maximalen Analogienutzung* wird hingegen unmittelbar nach dem Bemerkten einer analogen Problemsituation mit bekannter Lösung auf diese Lösung referiert und versucht, durch Übertragung eine analoge Lösung für das aktuelle Problem zu entwickeln.

Structure Mapping: Die bekannteste Strategie der vollständigen Übertragung einer analogen Lösung auf ein aktuelles Problem ist das bereits kurz beschriebene *structure mapping* (Gentner, 1983; Gentner & Markman, 1997). Die Strategie wird auch mit den Begriffen *Lösungsanalogie* (Reimann, 1997), *transformational analogy* (Carbonell, 1984) oder einfach *analoges Problemlösen* (z.B. Gick & Holyoak, 1980, 1983) bezeichnet. Der Prozess des analogen Problemlösens in diesem Sinne beginnt damit, dass bei der Bearbeitung eines aktuellen Problems nach einem ähnlichen Problem mit bekannter Lösung gesucht wird. Wird ein entsprechendes Quellproblem gefunden, so wird versucht, die einzelnen Problembestandteile den Elementen des aktuell bearbeiteten Zielproblems so zuzuordnen (*mapping*), dass die bereits bekannte Problemlösung auf das aktuelle Problem übertragen werden kann. In Folgeschritt wird dann die Problemlösung des Quellproblems als Ganzes auf das Zielproblem übertragen. Der zentrale Transferprozeß beim *structure mapping* besteht damit im Kopieren und Ersetzen von Lösungsbestandteilen (Holyoak, Novick & Melz, 1994). Die Elemente der bekannten Problemlösung werden dabei zunächst kopiert und dann jeweils durch die entsprechenden Elemente der aktuellen Problemsituation ersetzt. In der kognitionspsychologischen Literatur ist die Strategie des *structure mapping* das allgemein akzeptierte Modell des analogen Problemlösens.

Analogical Replay: Eine alternative Strategie zur Übertragung einer analogen Lösung auf ein aktuelles Problem ist das sogenannte *analogical replay* (Carbonell & Veloso, 1988). Die Strategie wird auch als *derivational analogy* (Carbonell, 1986) oder als *Herleitungsanalogie* (Reimann, 1997) bezeichnet. Die Grundidee ist dabei, dass Wissen über die Funktion und Begründung einzelner Schritte einer bekannten Problemlösung genutzt werden kann, um eine Lösung für ein neuartiges Problem herzuleiten. Es wird nicht vorausgesetzt, dass eine Isomorphie zwischen Quell- und Zielproblemen bestehen muss. Stattdessen ist es für die Herleitung einer Zielproblemlösung aus der Quellproblemlösung ausreichend, dass bei der Lösung des Zielproblems ähnliche Teilziele erreicht werden müssen wie bei der Bearbeitung des Quellproblems. Ein Vorteil dieses Ansatzes besteht darin, dass Lösungsschritte oder Gruppen von Lösungsschritten auf Grund ihrer modularen Repräsentation auch einzeln übernommen werden können. Damit ist dieser Ansatz besonders zur analogen Bearbeitung neuartiger Probleme geeignet, die eine Modifikation bekannter Lösungswege erforderlich machen. Bei der Bildung von Herleitungsanalogien wird im Vergleich zu Lösungsanalogien von einer wesentlich reichhaltigeren Repräsentation gelöster Beispielprobleme ausgegangen. Während für Lösungsanalogien nur ange-

nommen werden muss, dass für jedes Beispielproblem Merkmale der Problemsituation und die Abfolge von Lösungsschritten im Gedächtnis gespeichert sind, wird bei Herleitungsanalogien vorausgesetzt, dass darüber hinaus für die gelösten Beispielprobleme auch Teile des *Kontrollwissens* verfügbar sind, welches für die Entwicklung der jeweiligen Problemlösung von Bedeutung war. Zu diesem Kontrollwissen gehören neben der Teilzielstruktur der Problemlösung z.B. auch Kenntnisse über die im Lösungsprozess getroffenen Entscheidungen, die dabei herangezogenen Informationen sowie über Begründungen für die gewählten Lösungsschritte und die mit ihnen verfolgten Ziele. Reimann (1997, S. 78-79) charakterisiert das Vorgehen bei der Herleitungsanalogie folgendermaßen:

Die Grundidee der Herleitungsanalogie ist, die aktuelle Aufgabe zunächst zu analysieren, d.h. die ersten Problemlöseziele zu formulieren, und dann nach einem Beispiel zu suchen, dessen erste Schritte ähnlich sind, d.h. dem zumindest zu Beginn dieselbe Lösungsstrategie zugrunde liegt. Bei der weiteren Bearbeitung der Aufgabe sind bei jedem Schritt die relevanten Aspekte der früheren, beispielhaften Problemlösesituation zu rekonstruieren und zu prüfen, inwieweit sie für das aktuelle Problem tauglich sind. Sind sie es nicht, ist zu prüfen, welche anderen Lösungsmöglichkeiten im Beispiel zur Verfügung standen. Die Methode der Herleitungsanalogie stellt offensichtlich hohe Anforderungen an die Repräsentation früherer Lösungen. Zum einen wird die Ähnlichkeit zu früheren Lösungen nicht anhand von Merkmalen der Aufgabenstellung, sondern anhand von Merkmalen des Lösungswegs bewertet. Zum anderen wird für jeden einzelnen früheren Lösungsschritt überprüft, inwieweit er für die aktuelle Lösung übertragen werden kann. Die frühere Lösung wird also nicht als Ganzes übernommen, sondern Schritt für Schritt rekonstruiert. ... Carbonell hat gute Gründe, diesen relativ aufwendigen Weg vorzuschlagen, um analoges Problemlösen zu realisieren. Er mußte nämlich erfahren, daß eine einfachere Form der Analogiebildung, nämlich die Übertragung des Lösungsweges ohne Herleitungsstruktur, in den meisten Fällen zu nicht effektiven oder unsinnigen Problemlöseversuchen führte.

Carbonells Annahme einer vollständigen Speicherung sämtlicher Informationen, die für die Herleitung einer Problemlösung von Bedeutung sind, ist psychologisch vermutlich unplausibel und geht vor allem darauf zurück, dass die Konzeption des *analogical replay* in der Künstlichen Intelligenz im Rahmen der Konstruktion fallbasierter Systeme entwickelt wurde. Reimann (1997) schlägt vor, ein *Kontinuum analoger Problemlösestrategien* anzunehmen, an dessen einem Ende die relativ einfache Lösungsanalogie steht und an dessen anderem Ende sich die relativ komplizierte Herleitungsanalogie befindet.

Principle-Interpretation ist eine beispielbasierte Bearbeitungsstrategie aus dem Bereich der minimalen Analogienutzung, die darauf abzielt, eine bekannte Beispiellösung nur als Quelle für bestimmte beim Problemlösen benötigte Informationen heranzuziehen, ohne die Beispiellösung vollständig auf das Zielproblem zu übertragen. Die Strategie wird auch als *rule-instance-mechanism* bezeichnet (Ross & Kennedy, 1990; Bernardo, 1994). Die Grundannahme der Strategie besteht darin, dass der Problemlöser nicht nur über bekannte Beispielprobleme sondern auch über eine abstraktere deklarative Repräsentation von Problemkategorien und den zugehörigen Problemlöseprozeduren verfügt, oh-

ne dass er diese Prozeduren direkt ausführen könnte. Es ist also noch keine kognitive Struktur im Sinne eines Problemschemas verfügbar, die im Anschluß an die Problemklassifikation auch die prozeduralisierte und automatisierte Ausführung einer Lösungsprozedur ermöglicht. In dieser Situation kann Wissen über konkrete Beispielprobleme und ihre Lösungen verwendet werden, um die Anwendung der abstrakten Problemlöseprozedur zu exemplifizieren und diese Prozedur für das aktuelle Zielproblem zu instantiieren. Beispielprobleme können damit im Sinne der *principle-interpretation* Strategie die Anwendung abstrakter Prozeduren und Methoden verdeutlichen, indem sie illustrieren, wie diese abstrakten Wissensbestandteile in konkreten Situationen eingesetzt werden (Spiro & Jehng, 1990). Beispielprobleme erleichtern damit das Verständnis von abstrakten Prozeduren und Methoden und die Anwendung dieser Prozeduren und Methoden auf neue Problemsituationen. Im Unterschied zur maximalen Analogienutzung bei der Lösungsanalogie und der Herleitungsanalogie wird das Zielproblem bei der *principle-interpretation* Strategie jedoch in erster Linie auf der Grundlage von abstraktem deklarativem Wissen über Problemkategorien und entsprechende Problemlöseprozeduren gelöst. Beispielwissen wird bei dieser Strategie herangezogen, um die Anwendung dieses abstrakten Wissens zu unterstützen.

Principle-Cueing (Ross, 1987) ist eine weitere beispielbasierte Bearbeitungsstrategie aus dem Bereich der minimalen Analogienutzung, die nicht darauf abzielt, eine bekannte Beispiellösung vollständig auf ein Zielproblem zu übertragen, sondern Beispielwissen im Zusammenspiel mit abstrakterem deklarativem Problemlösewissen einsetzt. Die Strategie wird auch als *instance-rule-mechanism* bezeichnet (Ross & Kennedy, 1990). Die Grundannahme der Strategie besteht darin, dass bei der Bearbeitung eines Zielproblems Ähnlichkeiten zu bekannten Beispielproblemen entdeckt werden können und dass derartige Erinnerungen (*reminders*) nutzbar sind, um Hinweise auf Prozeduren zu gewinnen, die zur Lösung des aktuellen Problems eingesetzt werden können (vgl. auch Schunn und Dunbar (1996). Es wird damit angenommen, dass der Problemlöser zwar über eine Repräsentation von Problemlöseprozeduren verfügt, dass er jedoch nicht in der Lage ist, diese Prozeduren im Sinne eines Problemschemas direkt auszuwählen. Das Auftreten von Erinnerungen an ähnliche Beispielprobleme kann z.B. auf der Basis von Aktivationsmodellen durch die Summation von Aktivationen modelliert werden (vgl. Holyoak & Koh, 1987). Zusammenfassend sind *Principle-Interpretation* und *Principle-Cueing* beispielbasierte Bearbeitungsstrategien, die besonders wichtig sind, wenn Problemlöser noch nicht über vollständig ausgebildete Problemschemata verfügen und entweder bei der Zuordnung von Zielaufgaben zu Lösungsprozeduren oder bei der konkreten Umsetzung von Lösungsprozeduren auf Beispielwissen angewiesen sind.

3.3 Bearbeitungsstrategien für Textaufgaben:

Schlüsselwort-Strategie und Situationsmodell-Strategie

Im Folgenden sollen einige der bislang eingeführten Bearbeitungsstrategien konkretisiert und auf die Bearbeitung von Kombinatorikaufgaben angewendet werden. Es wird eine spezifische beispielbasierte Bearbeitungsstrategie vorgestellt, die auf *principle-cueing* beruht („*Schlüsselwort-Strategie*“), und eine

spezifische schematische Bearbeitungsstrategie, die wir als „*Situationsmodell-Strategie*“ bezeichnen. Diese Strategien werden hier zunächst in ihren Grundannahmen geschildert; in Teil II dieses Arbeitsberichtes werden lauffähige Simulationsmodelle im Rahmen der ACT-R-Architektur vorgestellt.

Schlüsselwort-Strategie: Eine relativ einfache, aber entsprechend fehleranfällige Strategie zur Bearbeitung von Textaufgaben besteht darin, auf eventuelle Schlüsselwörter im Aufgabentext zu achten, die einen Hinweis auf die für die Aufgabe adäquate Lösungsprozedur geben (Nesher & Teubal, 1975; Reed, 1999; Sowder, 1988). Diese Strategie ist besonders dann geeignet, wenn Problemlöser zwar Wissen über die Existenz verschiedene Aufgabenkategorien und die mit ihnen verbundenen Lösungsprozeduren besitzen, wenn sie jedoch nur über eine oberflächliche Repräsentation der Textaufgabe verfügen. Dies ist z.B. dann zu erwarten, wenn Problemlöser die zur Aufgabenklassifikation wesentlichen strukturellen Aufgabenmerkmale noch nicht erworben haben oder wenn sie nicht in der Lage sind, eine elaborierte Repräsentation der Aufgabenstellung zu entwickeln, aus der sich die Ausprägungen struktureller Aufgabenmerkmale entnehmen lässt. In diesen Fällen kann auf Wissen über Beispielaufgaben für die verschiedenen Aufgabenkategorien zurückgegriffen werden, indem überprüft wird, ob das Zielproblem Schlüsselwörter oder Schlüsselphrasen enthält, die für bestimmte Aufgabenkategorien typisch sind. Beispielsweise sind im Bereich einfacher arithmetischer Textaufgaben Phrasen wie „alle zusammen“ oder „mehr“ Hinweise auf Additionsaufgaben. Diese Nutzung von Schlüsselwörtern als Hinweisreize auf bestimmte Aufgabenkategorien ist allerdings eine relativ fehleranfällige Strategie, wie Reed (1999, S. 47) bemerkt:

The more mature strategies for solving word problems are based on the meaning of the text. The key-word strategy is a step in this direction but is limited. In the key-word strategy students are taught to look for an important word in the text that tells them what arithmetic operation to use. For example, teachers might tell students that they should add whenever they see the word *altogether*. According to Sowder (1988), the key-word strategy is taught by some well-meaning teachers who are not aware of how students can abuse it.

Nesher und Teubal (1975) nennen als Beispiel dafür, auf welche Weise die Schlüsselwort-Strategie zu falschen Problemlösungen führen kann, die folgenden Textaufgaben, in der das Wort “more” irreführenderweise auf eine Additionsaufgabe hindeutet, obwohl es sich um eine Subtraktionsaufgabe handelt:

The milkman brought 11 bottles of milk on Sunday. That was 4 more than he brought on Monday. How many bottles did he bring on Monday.

Die Schlüsselwort-Strategie kann auch zur Bearbeitung von Textaufgaben aus der Kombinatorik eingesetzt werden. Beispielsweise lernen Versuchspersonen in der HYPERCOMB-Lernumgebung, dass die Aufgabenkategorie *Variation mit Wiederholung* vorliegt, wenn es um eine Auswahl von Elementen geht, wobei die Reihenfolge der Auswahl relevant ist und Elemente mehrfach in einer Auswahl auftre-

ten können. Als illustrierendes Beispiel wird die folgende Urnenaufgabe präsentiert, in der Schlüsselwörter identifiziert werden können, die einen Hinweis auf diese Aufgabenkategorie geben. Entsprechende Phrasen, die vom Lerner zur Identifikation der Aufgabenkategorie genutzt werden könnten, sind fett markiert:

In einer Urne befindet sich eine weiße, eine gelbe, eine rote, eine grüne und eine blaue Kugel. **Nacheinander** werden **drei Kugeln gezogen**, die jeweils **sofort wieder in die Urne zurückgelegt** werden. Wie groß ist die Wahrscheinlichkeit **zuerst die rote, dann die blaue und dann wieder die rote Kugel zu ziehen?**

Die Worte „nacheinander“ und „zuerst“ deuten darauf hin, dass die Reihenfolge von Elementen in dieser Aufgaben relevant ist. „Sofort wieder in die Urne zurückgelegt“ gibt einen Hinweis darauf, dass Elemente mehrfach auftreten können. Und die Phrasen „drei Kugeln gezogen“ sowie „die rote, dann die blaue und dann wieder die rote Kugel zu ziehen,“ zeigen, dass es sich um eine Auswahl aus den 5 Kugeln in der Urne handelt. Die Kenntnis dieser Schlüsselwörter kann nun verwendet werden, um die Testaufgaben im Klausurteil von HYPERCOMB zu lösen. Beispielsweise finden sich in der *Angleraufgabe* viele Schlüsselwörter, die mit der vorangegangenen Urnenaufgabe übereinstimmen und somit einen Hinweis darauf geben, dass die Aufgabe vom Typ Variation mit Wiederholung sein könnte:

Der Verein vegetarischer Angler hat 4 Mitglieder. Alle Angler haben sich verpflichtet, gefangene Fische **sofort wieder zurück in den Teich** zu setzen. Eines Tages angeln die Vereinsmitglieder **nacheinander** an einem Teich von 8 Quadratmetern, in dem sich 5 Fische befinden: Ein Zander, ein Aal, eine Forelle, ein Hecht und ein Karpfen. Alle Mitglieder angeln **in absteigender Altersreihenfolge** jeweils einen Fisch. Wie berechnet sich die Wahrscheinlichkeit, dass per Zufall **der älteste Angler den Aal geangelt hat und der zweitälteste die Forelle?**

Wie in der Urnenaufgabe treten die Schlüsselwörter „nacheinander“ und „sofort wieder zurück“ auf. Die Phrase „der älteste Angler den Aal und der zweitälteste die Forelle“ kann als Hinweis darauf verstanden werden, dass es nur um die Auswahl von zwei Fischen aus den fünf Fischen im Teich geht.

Wir modellieren die Schlüsselwort-Strategie in Form einer *bottom-up*-Verarbeitung. Dabei wird angenommen, dass zunächst die Textphrasen einer Klausuraufgabe eingelesen werden. Wenn dabei Textphrasen verarbeitet werden, die identisch oder ähnlich zu Textphrasen sind, die in der Lernphase als Schlüsselwörter für bestimmte Aufgabenkategorien memoriert wurden, so kommt es durch Aktivationsausbreitung zur Aktivierung der entsprechenden Aufgabenkategorie im Gedächtnis. Wenn die gesamte Klausuraufgaben eingelesen worden ist, wird diejenige Aufgabenkategorie ausgewählt, die am höchsten aktiviert ist.

Da die Vorgehensweise bei dieser Strategie stark auf Assoziationen zwischen Schlüsselwörtern und Aufgabenkategorien beruht, die auf Grundlage von Lernbeispielen aufgebaut wurden, sollte die Strategie

besonders fehleranfällig für irreführende Assoziationen zwischen oberflächlichen Aufgabenmerkmalen und Aufgabenkategorien sein. Das Problemlöser tatsächlich häufig entsprechende Kategorisierungsfehler machen, die auf entsprechende oberflächliche Assoziationen zurückgehen, spricht für die Plausibilität dieser Bearbeitungsstrategie (Holyoak & Koh, 1987; Quilici & Mayer, 1997; Ross, 1989).

Situationsmodell-Strategie: Eine alternative Bearbeitungsstrategie für Textaufgaben aus der Kombinatorik, die über das verständnisarme Vorgehen der Schlüsselwort-Strategie insofern hinausgeht, als dass sie eine elaboriertere Repräsentation des Aufgabentextes sowie ein Verständnis der für eine Aufgabenkategorie wesentlichen strukturellen Aufgabenmerkmale voraussetzt, ist die Situationsmodell-Strategie. Die Situationsmodell-Strategie ist eine schemabasierte Strategie und beruht auf einer *top-down*-Verarbeitung. Es wird angenommen, dass für die verschiedenen Aufgabenkategorien Problemschemata verfügbar sind. In diesen werden die relevanten strukturellen Aufgabenmerkmale spezifiziert, die für eine Kategorie wesentlich sind. Die Ausprägung dieser Aufgabenmerkmale werden aus elaborierten Aufgabenrepräsentationen (Situationsmodellen) erschlossen.

Wir modellieren die Situationsmodell-Strategie dreistufig. Im ersten Schritt werden die Textphrasen einer Klausuraufgabe eingelesen und interpretiert. Aus den dabei resultierenden kompakteren Paraphrasen wird im zweiten Schritt eine elaborierte Aufgabenrepräsentation generiert, die als Situationsmodell fungiert. Im dritten Schritt werden auf Grundlage dieses Situationsmodells Inferenzen über die Ausprägungen struktureller Aufgabenmerkmale gezogen. Daraufhin wird ein geeignetes Problemschema ausgewählt, das mit diesen Ausprägungen struktureller Merkmale übereinstimmt.

Die Situationsmodell-Strategie ist in zweierlei Hinsicht stärker verständnisbasiert als die Schlüsselwort-Strategie. Zum einen wird angenommen, dass der Aufgabentext besser verstanden wird als bei der Schlüsselwort-Strategie, d.h. es wird angenommen, dass Textpassagen in abstrakterer Form paraphrasiert und in einem Situationsmodell integriert werden. Zum anderen wird angenommen, dass der Problemlöser Wissen über strukturelle Aufgabenmerkmale besitzt, die für die Kategorisierung der Testaufgaben zentral sind. Es wird angenommen, dass er weiß, nach welchen Aufgabenmerkmalen er gezielt suchen muss und es wird angenommen, dass er durch eine Inspektion des Situationsmodells entscheiden kann, wie ein strukturelles Aufgabenmerkmal bei einer Aufgabe ausgeprägt ist. Diese Annahmen entsprechen allgemein verbreiteten Vorstellungen über die Rolle von Situationsmodellen bei der Bearbeitung mathematischer Textaufgaben (Reed, 1999, S. 89f.):

Constructing a situation model in the service of understanding is not unique to algebra word problems. Text comprehension, in general, depends on constructing a model of the situation described in the text. In fact, recent efforts to improve instruction on mathematical word problems (Nathan et al., 1992; Staub & Reusser, 1995) have been influenced by theories of text comprehension in which a reader combines information in a text with prior knowledge to create a situation model of the events described in the text (Kintsch, 1988, 1994). Constructing a correct equation to represent a word problem depends on coordinating a situation model of the problem with a problem model that expresses the quantitative (algebraic) relations among the concepts described in the situation model.

Am Beispiel der Angleraufgabe aus HyperComb kann verdeutlicht werden, welche Information ein Situationsmodell in unserer Modellierung enthält.

Der Verein vegetarischer Angler hat 4 Mitglieder. Alle Angler haben sich verpflichtet, gefangene Fische sofort wieder zurück in den Teich zu setzen. Eines Tages angeln die Vereinsmitglieder nacheinander an einem Teich von 8 Quadratmetern, in dem sich 5 Fische befinden: Ein Zander, ein Aal, eine Forelle, ein Hecht und ein Karpfen. Alle Mitglieder angeln in absteigender Altersreihenfolge jeweils einen Fisch. Wie berechnet sich die Wahrscheinlichkeit, dass per Zufall der älteste Angler den Aal geangelt hat und der zweitälteste die Forelle?

Das entsprechende Situationsmodell enthält bei vollständiger Spezifikation die Angabe,

- dass es in der Angleraufgabe insgesamt 4 Subjekte (Angler) gibt,
- dass von diesen Subjekten lediglich 2 relevant sind,
- dass es insgesamt 5 Objekte (Fische) gibt,
- dass von diesen Objekten lediglich 2 relevant sind,
- dass es sich beim Angeln um eine Auswahl von Elementen handelt,
- dass das Angeln in einer bestimmten Reihenfolge stattfindet und
- dass gefangene Fische sofort wieder zurück in den Teich gesetzt werden.

Dieses Situationsmodell wird herangezogen, um Wertebelegungen für die drei Attribute *Anordnung/Auswahl*, *Reihenfolge relevant/irrelevant* und *mit/ohne Wiederholung* zu finden. Wenn eine Wertekombination inferiert wurde, die z.B. die zutreffenden Werte *Auswahl*, *Reihenfolge relevant*, *mit Wiederholung* spezifiziert, dann kommt es zur Ausführung einer Produktionsregel, die für die Kategorisierung der Aufgabe als *Variation mit Wiederholung* verantwortlich ist. Die spezifischen Problemschemata für die sechs Aufgabenkategorien bestehen also in diesem Modell sowohl aus deklarativen Repräsentationen verschiedener Aufgabenkategorien und den entsprechenden strukturellen Aufgabenmerkmalen als auch aus einer Reihe von Produktionen, die auf diesen deklarativen Einheiten operieren.

Auch die Situationsmodell-Strategie beruht auf Assoziationen zwischen verschiedenen Wissenseinheiten, aber diese Assoziationen werden im Sinne eine *top-down*-Prozesses stärker zielgesteuert genutzt als bei der Schlüsselwortstrategie. Einerseits werden Assoziationen zwischen Passagen des Aufgabentextes und aus dem Gedächtnis abrufbaren Informationen genutzt, um eine abstrahierte Paraphrase des Aufgabentextes zu generieren und in einem Situationsmodell zu integrieren. Andererseits werden bei der Extraktion der Ausprägungen struktureller Aufgabenmerkmale aus dem Situationsmodell Assoziationen zwischen Elementen des Situationsmodells und den möglichen Merkmalsausprägungen genutzt. Die Situationsmodell-Strategie setzt damit nicht nur voraus, dass gegenüber der Schlüsselwort-Strategie zusätzliche deklarative und prozedurale Wissens-elemente verfügbar sind (z.B. deklarative Repräsentationen verschiedener Aufgabenkategorien und der entsprechenden strukturellen Aufgabenmerkmale sowie Produktionen, die auf diesen deklarativen Einheiten operieren).

ren), sondern auch, dass diese Wissens-elemente im Verlauf des Lernprozesses miteinander sinnvoll durch Assoziationen verbunden werden.

Im Weiteren werden die Schlüsselwort-Strategie und die Situationsmodell-Strategie als zwei diskrete Vorgehensweisen behandelt, obwohl es sich bei dieser Annahme um eine gewisse Idealisierung handelt. Beide Vorgehensweisen können auch miteinander verbunden werden, bzw. es sind auch verschiedene Mischformen und Varianten denkbar. Für die Modellierung von Bearbeitungsstrategien ist diese Idealisierung jedoch von Vorteil. Spätestens beim Vergleich von Simulationsdaten und empirischen Daten muss jedoch berücksichtigt werden, dass die beiden hier präzisierten Strategien streng genommen Punkte auf einem Kontinuum mehr oder weniger elaborierter und ressourcenfordernder Bearbeitungsstrategien darstellen.

4. Grundannahmen der kognitiven Architektur ACT-R

Zur Formalisierung der Schlüsselwort-Strategie und der Situationsmodell-Strategie greifen wir im Teil II dieser Arbeit auf die kognitive Architektur ACT-R von Anderson und Lebiere (1998) zurück. Mit dem Begriff der kognitiven Architektur wird der Anspruch verbunden, eine umfassende Theorie menschlicher kognitiver Leistungen mit einem breiten Anwendungsbereich vorzulegen (vgl. Tack, 1987), die als *unified theory of cognition* bezeichnet wird. In einer kognitiven Architektur sollen grundlegende kognitive Strukturen und Prozesse so spezifiziert werden, dass mit Hilfe einer einheitlichen Menge von Konstrukten und Annahmen eine Vielzahl von Befunden aus der kognitionswissenschaftlichen Problemlöse-, Lern- und Gedächtnisforschung erklärt werden können.

In den letzten Jahren wurden innerhalb des kognitionswissenschaftlichen Paradigmas der symbolischen Informationsverarbeitung eine Reihe *kognitiver Architekturen* entwickelt, die auf Produktionensystemen basieren (z.B. SOAR, EPIC oder 3CAPS). Gemeinsames Kennzeichen dieser Architekturen ist die Annahme, dass Wissen (zumindest teilweise) in Form von Produktionsregeln repräsentiert wird, die auf Grund eines Mustervergleichs mit aktivierten deklarativen Gedächtniseinheiten zur Ausführung ausgewählt werden. Prinzipiell sind alle diese Architekturen geeignet, um präzise kognitive Aufgabenanalysen durchzuführen, die auf plausiblen Grundannahmen über kognitive Verarbeitungsmechanismen beruhen (vgl. Pirolli, 1999). Da ACT-R insbesondere als umfassende Theorie menschlichen Wissens sowie der Anwendung und des Erwerbs dieses Wissens konzipiert ist, haben wir uns im Kontext unseres Interesses an Lern- und Problemlöseprozessen für die Verwendung dieser Architektur entschieden.

In der ACT-R-Architektur wird angenommen, dass kognitive Systeme sich aus einer deklarativen und einer prozeduralen Gedächtniskomponente zusammensetzen, wobei sich beide Gedächtniskomponenten sowohl auf einer diskreten, symbolischen als auch auf einer subsymbolischen, kontinuierlichen

Ebene beschreiben lassen. Einen zusammenfassender Überblick über die wesentlichen Definitionen und Gleichungen von ACT-R gibt Anhang A.

4.1 Deklaratives Gedächtnis

Das deklarative Gedächtnis umfasst Wissen um Sachverhalte, welches in Form atomarer Wissens-einheiten (*chunks*) repräsentiert ist. *Chunks* sind getypte schematische Strukturen, die je nach *chunk-type* verschiedene *slots* zur Repräsentation von Attributen aufweisen. Die einzelnen *chunks* können sowohl durch diskrete symbolische Relationen (sogenannte *slot-filler*-Relationen) als auch durch kontinuierliche subsymbolische Relationen (Assoziationen) miteinander verbunden sein, wodurch ein deklaratives Netzwerk entsteht. Deklaratives Wissen ist damit in Form eines semantischen Netzwerkes gespeichert.

Diejenigen *chunks*, die zu einem Zeitpunkt Bestandteil des kognitiven Systems sind, bilden das deklarative Langzeitgedächtnis. Die Aktivierung eines deklarativen *chunks* wird durch eine Aktivationsfunktion beschrieben. Kognitive Einheiten, deren Aktivierung einen Schwellenwert überschreitet, stehen dem System für die Informationsverarbeitung aktuell zur Verfügung; sie bilden das Arbeitsgedächtnis. Die Aktivierung eines *chunks* ergibt sich additiv aus einer kontextunabhängigen Basisaktivierung (*base-level activation*) und einer Kontextaktivierung. Die Basisaktivierung spiegelt wider, wie häufig ein *chunk* seit seiner Bildung benötigt wurde, und unterliegt einem automatischen Verfall. Die Kontextaktivierung gibt die Nützlichkeit eines *chunks* im Kontext eines bestimmten Verarbeitungsziels an. Verarbeitungsziele (repräsentiert durch *goal chunks*) sind unter den deklarativen kognitiven Einheiten besonders wichtig. Sie ermöglichen die Steuerung und Kontrolle der Informationsverarbeitung, da sie einerseits als Aktivationsquellen im deklarativen Gedächtnis fungieren und damit die aktuelle verfügbare deklarative Information bestimmen, und da sie andererseits einschränken, welche Produktionsregeln zur Ausführung kommen können. Die Verwaltung von *goal chunks* geschieht durch einen Zielstapel (*last-in-first-out-stack*), wobei jedoch nur der oberste *goal chunk*, der das jeweils aktuell verarbeitungssteuernde Ziel repräsentiert, als Aktivationsquelle im deklarativen Gedächtnis fungiert. Die Kontextaktivierung eines *chunks* im deklarativen Gedächtnis ergibt sich aus der Quellaktivierung, die vom aktuellen Ziel ausgeht und der Assoziationsstärke, welche zwischen diesem *chunk* und den *filler-chunks* des aktuellen Ziels besteht. Da die Anzahl deklarativer Einheiten, die gleichzeitig im Gedächtnis aktiv gehalten werden können, oft als Arbeitsgedächtniskapazität interpretiert wird, kann die Quellaktivierung von *goal chunks* als wesentliche Determinante dieser Kapazität angesehen werden. Die Assoziationsstärke drückt die spezifische Brauchbarkeit des *chunks* im Kontext des aktuellen Verarbeitungsziels aus.

4.2 Prozedurales Gedächtnis

Die prozedurale Gedächtniskomponente der ACT-R-Architektur besteht aus symbolischen Produktionsregeln, welche einzelne kognitive Verarbeitungsschritte repräsentieren. Produktionsregeln setzen

sich aus Bedingungs- und Aktionskomponenten zusammen. Im Bedingungsteil erfolgt ein Mustervergleich mit der aktuellen Zielsetzung sowie mit deklarativen Gedächtnisinhalten (Gedächtnisabruf). Der Abruf deklarativer *chunks* im Bedingungsteil einer Produktion setzt eine genügend hohe Aktivierung dieser *chunks* voraus. Erfüllen verschiedene *chunks* eine Bedingungskomponente einer Produktion, so wird im Prozess der Instantiierung dieser Produktion der *chunk* mit der höchsten Aktivierung an diese Bedingungskomponente gebunden. Im Aktionsteil werden die Konsequenzen der Regelanwendung spezifiziert. Wenn die Aktionskomponente einer Produktion ausgeführt wird, führt dies zur Modifikation von *chunks* oder zur Generierung neuer *chunks* im deklarativen Gedächtnis.

Sind bei einer gegebenen Zielsetzung auf Grundlage des Abgleichs im Bedingungsteil prinzipiell mehrere Produktionen anwendbar, so wird auf der Basis subsymbolischer Prozesse die Produktionsregel mit dem höchsten erwarteten Netto-Nutzen zur tatsächlichen Anwendung ausgewählt (*conflict resolution*). Diese Prozesse basieren auf einer Kosten-Nutzen-Analyse. Zur Abschätzung des Nutzens durch den Einsatz einer Produktion wird deren Erfolgswahrscheinlichkeit mit dem Wert des aktuellen Ziels (operationalisiert durch die Zeit, die maximal zur Zielerreichung eingesetzt werden soll) in Bezug gesetzt. Dieser Nutzen wird an den Kosten durch den Einsatz der Produktion (operationalisiert als Zeitbedarf) relativiert, so dass sich ein erwarteter Nettonutzen einer Produktion ergibt (*expected gain E*). Dieser Wert ist um so größer, je stärker der Nutzen die Kosten überwiegt. Es wird diejenige Produktion zur tatsächlichen Anwendung ausgewählt, welcher das höchste *expected gain* zugesprochen wird.

4.3 ACT-R-Modelle

Auf der Basis von ACT-R können kognitive Aufgabenanalysen durchgeführt werden, indem man deklarative und prozedurale Gedächtnisinhalte auf der symbolischen und subsymbolischen Ebene so spezifiziert, dass ein lauffähiges Performanzmodell resultiert, das in der Lage ist, eine betrachtete Aufgabe erfolgreich zu lösen. Der Ablauf (*run*) eines derartigen Performanzmodells besteht aus einer Abfolge von Zyklen, die jeweils die Ausführung von Produktionsregeln beschreiben und aus den drei Schritten *matching*, *conflict resolution* und *execution* bestehen. Beim *matching* wird der aktuell verarbeitungssteuernde *goal chunk* mit den Produktionsregeln im prozeduralen Gedächtnis auf Übereinstimmung geprüft. Bei der *conflict resolution* wird unter den jeweils anwendbaren Produktionen diejenige mit dem höchsten erwarteten Netto-Nutzen ausgewählt. Im *execution*-Schritt wird die ausgewählte Produktion durch Abruf aktuell aktivierter deklarativer Gedächtnisinhalte instantiiert und ausgeführt. Der Zeitbedarf für diesen Gedächtnisabruf hängt von der Anzahl und Aktivierung der benötigten *chunks* sowie von der Verwendungshäufigkeit der entsprechenden Produktionen ab. Die Produktionsausführung bewirkt eine Veränderung des deklarativen Gedächtnisses oder auch des Zielstapels. Daraufhin beginnt der gleiche Zyklus von vorne.

Aufbauend auf Performanzmodelle können in ACT-R auch Lernmodelle generiert werden, die auf der Grundlage verschiedener deklarativer und prozeduraler Lernmechanismen spezifizieren, wie die Wissensvoraussetzungen, die in Performanzmodellen postuliert werden, auf der Grundlage von Lern- und

Problemlöseerfahrungen erworben werden können. Lernmechanismen werden sowohl auf symbolischer als auch auf der subsymbolischen Ebene postuliert. Symbolische Lernmechanismen betreffen die Aufnahme neuer deklarativer *chunks* und neuer Produktionsregeln in das kognitive System. Subsymbolische Lernmechanismen beschreiben eine erfahrungsbedingte Veränderung von Parametern, die auf *chunks* und Produktionsregeln definiert sind. Solche Parameter sind z.B. die Basisaktivierung von *chunks* oder die Assoziationsstärke zwischen *chunks* sowie die Erfolgswahrscheinlichkeit einer bestimmten Produktion und die mit dem Einsatz dieser Produktion verbundenen Kosten.

Teil II: Kognitive Modelle verschiedener Bearbeitungsstrategien

Zur genaueren Analyse der beiden oben dargestellten Strategien, Situationsmodell-Strategie und Schlüsselwort-Strategie, wurden im Rahmen der kognitiven Architektur ACT-R zwei Performanzmodelle entwickelt. Dabei wurde ein prototypischer Ansatz verfolgt, d.h. jedes der Modelle stellt eine prototypische Anwendung der zu Grunde liegenden Strategie dar. Auf der Darstellung der jeweiligen Strategie liegt auch der Fokus der Modellierung. Dies impliziert ein gewisses Abstraktionsniveau, so dass zu Grunde liegende Prozesse, wie etwa der Vorgang des Lesens eines Textes, lediglich abstrakt simuliert, nicht aber genauer analysiert werden.

Beiden Modellen ist gemeinsam, dass sie die Bearbeitung der Klausur von HYPERCOMB simulieren. Diese besteht aus den bereits dargestellten drei Textaufgaben, welche durch Angabe der Aufgabenkategorie und der Spezifikation von zwei Parametern gelöst werden. Für jedes Modell wird im Folgenden noch einmal kurz die Grundidee vorgestellt. Darauf aufbauend wird das Ergebnis einer Analyse der Teilzielstruktur dargestellt, welche als Grundlage der kognitiven Modellierung zu verstehen ist. Da in der kognitiven Architektur ACT-R die gesamte Verarbeitung durch das jeweils aktuelle Ziel gesteuert wird, lässt sich ein Modell gemäß der Zielstruktur in einzelne Komponenten zergliedern. Dabei umfasst eine Komponente ein Ziel sowie die Prozeduren, die zur Zielerreichung eingesetzt werden können. Dementsprechend werden für jedes Modell die konstituierenden Komponenten im Detail erläutert; im Anschluss daran wird der (Ablauf) *run* des Modells beschrieben.

5. ACT-R-Modell für die Schlüsselwort-Strategie

5.1 Grundidee der Schlüsselwort-Strategie

Wie bereits dargestellt beruht die Schlüsselwort-Strategie auf einer *bottom-up*-Verarbeitung. Dabei wird angenommen, dass Textphrasen einer Klausuraufgabe an Schlüsselwörter erinnern, welche im Instruktionsmaterial des Experimentes in ausgearbeiteten Beispielen sowie abstrakten Darstellungen vermittelt wurden (vgl. Anhang D). Diese Schlüsselwörter sind daher mit den Aufgabenkategorien

assoziiert, in deren Kontext sie erlernt wurden, bzw. geben Aufschluß über die zu spezifizierenden Parameter.

5.2 Analyse der Teilzielstruktur der Schlüsselwort-Strategie

Die Schlüsselwort-Strategie lässt sich in verschiedene Komponenten untergliedern. Dabei wird unter einer Komponente ein zu erreichendes Teilziel verstanden sowie Prozeduren, die zur Zielerreichung eingesetzt werden können. Insofern scheint es sinnvoll zu sein, zunächst die Teilzielstruktur der Strategie zu explizieren, welche die Grundlage der ACT-R-Modellierung bildet, bevor die einzelnen Komponenten der Strategie detailliert dargestellt werden.

Das oberste Ziel der Zielstruktur wird per Instruktion vorgegeben und beinhaltet die Bearbeitung des Kombinatorik-Tests. Die drei Klausuraufgaben, die der Kombinatorik-Test umfasst, können jede als diesem untergeordnetes Teilziel verstanden werden. Die Bearbeitung einer solchen Klausuraufgabe mittels der Schlüsselwort-Strategie beinhaltet zunächst das Lesen der Aufgabe. Von den einzelnen Textphrasen des Aufgabentextes ausgehend werden bereits bekannte Schlüsselwörter aktiviert, welche zur Generierung der Lösung einer Klausuraufgabe weiter verarbeitet werden. Insofern sind Interpretationen von Schlüsselwörtern Unterziele des Lesevorgangs.

Die Zielstruktur der Schlüsselwort-Strategie lässt sich folgendermaßen darstellen:

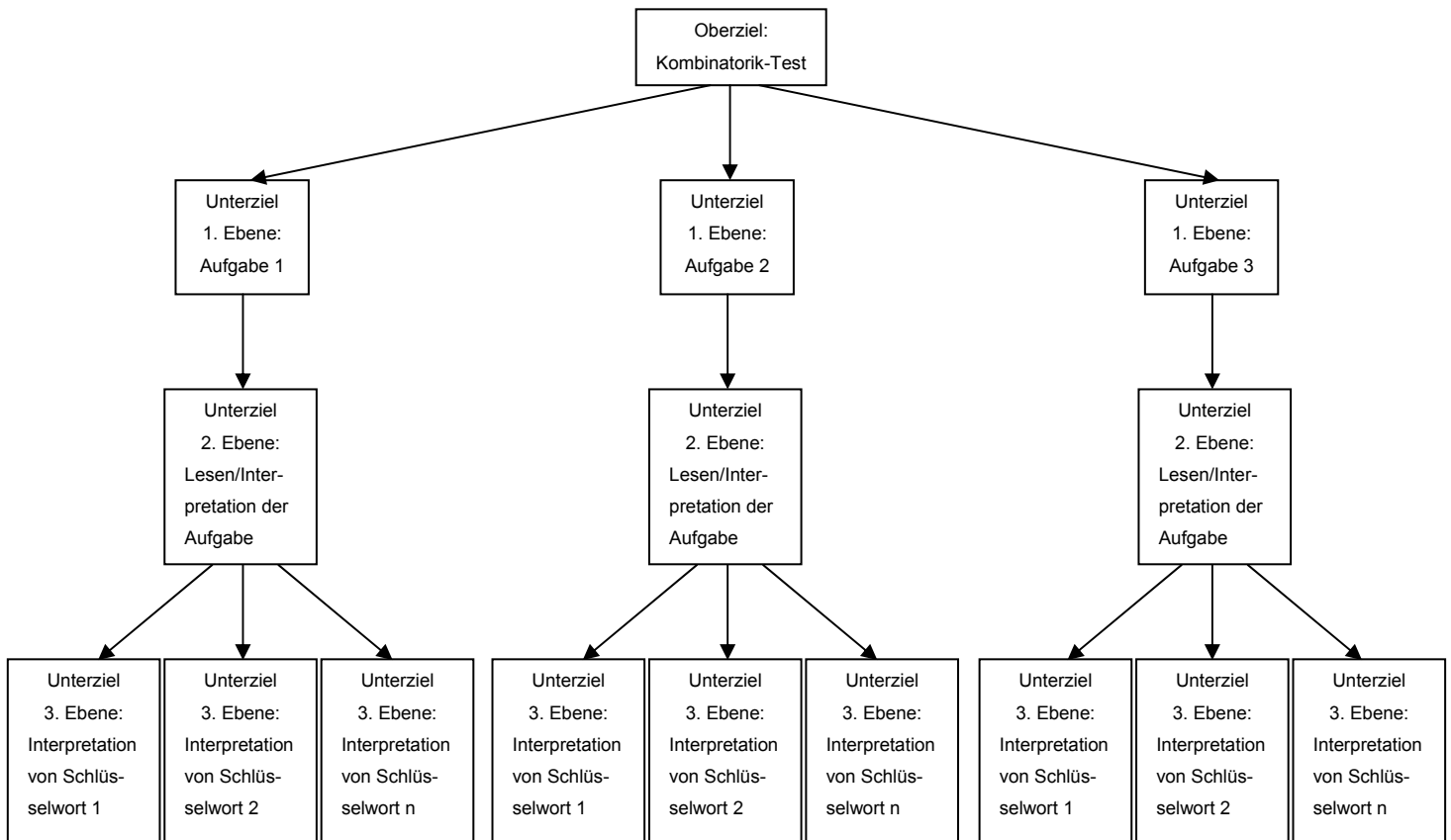


Abbildung 7: Teilzielstruktur der Schlüsselwort-Strategie

5.3 Komponenten der Schlüsselwort-Strategie

Im Folgenden werden die einzelnen Komponenten der Schlüsselwort-Strategie und ihrer Modellierung genauer dargestellt. Dabei werden jeweils diejenigen Produktionen gemeinsam dargestellt, die zur Bearbeitung eines Ziels eines bestimmten *chunk-type* eingesetzt werden können. Der Quellcode der Modellierung der Schlüsselwort-Strategie findet sich in Anhang B.

(A) Strategiekomponente: Ziel vom *chunk-type* 'combinatoric-test'

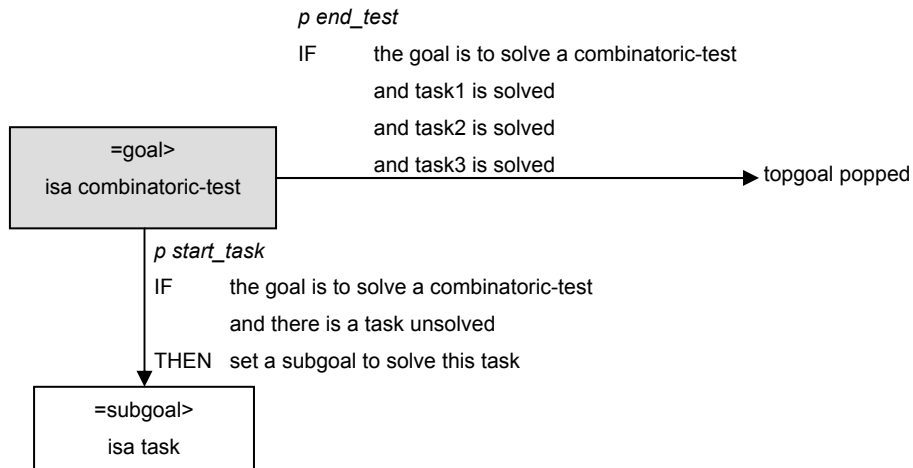


Abbildung 8: Strategiekomponente: Ziel vom *chunk-type* ,combinatoric-test'¹

Das oberste Ziel, einen Kombinatorik-Test zu lösen, liegt mit dem Start des Modells auf dem *goal stack* und umfasst die Bearbeitung der drei Klausuraufgaben. Auf dieses Ziel können zwei Produktionen zugreifen:

(1) *p start_task*

Diese Produktion legt eine der zu lösenden Klausuraufgaben (*chunk* vom *chunk-type* ,task') als *subgoal* auf den *goal stack*; wählt die Aufgabe damit also zur weiteren Bearbeitung aus. Der mit dieser Produktion startende Bearbeitungsvorgang wird im Rahmen des Modells für jede Klausuraufgabe einmal durchlaufen.

(2) *p end_test*

Wenn alle drei zu bearbeitenden Klausuraufgaben gelöst wurden, so ist das Ziel der Bearbeitung eines Kombinatorik-Tests erreicht. Daher wird es vom *goal stack* entfernt; der *run* des Modells ist damit beendet.

¹ In dieser wie in allen weiteren Abbildungen dieser Art stehen Kästchen jeweils für Ziele, Pfeile symbolisieren Produktionsregeln.

(B) Strategiekomponente: Ziel vom *chunk-type* 'task'

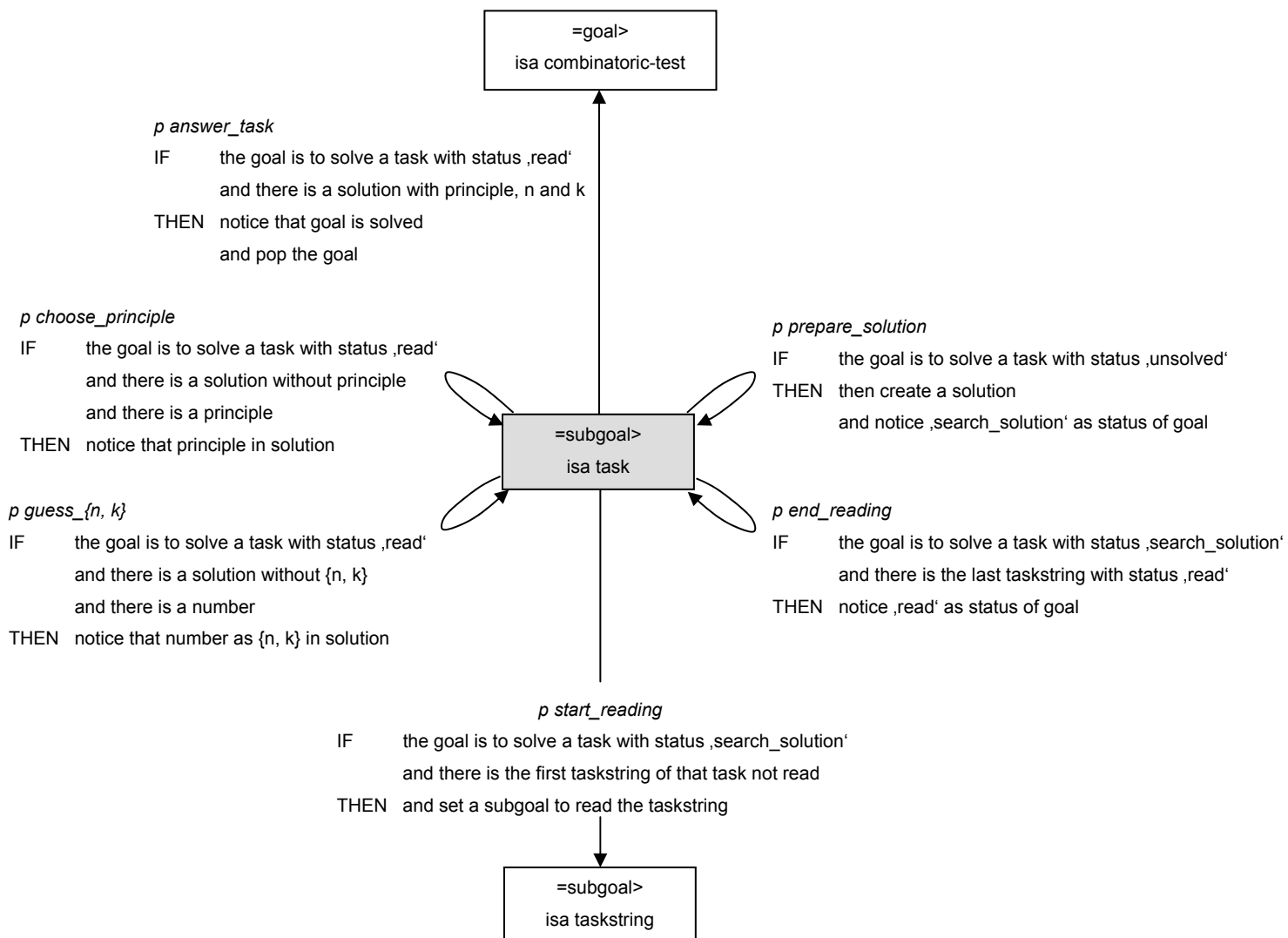


Abbildung 9: Strategiekomponente: Ziel vom *chunk-type* ,task'

Liegt eine Klausuraufgabe zur Bearbeitung auf dem *goal stack*, so können die folgenden Produktionen eingesetzt werden:

(1) *p prepare_solution*

Der Einsatz der vorliegenden Produktion führt zu der Generierung eines *chunks*, der die Lösung der Klausuraufgabe repräsentiert. In diesen können im weiteren Verlauf der Bearbeitung die angenommene Aufgabenkategorie eingetragen werden sowie die Spezifikationen der Parameter *n* und *k*. Weiterhin wird im Aktionsteil der Status der Klausuraufgabe modifiziert. Dadurch ist gewährleistet, dass die Produktion nur einmal, und zwar zu Beginn der Bearbeitung einer Aufgabe, feuern kann, so dass für jede Aufgabe genau ein *chunk* für deren Lösung erzeugt wird.

(2) *p start_reading*

Mit dem Einsatz dieser Produktion startet der Lesezyklus. Dazu wird die erste Textphrase der zu bearbeitenden Klausuraufgabe als neues *subgoal* auf den *goal stack* gelegt, so dass die weitere Verarbeitung durch diese Textphrase gesteuert wird. Der Lesevorgang kann für jede Aufgabe nur einmal initiiert werden. Dies wird dadurch gewährleistet, dass sowohl die erste Textphrase als auch die gesamte Aufgabe als ungelesen gekennzeichnet sein muss, damit die ‚*start_reading*‘-Produktion feuern kann. Während des Lesevorgangs wird auf Grundlage der jeweils betrachteten Textphrase über Schlüsselwörter versucht, ihr Informationen über die vorliegende Aufgabenkategorie bzw. die Spezifikationen der Parameter *n* und *k* zu entnehmen.

(3) *p end_reading*

Wenn die letzte Textphrase der Klausuraufgabe eingelesen wurde, so wird in der Klausuraufgabe durch entsprechende Modifikation des Status vermerkt, dass der gesamte Aufgabentext gelesen ist, so daß der Lesezyklus nicht noch einmal durchlaufen werden kann.

(4) *p choose_principle*

Ist eine Klausuraufgabe gelesen und kann ein Lösungschunk dazu aus dem Gedächtnis abgerufen werden, in welchem noch keine Aufgabenkategorie vermerkt ist, so kann die vorliegende Produktion feuern. Dabei ruft sie diejenige Aufgabenkategorie aus dem deklarativen Gedächtnis ab, die am höchsten aktiviert ist. Diese wird daraufhin im Lösungschunk als die der Klausuraufgabe zugrunde liegende Aufgabenkategorie angegeben.

(5) *p guess_{n, k}*²

Ist es während des Leseprozesses nicht gelungen, Informationen über die Spezifikationen eines oder beider Parameter *n* bzw. *k* zu sammeln, so wird der entsprechende Parameter geraten. Dazu wird eine der Ziffern, die im Aufgabentext genannt werden, im Lösungschunk als Spezifikation von *n* bzw. von *k* eingesetzt.

(6) *p answer_task*

Wenn für eine Aufgabe ein Lösungschunk vorliegt, der Wissen um die Aufgabenkategorie sowie die Spezifikation der Parameter *n* und *k* enthält, so wird die Aufgabe als gelöst gekennzeichnet und vom *goal stack* genommen; die Bearbeitung dieser Aufgabe ist damit beendet. Damit ist automatisch das Ziel, einen Kombinatorik-Test zu lösen, wieder verarbeitungssteuernd, da dieses auf dem *goal stack* an nächster Position liegt.

² Diese Darstellung steht für 2 verschiedene Produktionen, die jeweils eines der Elemente der geschweiften Klammer enthalten.

(C) Strategiekomponente: Ziel vom *chunk-type* 'taskstring'

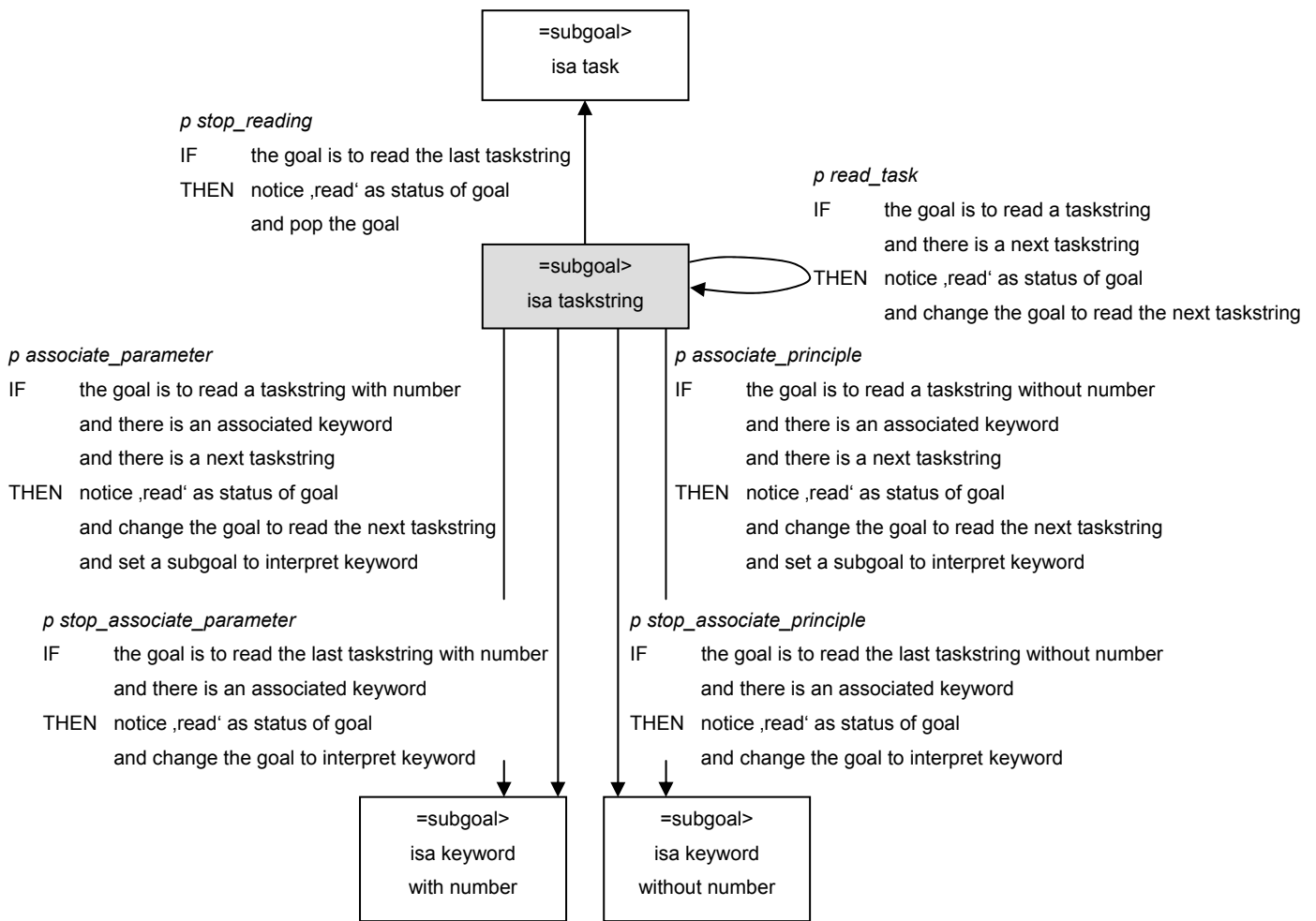


Abbildung 10: Strategiekomponente: Ziel vom *chunk-type* ,taskstring'

Wie bereits angesprochen werden mit dem Start des Lesevorgangs die einzelnen Textphrasen einer Klausuraufgabe nacheinander auf den *goal stack* gelegt. Ist eine Textphrase als oberstes Ziel auf dem *goal stack* verarbeitungssteuernd, so können Produktionen eingesetzt werden, die unter zwei Gesichtspunkten gruppiert werden können. Zum einen kann man die Produktionen danach unterscheiden, ob sie einen einfachen Lesevorgang ohne tiefere Verarbeitung der Information abbilden (Produktionen 1, 4) oder ob zusätzlich zu dem Lesevorgang ein Assoziationsprozess angestoßen wird (Produktionen 2, 3, 5, 6). Zum anderen besteht die Möglichkeit der Unterscheidung danach, ob der Lesevorgang nach dem Einsatz einer Produktion fortgeführt (Produktionen 1, 2, 3) oder beendet wird (Produktionen 4, 5, 6). Im Detail stellen sich die Produktionen wie folgt dar:

(1) *p read_task*

Durch diese Produktion wird ein einfacher Lesevorgang dargestellt, bei dem keine tiefere Verarbeitung der Information erfolgt. Die aktuell auf dem *goal stack* liegende Textphrase wird lediglich als gelesen gekennzeichnet und zu Gunsten der nachfolgenden Textphrase vom *goal stack* entfernt.

(2) *p associate_principle*

Liegt eine Textphrase an oberster Position auf dem *goal stack*, so werden von dieser ausgehend alle assoziierten Schlüsselwörter aktiviert. Die vorliegende Produktion kann alternativ zu der einfachen ‚*read_task*‘-Produktion feuern, wenn ein aktiviertes Schlüsselwort gebunden werden kann, wobei für diesen Abruf das Schlüsselwort mit der höchsten Aktivierung im deklarativen Gedächtnis ausgewählt wird. Außerdem darf in der Textphrase keine Ziffer genannt werden, damit die Produktion feuern kann. Ebenso wie beim Einsatz der ‚*read_task*‘-Produktion wird die Textphrase hier als gelesen gekennzeichnet und durch die nächste Textphrase ersetzt. Zusätzlich dazu wird allerdings das abgerufene Schlüsselwort als neues *subgoal* an oberster Stelle auf den *goal stack* gelegt, so dass es die weitere Vorgehensweise steuert.

(3) *p associate_parameter*

Ebenfalls alternativ zu der ‚*read_task*‘-Produktion kann die ‚*associate_parameter*‘-Produktion eingesetzt werden, wenn ein mit der Textphrase assoziiertes Schlüsselwort gebunden werden kann. Auch hier wird dazu dasjenige Schlüsselwort ausgewählt, das die höchste Aktivierung hat. Im Gegensatz zu der ‚*associate_principle*‘-Produktion kann die hier vorliegende Produktion nur dann feuern, wenn in der Textphrase eine Ziffer genannt wird. Als Ergebnis des Einsatzes der ‚*associate_parameter*‘-Produktion wird ebenfalls die aktuelle Textphrase als gelesen gekennzeichnet und durch die nachfolgende Textphrase auf dem *goal stack* ersetzt. Außerdem wird das abgerufene Schlüsselwort als neues *subgoal* auf den *goal stack* gelegt, wobei die in der Textphrase genannte Ziffer an dieses Schlüsselwort gebunden wird. Dadurch ist diese Ziffer auch bei der weiteren Verarbeitung des Schlüsselwortes verfügbar.

(4) *p stop_reading*

Mit dem Feuern dieser Produktion ist das Einlesen der Klausuraufgabe beendet. Die Produktion kann nur eingesetzt werden, wenn die letzte Textphrase der Klausuraufgabe auf dem *goal stack* liegt. Ebenso wie bei der ‚*read_task*‘-Produktion wird die Textphrase als gelesen gekennzeichnet und vom *goal stack* heruntergenommen; im Unterschied zur ‚*read_task*‘-Produktion wird allerdings keine andere Phrase auf den *goal stack* gelegt.

(5) *p stop_associate_principle*

Auch mit dem Einsatz dieser Produktion kann – alternativ zur ‚*stop_reading*‘-Produktion – der Leseprozess beendet werden, d.h. die aktuelle Textphrase wird als gelesen gekennzeichnet und vom *goal stack* genommen, ohne durch eine weitere Phrase ersetzt zu werden. Auch diese Produktion kann daher nur feuern, wenn die letzte Textphrase einer Klausuraufgabe auf dem *goal stack* liegt. Außerdem muss jedoch ein Schlüsselwort abrufbar sein, das ausgehend von der Textphrase aktiviert werden kann; in der Textphrase darf darüber hinaus keine Ziffer auftreten. Das Schlüsselwort wird auch hier als neues *subgoal* auf den *goal stack* gelegt, so dass es die weitere Verarbeitung steuert.

(6) *p stop_associate_parameter*

Der Leseprozess kann auch mit dem Feuern der vorliegenden Produktion beendet werden, welche wieder nur bei der letzten Textphrase auf dem *goal stack* eingesetzt werden kann. Auch hier muss ein Schlüsselwort abrufbar sein; in der Textphrase muss eine Ziffer genannt

werden. Als Ergebnis des Einsatzes der Produktion wird die letzte Textphrase als gelesen gekennzeichnet und ersatzlos vom *goal stack* entfernt. Außerdem wird das Schlüsselwort, in welchem die genannte Ziffer gebunden wird, als neues *subgoal* auf den *goal stack* gelegt.

(D) Strategiekomponente: Ziel vom *chunk-type* 'keyword'

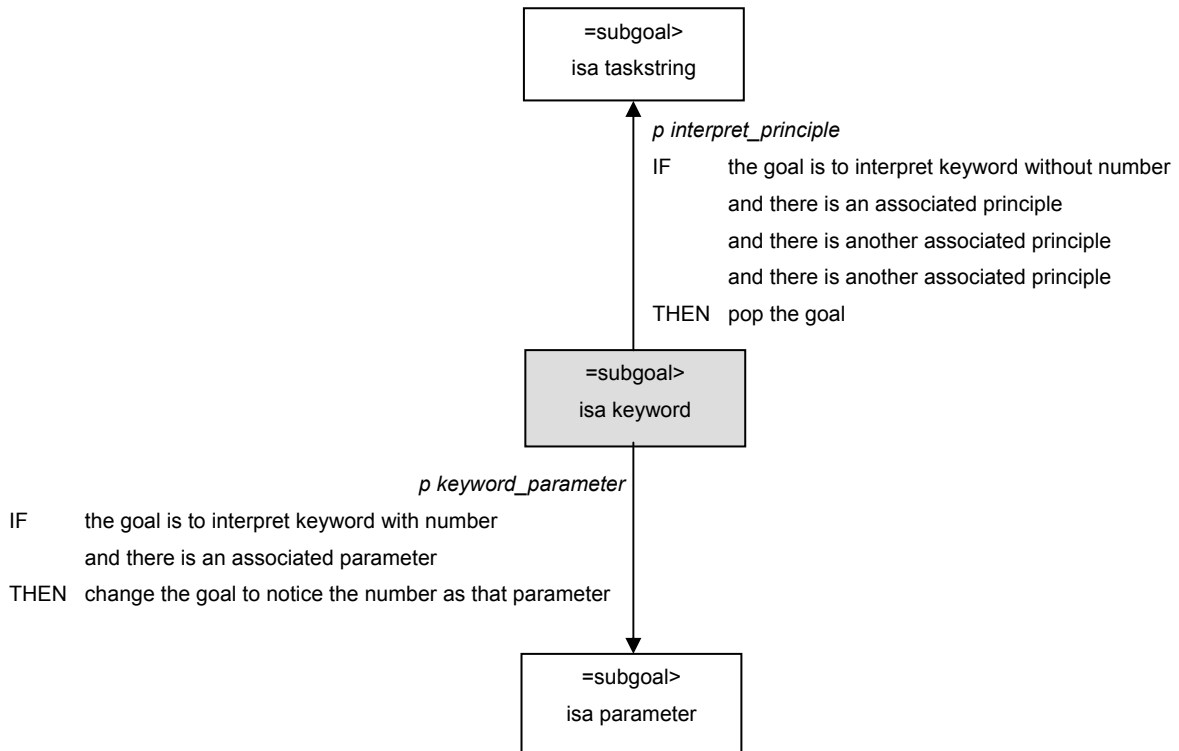


Abbildung 11: Strategiekomponente: Ziel vom *chunk-type* 'keyword'

Wurde beim Einlesen einer Textphrase ein Schlüsselwort assoziiert und als oberstes Ziel auf den *goal stack* gelegt, so stehen für dessen Verarbeitung die folgenden beiden Produktionen zur Verfügung:

(1) *p interpret_principle*

Liegt auf dem *goal stack* ein Schlüsselwort, in dem keine Ziffer gebunden ist, kann die vorliegende Produktion feuern. Ausgehend von dem Schlüsselwort werden dabei Aufgabenkategorien aktiviert und an die Produktion gebunden, die mit diesem assoziiert sind. Durch diesen Abruf wird eine Erhöhung der Basisaktivierung der entsprechenden Aufgabenkategorien erreicht. Dadurch ist gewährleistet, dass letztlich diejenige Aufgabenkategorie als Lösung der Klausuraufgabe angegeben wird (vgl. Produktion '*choose_principle*'), welche während der Bearbeitung der Klausuraufgabe am häufigsten assoziiert und abgerufen wurde. Bei jedem Einsatz der Produktion werden drei Aufgabenkategorien abgerufen. Dies ist als Heuristik zu verstehen, d.h. es wird die durch Abrufe am höchsten aktivierte Hälfte der prinzipiell möglichen Aufgabenkategorien an die Produktion gebunden. Das Schlüsselwort wird im Aktionsteil der Produktion wieder vom *goal stack* entfernt, so dass die darunterliegende Klausuraufgabe wieder verarbeitungssteuernd wird.

(2) *p keyword_parameter*

Diese Produktion kann eingesetzt werden, wenn das aktive Ziel ein Schlüsselwort ist, in dem eine Ziffer gebunden ist. Dabei wird ein Parameter abgerufen, der mit dem vorliegenden Schlüsselwort assoziiert ist. Dieser Parameter wird als neues *subgoal* auf den *goal stack* gelegt.

(E) Strategiekomponente: Ziel vom *chunk-type 'parameter'*

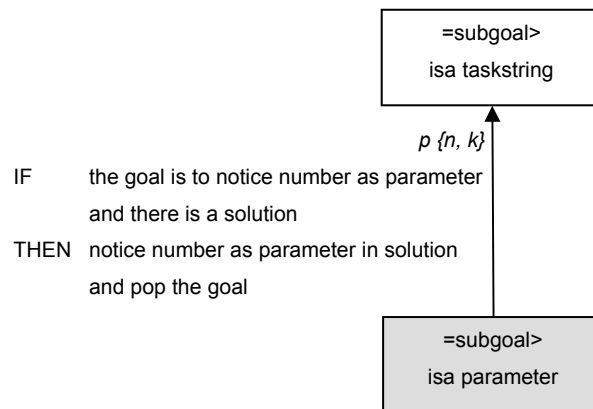


Abbildung 12: Strategiekomponente: Ziel vom *chunk-type 'parameter'*

Wenn es das Ziel ist, einen Parameter im Lösungschunk mit einem numerischen Wert zu belegen, kann eine der folgenden Produktionen feuern:

(1,2) $p\{n, k\}$ ³

Als Ergebnis des Einsatzes einer der hier vorliegenden Produktionen wird in den Lösungschunk die im Aufgabentext genannte Ziffer als jeweiliger Parameter (*n* oder *k*) eingetragen und das Ziel vom *goal stack* entfernt.

5.4 Darstellung eines *run* des Modells der Schlüsselwort-Strategie

Mit dem Starten des Modells liegt das Oberziel, einen Kombinatorik-Test zu bearbeiten, auf dem *goal stack* und ist damit verarbeitungssteuernd. Dieses Oberziel beinhaltet drei Klausuraufgaben, welche getrennt voneinander zu lösen sind. Insofern wird der nachfolgend beschriebene Ablauf für jede der Aufgaben einmal durchlaufen.

Als erstes wird eine der Klausuraufgaben, welche bisher noch nicht gelöst wurde, als neues *subgoal* auf den *goal stack* gelegt. Die Bearbeitung dieser Aufgabe beginnt mit der Initiierung eines Leseprozesses, wozu die erste Textphrase der Klausuraufgabe auf den *goal stack* gelegt wird. Für den nun stattfindenden Leseprozeß stehen bei jeder Textphrase zwei Bearbeitungsmöglichkeiten zur Verfü-

³ Diese Darstellung steht für 2 verschiedene Produktionen, die jeweils eines der Elemente der geschweiften Klammer enthalten.

gung. Zum einen kann lediglich ein einfacher Leseprozess simuliert werden, bei dem die Textphrase nur eingelesen wird, aber keine weitere Verarbeitung der Information erfolgt. Zum anderen kann zusätzlich zu diesem Einlesen ein Assoziationsprozeß einsetzen. Dabei wird ausgehend von der Textphrase ein assoziiertes Schlüsselwort aktiviert, welches als neues *subgoal* auf den *goal stack* gelegt wird und dadurch die weitere Verarbeitung steuert.

Für diese Verarbeitung gibt es wieder zwei Möglichkeiten, und zwar in Abhängigkeit davon, ob in der zu Grunde liegenden Textphrase eine Ziffer genannt wird. Ist dies der Fall, so wird diese Ziffer als Spezifikation eines der beiden zur Lösung benötigten Parameter interpretiert. Dazu wird zunächst ebenfalls über Assoziationen – diesmal ausgehend von dem Schlüsselwort – ein Parameter aktiviert, abgerufen und an Stelle des Schlüsselworts als neues *subgoal* auf den *goal stack* gelegt. Bei der Bearbeitung dieses *subgoals* wird die Ziffer als Spezifikation des entsprechenden Parameters im Lösungschunk vermerkt; das *subgoal* wird daraufhin wieder vom *goal stack* entfernt. Wurde in der Textphrase keine Ziffer genannt, so erfolgt eine Interpretation des Schlüsselworts im Hinblick auf mögliche Aufgabenkategorien der Klausuraufgabe. Dazu werden die drei Aufgabenkategorien abgerufen, welche mit dem Schlüsselwort auf Grund des gemeinsamen Auftretens in der Lernphase am höchsten assoziiert sind. Dadurch wird die Basisaktivierung dieser Aufgabenkategorien erhöht; entsprechend steigt die Abrufwahrscheinlichkeit im weiteren Verlauf der Bearbeitung. Nach dieser Interpretation des Schlüsselworts wird es wieder vom *goal stack* entfernt.

Unabhängig davon, ob nur ein einfacher Leseprozeß simuliert wurde oder ein Assoziationsprozess stattfand, endet die Bearbeitung einer Textphrase damit, dass sie auf dem *goal stack* durch die nachfolgende Textphrase ersetzt wird. Für diese bestehen wieder die Möglichkeiten, nur gelesen zu werden oder zusätzlich Ausgangspunkt der Aktivierung eines assoziierten Schlüsselworts zu sein. Dies gilt auch für die letzte Textphrase einer Klausuraufgabe, nach deren Bearbeitung der Leseprozeß abgeschlossen ist.

Damit liegt wieder die Klausuraufgabe als aktives Ziel auf dem *goal stack* und steuert so die weitere Verarbeitung. In der Aufgabe ist der Abschluß des Leseprozesses vermerkt, wodurch sichergestellt ist, dass die Textphrasen nicht noch einmal eingelesen werden können. Nun wird die Aufgabenkategorie, welche durch die Abrufe ausgehend von den verschiedenen Schlüsselwörtern am höchsten aktiviert wurde, als Lösung der Klausuraufgabe im Lösungschunk eingetragen. Ist es im Verlauf des Leseprozesses nicht gelungen, Informationen zu einem oder beiden der Parameter n und k zu finden, so wird die Spezifikation des entsprechenden Parameters geraten. Damit liegen alle Informationen, die zur Lösung der Klausuraufgabe angegeben werden müssen, im Lösungschunk vor. Diese werden als *output* ausgegeben; die Klausuraufgabe wird als gelöst gekennzeichnet und vom *goal stack* entfernt.

Damit wird das Ziel, den Kombinatorik-Test zu lösen, wieder verarbeitungssteuernd, so dass der hier beschriebene Ablauf wieder von vorne beginnen kann. Es wird also eine der Aufgaben, zu der noch keine Lösung vorliegt, als neues *subgoal* auf den *goal stack* gelegt und bearbeitet. Wurde auf diese

Weise zu allen drei Klausuraufgaben eine Lösung erarbeitet, kann auch das Ziel des Kombinatorik-Tests vom *goal stack* genommen werden; der *run* des Modells ist damit beendet.

6. ACT-R-Modell für die Situationsmodell -Strategie

6.1 Grundidee der Situationsmodell-Strategie

Die Situationsmodell-Strategie beruht auf *top-down*-Prozessen, mittels derer strukturiert nach relevanten strukturellen Aufgabenmerkmalen gesucht wird. Dabei wird angenommen, dass beim Lesen einer Textaufgabe zunächst ein abstrakteres Situationsmodell inferiert wird, das die dargestellte Aufgabensituation in komprimierter Form repräsentiert. In einem zweiten Schritt wird dann dieses Situationsmodell systematisch mit den möglichen Ausprägungen struktureller Aufgabenmerkmale verglichen. Anhand dieser Ausprägungen kann schließlich die zu Grunde liegende Aufgabenkategorie identifiziert werden, welche der gestellten Aufgabe zu Grunde liegt.

6.2 Analyse der Teilzielstruktur der Situationsmodell-Strategie

Auf Grundlage der Annahmen, die zur Vorgehensweise bei der Situationsmodell-Strategie getroffen wurden, lässt sich die Strategie in verschiedene Komponenten zergliedern. Jede der Komponenten ist durch ein Teilziel gekennzeichnet, das erreicht werden soll, sowie durch Prozeduren, welche zur Zielerreichung eingesetzt werden können. Bevor die einzelnen Komponenten in ihrer Umsetzung in einem ACT-R-Modell detailliert dargestellt werden, soll zunächst die Teilzielstruktur aufgezeigt werden, welche der Situationsmodell-Strategie zu Grunde liegt.

Durch die Instruktion wird das oberste Ziel vorgegeben, welches darin besteht, einen Kombinatorik-Test zu bearbeiten. Die drei Klausuraufgaben, aus denen sich der Kombinatorik-Test zusammensetzt, sind jeweils als untergeordnetes Ziel zu verstehen. Die Bearbeitung einer Klausuraufgabe durch die Anwendung der Situationsmodell-Strategie erfordert zunächst das Lesen und Interpretieren des Aufgabentextes. In einem zweiten Schritt wird auf Grundlage des Lese- und Interpretationsprozesses ein Situationsmodell erstellt, welches die Situation charakterisiert, die der Aufgabe zu Grunde liegt. Dieses Situationsmodell dient dann als Basis für die Vervollständigung eines Lösungschunks. Gegenüber der Schlüsselwort-Strategie ist der Lösungschunk bei der Situationsmodell-Strategie komplexer. Er repräsentiert nicht nur die Aufgabenkategorien und die numerischen Spezifikationen von Parametern, sondern auch die Ausprägungen struktureller Aufgabenmerkmale. Er bildet damit den Kern eines Lösungsschemas, das als eine Struktur aufgefasst werden kann, die sich aus dem Lösungschunk sowie den mit ihm verbundenen Produktionsregeln ergibt.

Die angenommene Teilzielstruktur bei der Situationsmodell-Strategie lässt sich folgendermaßen darstellen:

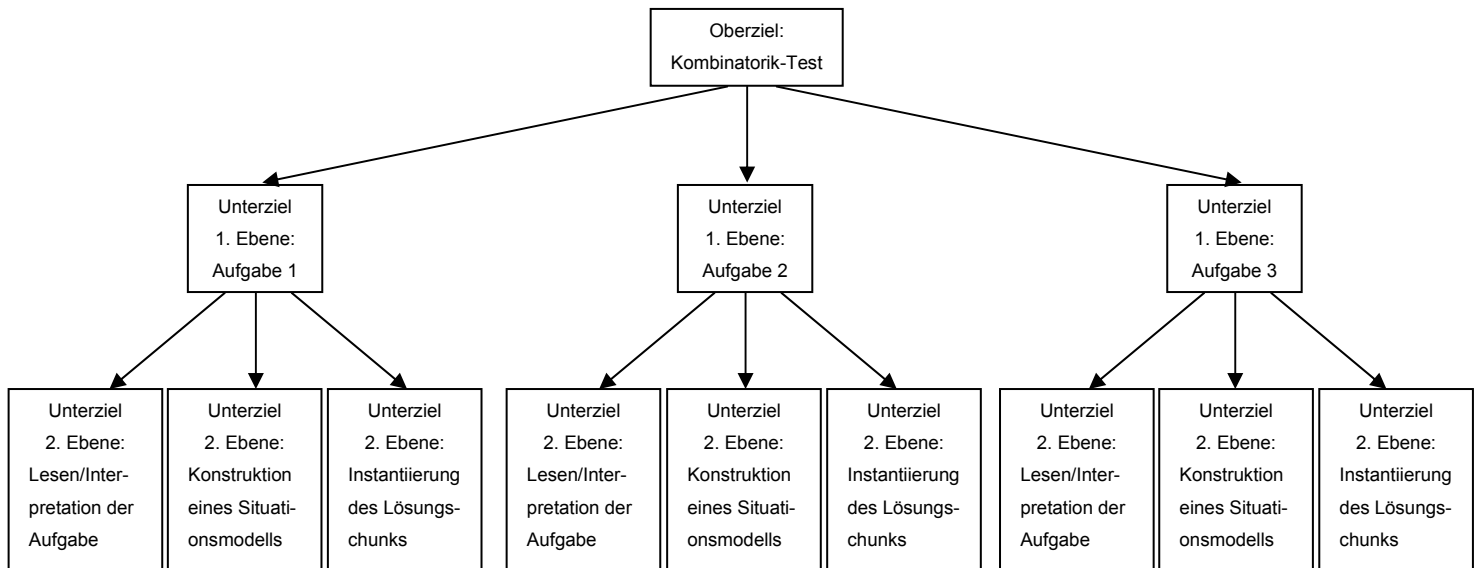


Abbildung 13: Teilzielstruktur der Situationsmodell-Strategie

6.3 Komponenten der Situationsmodell-Strategie

Die oben angeführte Zielstruktur dient als Grundlage der Modellierung der Situationsmodell-Strategie in ACT-R. Zunächst wird also – ebenso wie bei der Schlüsselwort-Strategie – die zu bearbeitende Klausuraufgabe eingesehen. Allerdings werden bei der Situationsmodell-Strategie während des Leseprozesses die relevanten Informationen über die Aufgabe extrahiert und in komprimierter Form repräsentiert. Aus diesen Repräsentationen wird nach Abschluss des Leseprozesses ein Situationsmodell generiert, welches die wichtigsten Informationen der Aufgabe zusammenfasst. Auf Grundlage dieses Situationsmodells werden die Ausprägungen der Strukturmerkmale der Aufgabe erschlossen. Dadurch kann der Lösungschunk mit den Ausprägungen der Strukturmerkmale sowie den Spezifikationen der Parameter vervollständigt und letztlich die einer Klausuraufgabe zu Grunde liegende Aufgabenkategorie inferiert werden.

Im Folgenden werden nun die verschiedenen Komponenten der Situationsmodell-Strategie detailliert dargestellt. Dabei ergeben sich an einigen Stellen Parallelen zur Modellierung der Schlüsselwort-Strategie. In diesem Fall wird unter Verweis auf die Darstellung dieser Modellierung nur eine verkürzte Erläuterung gegeben. Das Modell der Situationsmodell-Strategie ist in Anhang C dargestellt.

(A) Strategiekomponente: Ziel vom *chunk-type* 'combinatoric-test'

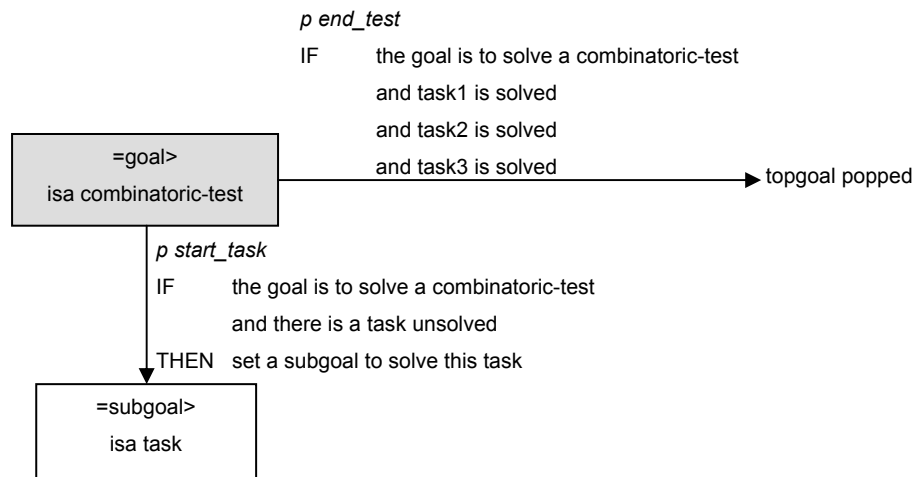


Abbildung 14: Strategiekomponente: Ziel vom *chunk-type* ,combinatoric-test'

Die Bearbeitung eines Ziels vom *chunk-type* ,combinatoric-test' ist vollständig identisch mit der entsprechenden Bearbeitung in der Modellierung der Schlüsselwort-Strategie. Auch hier umfasst das oberste Ziel alle drei zu bearbeitenden Klausuraufgaben; es können ebenfalls zwei Produktionen eingesetzt werden.

(1) *p start_task*

Mit dieser Produktion, die eine der zu lösenden Klausuraufgaben als *subgoal* auf den *goal stack* legt, startet das Modell. Für jede der drei Klausuraufgaben wird der davon ausgehende Zyklus einmal durchlaufen.

(2) *p end_test*

Der *run* des Modells endet mit dem Einsatz dieser Produktion, die feuern kann, wenn alle drei Klausuraufgaben gelöst wurden.

(B) Strategiekomponente: Ziel vom *chunk-type 'task'*

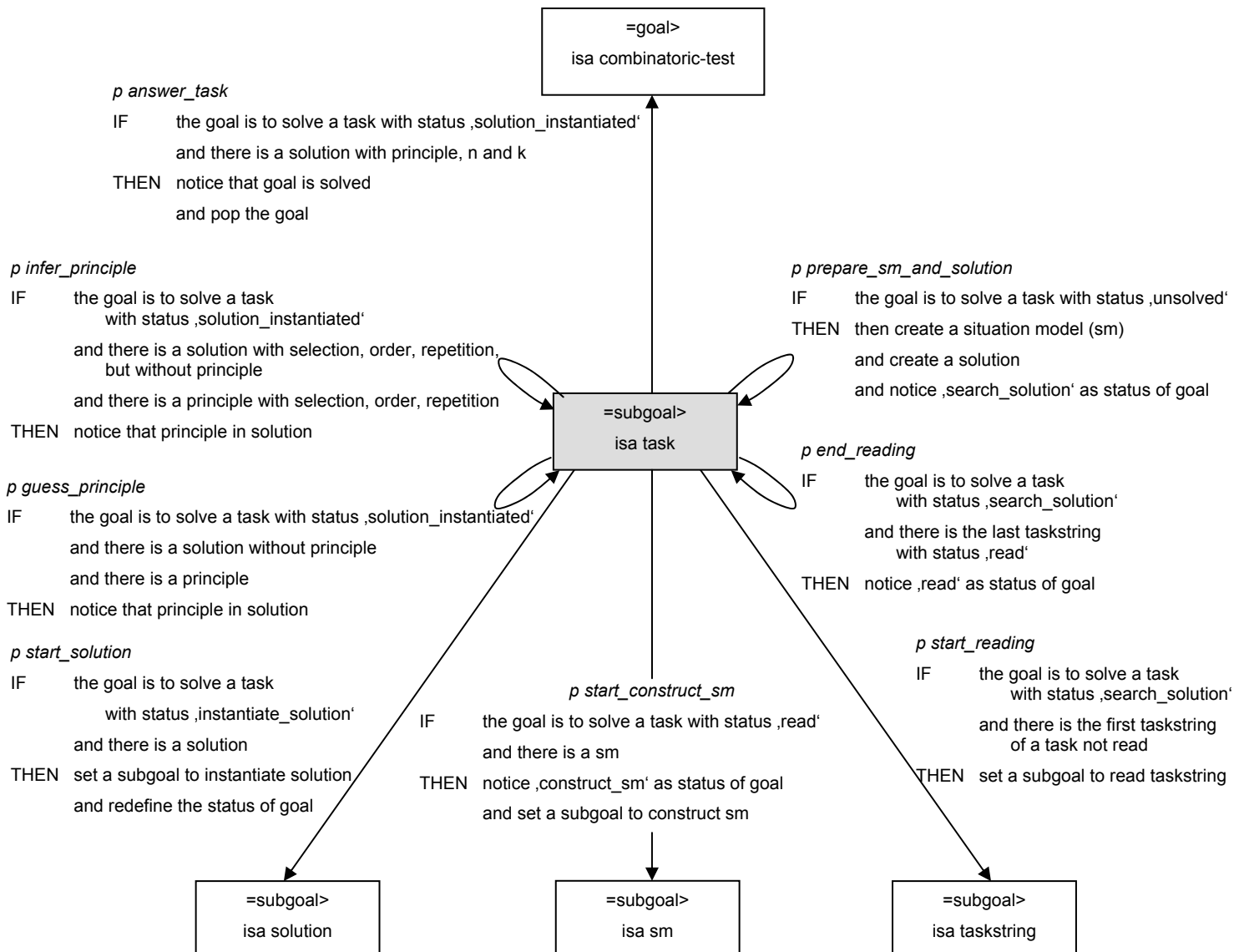


Abbildung 15: Strategiekomponente: Ziel vom *chunk-type ,task'*

Liegt eine Klausuraufgabe (*chunk-type ,task'*) an oberster Stelle des *goal stack*, so stehen zu dessen Bearbeitung die folgenden Produktionen zur Verfügung:

(1) *p prepare_sm_and_solution*

Ebenso wie in der Schlüsselwort-Strategie wird mittels dieser Produktion ein *chunk* für die Lösung generiert. Allerdings kann der *solution-chunk* hier als Kern eines Lösungsschemas interpretiert werden, da nicht nur die Aufgabenkategorie sowie die Parameter darin vermerkt werden, sondern auch die Strukturmerkmale. Diese charakterisieren die jeweilige Aufgabenkategorie und können zu einer prozeduralisierten Aufgabenkategorisierung eingesetzt werden. Außerdem wird ein zweiter *chunk* generiert, welcher die Grundlage eines Situationsmodells bildet. Dieser *chunk* enthält die bereits in Abschnitt 3.3 verdeutlichten Problemelemente, die durch die *slots ,action', ,subject', ,subject_all', ,object', ,object_all', ,mode' und ,special'* repräsentiert werden. Da zunächst noch keine Informationen über diese *slots* vorliegen, sind sie mit

nil belegt. Die vorliegende Produktion kann bei jeder Klausuraufgabe nur einmal – und zwar zu Beginn der Bearbeitung – eingesetzt werden, was durch eine Modifikation der Belegung des *status-slots* der Klausuraufgabe sichergestellt wird.

(2) *p start_reading*

Ebenfalls äquivalent zur Schlüsselwort-Strategie wird durch die vorliegende Produktion zunächst ein Leseprozess initiiert, welcher auch hier nur einmal pro Aufgabe vorgenommen werden kann.

(3) *p end_reading*

Sobald die letzte Textphrase einer Klausuraufgabe „gelesen“ ist, wird die Klausuraufgabe durch die Produktion ‚*end_reading*‘ als „gelesen“ gekennzeichnet, wodurch gewährleistet ist, daß der Leseprozess nicht noch einmal einsetzen kann.

(4) *p start_construct_sm*

Handelt es sich bei dem aktuellen Ziel um eine bereits gelesene Klausuraufgabe, so wird durch die Produktion ‚*start_construct_sm*‘ ein neues *subgoal* auf den *goal stack* gelegt, welches die Konstruktion eines Situationsmodells steuert. Über den Mechanismus des *goal passing* (vgl. Anhang A) ist sichergestellt, dass in der Klausuraufgabe vermerkt wird, sobald die Konstruktion des Situationsmodells beendet ist.

(5) *p start_solution*

Ist in der Klausuraufgabe vermerkt, dass ein Situationsmodell konstruiert wurde, so wird durch Einsatz vorliegender Produktion der Lösungschunk der Klausuraufgabe als neues *subgoal* auf den *goal stack* gelegt. Wenn die *slots* dieses Lösungschunks belegt sind, wird dies über den *goal passing*-Mechanismus in der Klausuraufgabe vermerkt.

(6) *p infer_principle*

Ausgehend von den Strukturmerkmalen, die an den Lösungschunk einer Aufgabe gebunden sind, wird die Aufgabenkategorie der Klausuraufgabe inferiert und diese ebenfalls im Lösungsschema vermerkt.

(7) *p guess_principle*

Alternativ zu der Produktion ‚*infer_principle*‘ kann auch die Produktion ‚*guess_principle*‘ feuern, um die Aufgabenkategorie der Klausuraufgabe zu „raten“, d.h. die im Lösungschunk gebundenen Strukturmerkmale werden dabei nicht berücksichtigt. Die Erfolgsquote dieses Ratevorganges ist – im Vergleich zu der wissensbasierten Vorgehensweise der ‚*infer_principle*‘-Produktion – niedriger anzusetzen. Daher wird die Produktion nur dann eingesetzt, wenn es keine Aufgabenkategorie gibt, welche durch die inferierten Strukturmerkmale charakterisiert wird. Der Einsatz der Produktion setzt damit eine fehlerhafte Identifikation der Strukturmerkmale voraus.

(8) *p answer_task*

Ebenso wie in der Modellierung der Schlüsselwort-Strategie wird eine Klausuraufgabe als gelöst betrachtet, wenn in dem entsprechenden Lösungschunk Wissen um die zu Grunde liegende Aufgabenkategorie vorliegt sowie die Parameter *n* und *k* spezifiziert sind. Damit ist die Bearbeitung der Klausuraufgabe erfolgreich beendet, und die Aufgabe wird vom *goal stack* entfernt.

(C) Strategiekomponente: Ziel vom *chunk-type* 'taskstring'

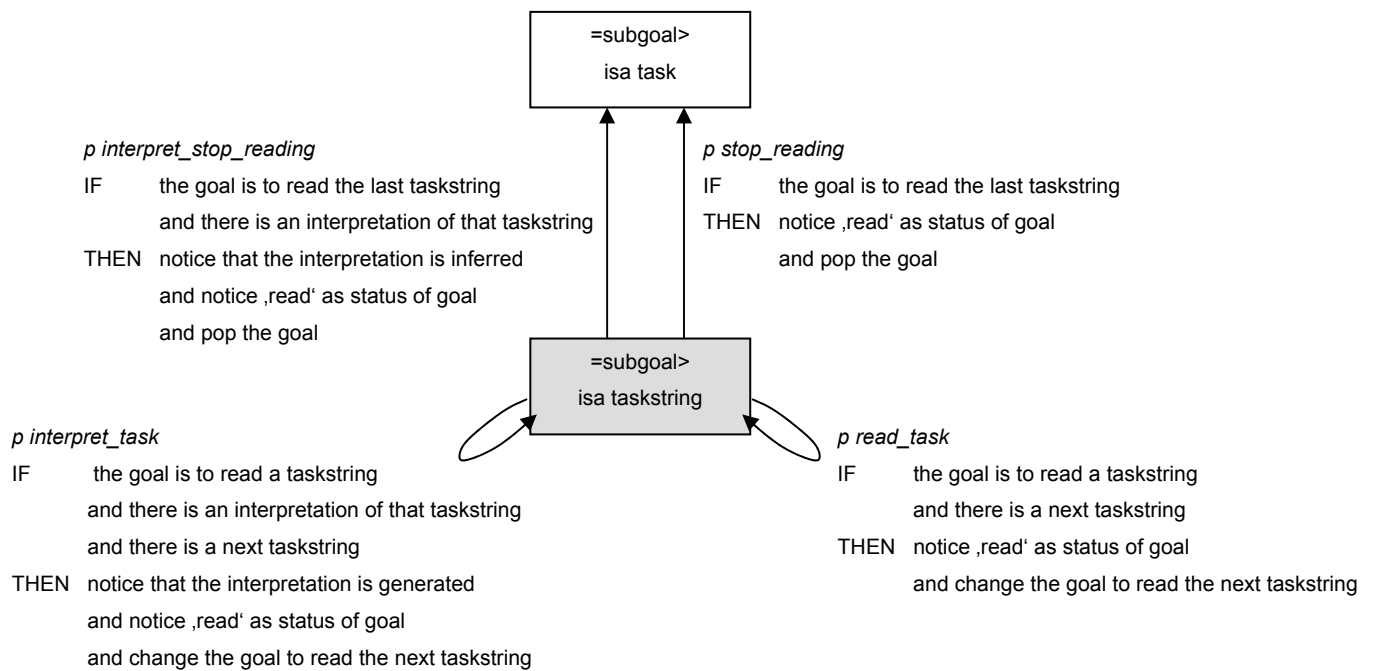


Abbildung 16: Strategiekomponente: Ziel vom *chunk-type* ,taskstring'

Wird durch die Produktion ,start_reading' der Leseprozess initiiert, wird die erste Textphrase der zu bearbeitenden Klausuraufgabe als neues *subgoal* auf den *goal stack* gelegt. Bei einem Ziel des *chunk-type* ,taskstring' können die folgenden Produktionen eingesetzt werden:

(1) *p read_task*

Ebenso wie in der Modellierung der Schlüsselwort-Strategie kann hier eine einfache Leseproduktion feuern, welche lediglich die aktuelle Textphrase der Klausuraufgabe als „gelesen“ kennzeichnet und durch die nachfolgende Textphrase ersetzt.

(2) *p stop_reading*

Liegt auf dem *goal stack* die letzte Textphrase einer Klausuraufgabe, so kann die vorliegende Produktion – ebenfalls eine einfache Leseproduktion – feuern. Als Ergebnis des Einsatzes wird auch hier die Textphrase als „gelesen“ gekennzeichnet, dann aber ersatzlos vom *goal stack* entfernt. Damit ist das Ziel, das auf dem *goal stack* an nächster Position liegt (also die aktuelle Klausuraufgabe vom *chunk-type* ,task') wieder verarbeitungssteuernd.

(3) *p interpret_task*

Alternativ zu der einfachen Leseproduktion ,read_task' kann die vorliegende Produktion eingesetzt werden, wenn aus dem Gedächtnis eine Interpretation der aktuellen Textphrase abgerufen werden kann. Eine solche Interpretation zeichnet sich dadurch aus, dass sie die wesentlichen Inhalte der Textphrase in einer komprimierten Form darstellt. Diese Interpretation wird beim Einsatz der ,interpret_task'-Produktion als „inferiert“ gekennzeichnet und steht damit für spätere Bearbeitungen zur Verfügung(vgl. Strategiekomponente: Ziel vom *chunk-type* ,sub-

goal sm). Außerdem wird auch hier die aktuelle Textphrase als „gelesen“ gekennzeichnet und auf dem *goal stack* durch die nachfolgende Textphrase der Klausuraufgabe ersetzt.

(4) *p interpret_stop_reading*

Liegt die letzte Textphrase einer Klausuraufgabe an oberster Stelle des *goal stack*, so kann alternativ zu der *,stop_reading*-Produktion auch die Produktion *,interpret_stop_reading* eingesetzt werden, wenn eine Interpretation dieser Textphrase abgerufen werden kann. Diese Interpretation wird dann als „generiert“, die Textphrase als „gelesen“ gekennzeichnet. Ebenso wie beim Einsatz der *,stop_reading*-Produktion wird die Textphrase ersatzlos vom *goal stack* genommen, so dass auch hier die Klausuraufgabe wieder verarbeitungssteuernd wird.

(D) Strategiekomponente: Ziel vom *chunk-type 'sm'*

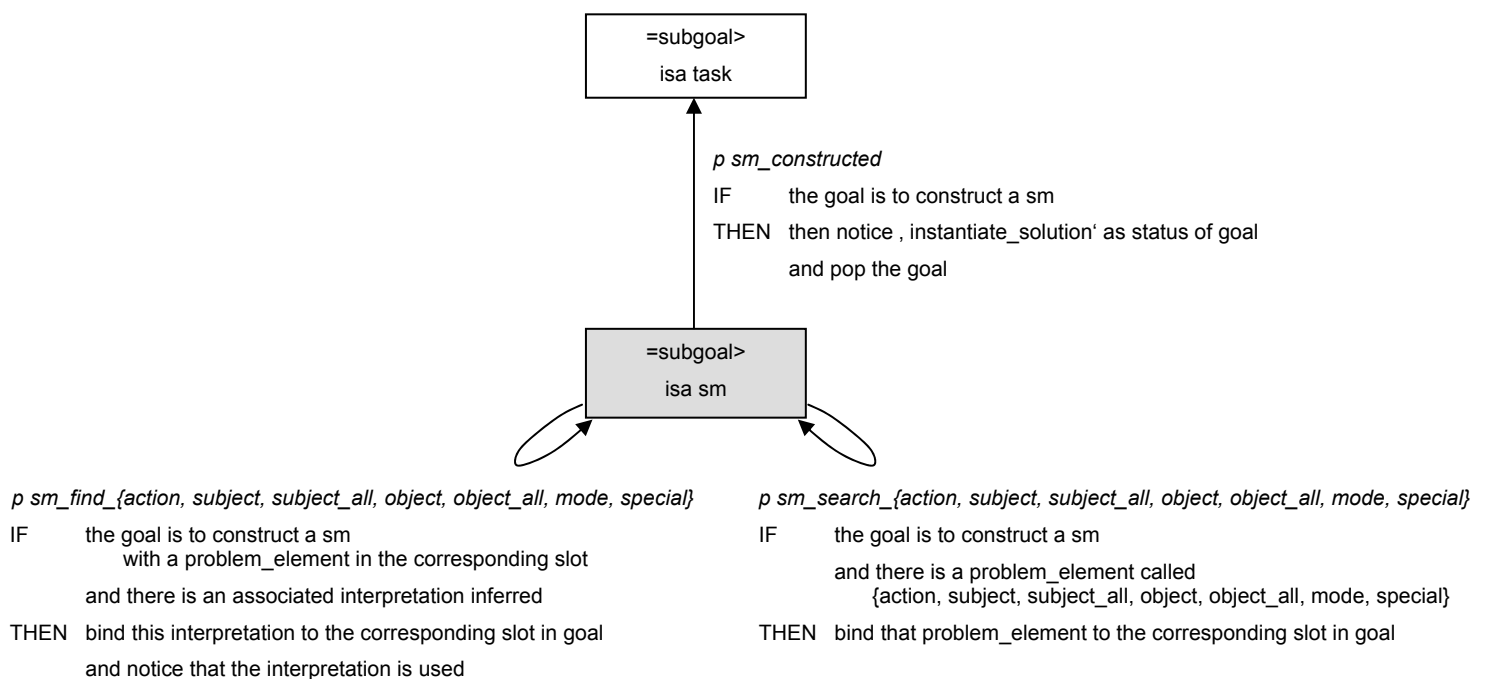


Abbildung 17: Strategiekomponente: Ziel vom *chunk-type ,sm'*

Nach Abschluss des Leseprozesses wird durch die Produktion *,start_construct_sm'* das Ziel, ein Situationsmodell zu konstruieren (*chunk-type ,sm'*) als neues *subgoal* auf den *goal stack* gelegt. Dieses Ziel kann durch nachfolgende Produktionen bearbeitet werden. Dabei werden die verschiedenen Elemente des Situationsmodells durch einen gezielten Suchprozeß instantiiert.

(1-7) *p sm_search_{action, subject, subject_all, object, object_all, mode, special}*⁴

Die Suche nach einem Element des Situationsmodells wird dadurch initiiert, daß in dem Situationsmodellchunk in den jeweiligen *slot* (*,action'*, *,subject'*, *,subject_all'*, *,object'*, *,object_all'*, *,mode'* oder *,special'*) der vorliegende Eintrag *,nil* durch einen *chunk* ersetzt wird, der das zugehörige Problemelement (also entsprechend *,action'*, *,subject'*, *,subject_all'*, *,object'*, *,object_all'*, *,mode'* oder *,special'*) repräsentiert. Dieser *chunk* fungiert in seiner Rolle als *slot-filler*

⁴ Diese Darstellung steht für 7 verschiedene Produktionen, die jeweils eines der Elemente der geschweiften Klammer enthalten.

des *goal-chunks* als Aktivationsquelle und wird benötigt, um seinerseits Interpretationschunks, die aus Textphrasen inferiert wurden, zu identifizieren.

(8-14) $p\ sm_find_{\{action, subject, subject_all, object, object_all, mode, special\}}^4$

Ausgehend von dem jeweiligen Problemelement können assoziierte Interpretationschunks aktiviert und abgerufen werden, in denen Informationen aus der Klausuraufgabe kompakt repräsentiert sind. Diese Interpretationschunks werden als „verwendet“ gekennzeichnet und an den jeweiligen *slot* des Situationsmodells (also *,action‘*, *,subject‘*, *,subject_all‘*, *,object‘*, *,object_all‘*, *,mode‘* oder *,special‘*) gebunden.

(15) $p\ sm_constructed$

Durch die vorliegende Produktion kann der Konstruktionsprozeß des Situationsmodells beendet werden. Durch entsprechende Modifikation des *status-slots* des Situationsmodells ist sichergestellt, dass der Konstruktionsprozeß nur einmal für jede Klausuraufgabe vorgenommen wird.

(E) Strategiekomponente: Ziel vom *chunk-type ‘solution’*

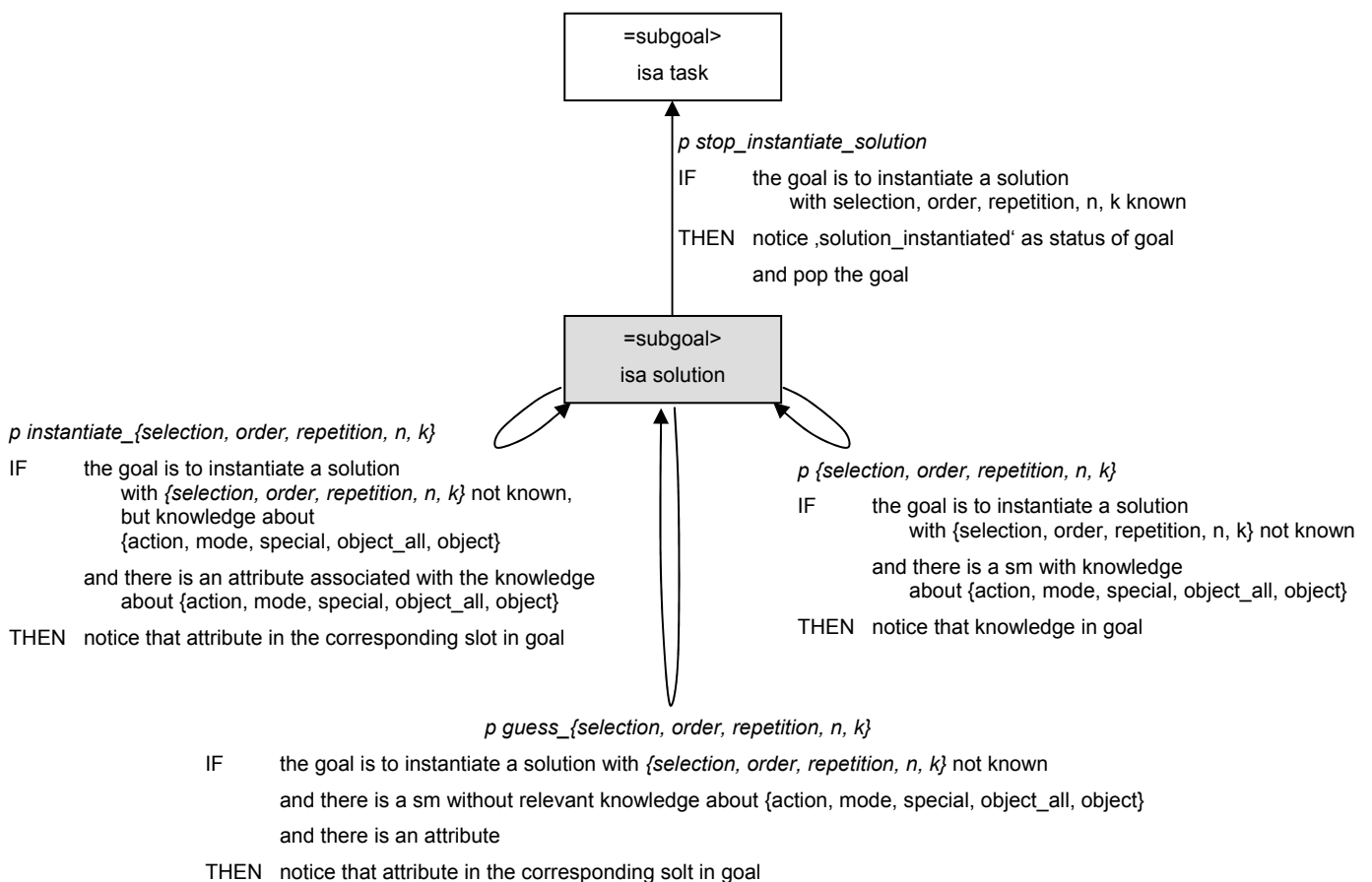


Abbildung 18: Strategiekomponente: Ziel vom *chunk-type ‘solution’*

Wenn der Leseprozess beendet und auch die Konstruktion des Situationsmodells für die zu bearbeitende Klausuraufgabe abgeschlossen ist, wird durch die Produktion *,start_solution‘* das Ziel gesetzt, den Lösungschunk der Klausuraufgabe zu vervollständigen. Dieser Lösungschunk enthält *slots* für die strukturellen Aufgabenmerkmale sowie einen *base-slot*, in den im Verlauf der Bearbeitung Informatio-

nen aus dem Situationsmodell eingetragen werden können (vgl. die Produktionen *{selection, order, repetition, n, k}*). Für die Vervollständigung des Lösungschunks (*chunk-type solution*) stehen die folgenden Produktionen zur Verfügung. Diese resultieren darin, dass in dem generierten Situationsmodell gezielt nach den verschiedenen Elementen des Lösungschunks gesucht wird.

(1-5) *p {selection, order, repetition, n, k}*⁵

Liegt im Lösungschunk noch kein Wissen zu einem der Strukturmerkmale (*selection*, *order*, *repetition*) oder zu einem der Parameter (*n*, *k*) vor und ist der *base-slot* noch nicht belegt, kann eine der hier vorliegenden Produktionen eingesetzt werden, um die gezielte Suche danach zu initiieren. Dabei muss gleichzeitig im Situationsmodell Wissen um ein entsprechendes Problemelement (*action*, *subject*, *subject_all*, *object*, *object_all*, *mode* oder *special*) abrufbar sein. Dieses Wissen wird im Aktionsteil der Produktion in den *base-slot* des Lösungschunks übertragen, so dass es als *slot-filler* des *goal-chunks* Aktivationsquelle sein kann.

(6-10) *p instantiate_{selection, order, repetition, n, k}*⁵

Eine dieser Produktionen kann feuern, wenn zwar kein Wissen zu dem gesuchten Strukturmerkmal bzw. Parameter im Lösungschunk vermerkt ist, aber ein entsprechendes Problemelement aus dem Situationsmodell an den *base-slot* gebunden ist. Ausgehend von diesem Problemelement als Aktivationsquelle wird die Ausprägung des gesuchten Strukturmerkmals bzw. die gesuchte Parameterassoziation assoziiert, welche dann im Lösungschunk vermerkt wird.

(11-15) *p guess_{selection, order, repetition, n, k}*⁵

Fehlt zu einem bestimmten Strukturmerkmal oder Parameter das entsprechende Problemelement im Situationsmodell, so kann auf das Strukturmerkmal oder den Parameter nicht in oben ausgeführtem Sinn geschlossen werden. Statt dessen kann dann eine der hier vorliegenden Produktionen eingesetzt werden, welche jeweils ein Strukturmerkmal bzw. einen Parameter raten. Die Ausprägung des Strukturmerkmals bzw. die Spezifikation des Parameters wird im Aktionsteil der jeweiligen Produktion in das Lösungsschema eingetragen.

(16) *p stop_instantiate_solution*

Entweder durch oben beschriebenen Assoziationsprozess oder alternativ durch den Ratevorgang liegt schließlich zu jedem der Strukturmerkmale und zu jedem Parameter Wissen im Lösungschunk vor. Daher kann die Produktion *stop_instantiate_solution* feuern, welche den Lösungschunk als „vollständig“ kennzeichnet und vom *goal stack* entfernt. Damit wird wieder die Klausuraufgabe verarbeitungssteuernd, da diese an nächster Position auf dem *goal stack* abgelegt ist.

⁵ Diese Darstellung steht für 5 verschiedene Produktionen, die jeweils eines der Elemente der geschweiften Klammer enthalten.

6.4 Darstellung eines *run* des Modells der Situationsmodell-Strategie

Mit dem Start des Modells liegt ein Ziel auf dem *goal stack*, das die drei zu bearbeitenden Klausuraufgaben beinhaltet. Der nachfolgend beschriebene Lösungsprozess wird für jede dieser Aufgaben einmal durchlaufen.

Zunächst wird eine Klausuraufgabe, zu der noch keine Lösung vorliegt, als neues *subgoal* auf den *goal stack* gelegt. Die Bearbeitung dieser Klausuraufgabe lässt sich in drei Komponenten zergliedern: (1) den Lese- bzw. Interpretationsprozess, (2) die Konstruktion des Situationsmodells sowie (3) die Vervollständigung des Lösungschunks.

Als erstes findet der Lese- bzw. Interpretationsprozess statt, der dadurch initiiert wird, dass die erste Textphrase einer Klausuraufgabe als neues *subgoal* auf den *goal stack* gelegt wird. Für die Bearbeitung dieser wie für alle nachfolgenden Textphrasen stehen zwei konkurrierende Produktionen zur Verfügung. Eine Produktion (*,read_task'*) simuliert einen einfachen Leseprozess, bei dem keine tiefere Verarbeitung der gelesenen Information erfolgt. Damit ist die Information für die weitere Bearbeitung der Klausuraufgabe nicht zugänglich. Die andere Produktion (*,interpret_task'*) liest die Textphrase ein und resultiert im Inferieren einer Textinterpretation. Diese Interpretation repräsentiert die wichtigsten Informationen aus der Textphrase in komprimierter Form und kann später bei der Konstruktion des Situationsmodells genutzt werden.

Beispiel: Textphrase → (Der König stellt für das Turnier 12 Pferde)
Interpretation → (12 Pferde)

Auf die dargestellte Weise werden alle Textphrasen eingelesen und evtl. zusätzlich interpretiert, wobei dieser Vorgang nur einmal pro Textphrase durchlaufen wird. Mit dem Abschluß des Lese- bzw. Interpretationsprozess liegt die Klausuraufgabe wieder an oberster Position des *goal stack*. Auf diese (jetzt als gelesen gekennzeichnete) Klausuraufgabe kann eine Produktion zugreifen, welche die Konstruktion eines Situationsmodells als neues *subgoal* initiiert.

Die Konstruktion des Situationsmodells erfolgt als gezielte Suche nach einzelnen Problemelementen, welche die in der Klausuraufgabe vorliegende Situation charakterisieren. Die gezielte Vorgehensweise wird für jedes Element durch zwei aufeinanderfolgende Produktionen realisiert. Dabei wird in einem ersten Schritt der *slot* des Situationsmodells, in dem ein bestimmtes Element eingetragen werden soll, modifiziert. Statt der Startbelegung *,nil'* wird an den *slot* ein *chunk* gebunden, der das jeweilige Problemelement repräsentiert (z.B. *,object_all'*). Dadurch kann sichergestellt werden, dass in einem zweiten Schritt nach Informationen zu eben diesem Element gesucht wird. Bei diesem Suchvorgang wird versucht, eine Textinterpretation zu finden, die aus der Klausuraufgabe inferiert wurde und die mit dem entsprechenden Problemelement assoziiert ist. Diese Interpretation wird dann in dem Lösungschunk vermerkt.

Beispiel: gesuchtes Element → object_all (= Gesamtmenge der Objekte der Situation)
assozierte Interpretation → (12 Pferde)

Wurden zu allen Elementen des Situationsmodells Informationen gefunden oder sind keine weiteren Informationen mehr verfügbar, wird das *subgoal*, ein Situationsmodell zu konstruieren, wieder vom *goal stack* entfernt. In der entsprechenden Klausuraufgabe, die nun wieder an oberster Position des *goal stack* liegt, wird der Abschluss der Konstruktion des Situationsmodells vermerkt. Dadurch wird gewährleistet, dass der beschriebene Konstruktionsprozess – ebenso wie zuvor der Lese- bzw. Interpretationsprozess – nur einmal pro Klausuraufgabe durchlaufen wird.

Wenn zu einer Klausuraufgabe ein Situationsmodell konstruiert wurde, kann mit der Vervollständigung des Lösungschunks begonnen werden. Dies wird dadurch initiiert, dass ein entsprechender Lösungschunk als neues *subgoal* auf den *goal stack* gelegt wird. Auch die Vervollständigung des Lösungschunks erfolgt, indem gezielt nach Elementen, hier nach den Strukturmerkmalen sowie den Parameterspezifikationen, gesucht wird. Auch dieser Vorgang wird durch zwei aufeinander folgende Produktionen pro gesuchter Ausprägung bzw. Spezifikation realisiert. Dabei wird in einem ersten Schritt Information aus dem Situationsmodell abgerufen, welche über die gesuchte Ausprägung bzw. Spezifikation Aufschluss geben kann. Diese Information wird in einem speziellen *base-slot* des Lösungschunks vermerkt. In einem zweiten Schritt wird ausgehend von dieser Information die assoziierte Ausprägung des Strukturmerkmals bzw. die Spezifikation des Parameters abgerufen und in das Lösungsschema eingetragen.

Beispiel: gesuchtes Element des Lösungschunks → n (= Gesamtmenge der Objekte, aus denen ausgewählt wird)
assoziertes Element des Situationsmodells → (12 Pferde) (=object_all)

Kann zu einem *slot* des Lösungschunks keine passende Information im Situationsmodell gefunden werden, so wird die Ausprägung des Strukturmerkmals bzw. die Parameterspezifikation geraten. Sobald Wissen um alle drei Strukturmerkmale (*'selection'*, *'order'*, *'repetition'*) sowie um die Parameter (*'n'*, *'k'*) im Lösungsschema vorliegt, ist dessen Vervollständigung abgeschlossen; das *subgoal* wird vom *goal stack* entfernt. Ausgehend von den Informationen zu den Strukturmerkmalen wird schließlich die Aufgabenkategorie inferiert, welche der Klausuraufgabe zu Grunde liegt. Sind die Informationen über Ausprägungen der Strukturmerkmale fehlerhaft, so kann es vorkommen, dass es keine Aufgabenkategorie gibt, die durch die angegebenen Strukturmerkmale charakterisiert wird. In diesem Fall wird die Aufgabenkategorie geraten.

Beendet wird die Bearbeitung einer Klausuraufgabe, wenn in dem entsprechenden Lösungschunk sowohl Wissen um die Aufgabenkategorie als auch um die Parameter *n* und *k* enthalten ist. Die Aufgabe wird damit als gelöst gekennzeichnet und vom *goal stack* entfernt. Damit kann die nächste Klausuraufgabe, zu der noch keine Lösung vorliegt, bearbeitet werden. Dazu wird diese Aufgabe als neues *subgoal* auf den *goal stack* gelegt und der hier dargestellte Prozess kann wieder einsetzen. Wurden auf diese Weise alle drei zu bearbeitenden Klausuraufgaben gelöst, wird das oberste Ziel, die Bear-

beitung des Kombinatorik-Tests, vom *goal stack* entfernt. Der *run* des Modells ist damit erfolgreich beendet.

7. Zusammenfassender Vergleich der Modelle und Implikationen

In den vorangegangenen Abschnitten wurden mit der Schlüsselwort-Strategie und der Situationsmodell-Strategie zwei mögliche Bearbeitungsstrategien für Textaufgaben aus der Kombinatorik vorgestellt und auf der Ebene lauffähiger ACT-R-Modelle präzisiert. Kognitive Aufgabenanalysen auf der Basis der ACT-R-Architektur sind mit einer Reihe von Vorteilen verbunden, die in diesem Abschnitt noch einmal zusammenfassend skizziert werden sollen. Insbesondere erlauben es derartige Modelle,

- die Ressourcenanforderungen der modellierten Bearbeitungsstrategien zu explizieren,
- Simulationsdaten für verschiedene Bearbeitungsstrategien zu generieren und sie miteinander sowie mit empirisch erhobenen Daten zu vergleichen,
- neue empirische Vorhersagen abzuleiten, die experimentell überprüft werden können, sowie
- Implikationen für den Aufbau weiterführender Modelle abzuleiten, z.B. für Lernmodelle oder Strategiewahlmodelle.

7.1 Ressourcenanforderungen der Schlüsselwort-Strategie und der Situationsmodell-Strategie

Lauffähige ACT-R-Modelle von Bearbeitungsstrategien können daraufhin betrachtet werden, welche Ressourcenanforderungen mit ihnen verbunden sind.

- Erstens können die *Wissensvoraussetzungen* expliziert werden, die erfüllt sein müssen, um eine Bearbeitungsstrategie erfolgreich auszuführen.
- Zweitens kann der gesamte *Zeitbedarf* einer Bearbeitungsstrategie abgeschätzt werden, der von der Anzahl auszuführender Produktion sowie deren jeweiligem Zeitbedarf (für Mustervergleich und Produktionsausführung) abhängt.
- Drittens kann die mit einer Bearbeitungsstrategie einher gehende *Arbeitsgedächtnisbelastung* abgeschätzt werden, indem geprüft wird, wie viele Wissens Elemente gleichzeitig im deklarativen Gedächtnis aktiv gehalten werden müssen, um eine Bearbeitungsstrategie auszuführen.

Wissensanforderungen der Schlüsselwort-Strategie: Die Vorgehensweise bei dieser Strategie beruht vor allem auf Assoziationen zwischen Schlüsselwörtern und Aufgabenkategorien, die auf Grundlage von Lernbeispielen aufgebaut wurden. Das Vorwissen, das zum Einsatz der Schlüsselwort-Strategie erforderlich ist, umfasst also

- Repräsentationen der benötigten Aufgabenkategorien und der Parameter,
- Repräsentationen von Schlüsselwörtern für die verschiedenen Aufgabenkategorien sowie für die Parameter,

- assoziative Verbindungen, die die Beziehungen zwischen Schlüsselwörtern und Aufgabenkategorien bzw. Parametern repräsentieren, sowie
- assoziative Verbindungen, die es erlauben, dass Textphrasen von Klausuraufgaben zur Aktivierung von Schlüsselwörtern führen.

Wissensanforderungen der Situationsmodell-Strategie: Die Vorgehensweise bei dieser Strategie beruht vor allem auf der Generierung eines komprimierten Situationsmodells und auf Wissen über strukturelle Aufgabenmerkmale, die ebenfalls auf Grundlage von Lernbeispielen aufgebaut wurden. Das Vorwissen, das zum Einsatz der Schlüsselwort-Strategie erforderlich ist, umfasst also

- Wissen darüber, welche Informationen in einem Situationsmodell für eine Kombinatorik-Textaufgabe enthalten sein müssen,
- Wissen darüber, wie diese Informationen aus Aufgabentexten extrahiert werden können,
- Repräsentationen der benötigten Aufgabenkategorien und der Parameter,
- Repräsentationen der für die verschiedenen Aufgabenkategorien charakteristischen strukturellen Aufgabenmerkmale,
- Wissen darüber, wie Informationen aus dem Situationsmodell genutzt werden können, um die Ausprägung struktureller Aufgabenmerkmale zu inferieren.

Vergleich der Wissensanforderungen der beiden Strategien: Wie die Aufzählungen der Wissensanforderungen der beiden Strategien zeigen, setzt die Situationsmodell-Strategie umfangreichere und elaboriertere Wissensbestände voraus als die Schlüsselwort-Strategie. Die Schlüsselwort-Strategie beruht vorwiegend auf einfachen Assoziationen zwischen Aufgabenkategorien bzw. Parametern und Schlüsselwörtern, ohne dass ein tieferes Verständnis von Aufgabentexten und Aufgabenkategorien erforderlich wäre. Daher sollte diese Strategie besonders fehleranfällig für irreführende Assoziationen sein. Demgegenüber setzt die Situationsmodell-Strategie tieferes Verständnis und abstrakteres Wissen voraus. Zum einen ist erforderlich, dass Aufgabentexte zumindest soweit verstanden werden, dass ein adäquates Situationsmodell generiert werden kann. Zum anderen wird vorausgesetzt, dass auf der Grundlage von Situationsmodellen strukturelle Aufgabenmerkmale inferiert werden können und dass Aufgaben auf Grund ihrer Ausprägung auf diesen Merkmalen kategorisiert werden können. Es kann daher angenommen werden, dass die Situationsmodell-Strategie im Vergleich zur Schlüsselwort-Strategie weniger fehleranfällig ist.

Zeitbedarf und Arbeitsgedächtnisanforderungen: Ein Vergleich der beiden Bearbeitungsstrategien mit Bezug auf ihren Zeitbedarf und ihre Arbeitsgedächtnisanforderungen zeigt deutlich, dass die Situationsmodell-Strategie aufwändiger ist als die Schlüsselwort-Strategie:

- Beide Strategien umfassen 5 Strategiekomponenten, wobei sich diese im Fall der Situationsmodell-Strategie aus 47 Produktionsregeln zusammensetzen, wohingegen die Schlüsselwort-Strategie lediglich auf 19 Produktionsregeln beruht.
- Zusätzlich sind die Teilziele der Situationsmodell-Strategie komplexer als diejenigen der Schlüsselwort-Strategie, insofern die entsprechenden *goal-chunks* eine größere Anzahl von *slots* beinhalten. Daraus resultiert ceteris paribus eine breitere, aber auch flachere Aktivationsverteilung im

deklarativen Gedächtnis. Damit steigt die Zeit für die erforderlichen Mustervergleichsprozesse bei der Instantiierung von Produktionen, da diese vom Aktivationsniveau der benötigten deklarativen Wissenseinheiten abhängt.

- Die Komplexität der *goal-chunks* bei der Situationsmodell-Strategie ist auch der Grund dafür, dass eine Verringerung der Quellaktivierung (Arbeitsgedächtniskapazität) bei dieser Strategie zu stärkeren Leistungseinbußen führen würde als bei der Schlüsselwort-Strategie. Da die Aktivationsverteilung bei der Situationsmodell-Strategie flacher ist als bei der Schlüsselwort-Strategie, würde ein Absinken der Quellaktivierung bei ersterer Strategie eher dazu führen, dass benötigte deklarative Einheiten im Mustervergleichsprozess nicht mehr abgerufen werden können als bei letzterer Strategie.

Zusammenfassend kann also auf der Grundlage der vorliegenden ACT-R-Modelle im Detail dafür argumentiert werden, dass die Situationsmodell-Strategie weniger fehleranfällig ist als die Schlüsselwort-Strategie, dafür aber mit erheblich höheren Ressourcenanforderungen einhergeht. Diese Anforderungen zeigen sich sowohl bei den Wissensvoraussetzungen als auch hinsichtlich des Zeitbedarfs und der notwendigen Arbeitsgedächtniskapazität.

7.2 Vergleich von Simulationsdaten und empirischen Daten

Um Hinweise auf eingesetzte Bearbeitungsstrategien zu erhalten, haben wir nicht nur quantitative Analysen von Simulationsdaten und Performanzdaten untersuchter Versuchspersonen vorgenommen, sondern auch verbale Protokolle von Versuchspersonen bei der Bearbeitung der drei Klausuraufgaben erhoben. Dabei konnten sowohl Hinweise auf die Schlüsselwort-Strategie als auch auf die Situationsmodell-Strategie gefunden werden.

Folgende Passagen bei der Bearbeitung der Hundeaufgabe weisen z.B. auf den Einsatz der Schlüsselwort-Strategie hin, weil sich hier eine direkte Assoziation zwischen Textphrasen und einer Aufgabenkategorie zeigt, ohne dass vermittelnde Verstehensprozesse auf der Basis einer abstrahierten Aufgabenrepräsentation oder auf der Basis von Wissen über relevante strukturelle Aufgabenmerkmale erkennbar wären. Allerdings fallen der Versuchsperson nach der Assoziation einer Aufgabenkategorie strukturelle Aufgabenmerkmale ein, die diese Kategorie charakterisieren, und sie überprüft deren Zutreffen:

Aufgabentext: Ein Tierheim sucht für 11 Hunde ein Zuhause. 4 der Hunde sind Terrier, die restlichen 7 sind Mischlinge. Es melden sich 2 blonde und 4 schwarzhaarige Kinder, die ein Haustier suchen. Um Streit zu vermeiden, werden die Hunde per Zufall ausgelost. Zuerst ziehen die schwarzhaarigen Kinder jeweils ein Los. Wie berechnet sich die Wahrscheinlichkeit, dass jedes schwarzhaarige Kind einen Terrier bekommt?

Verbales Protokoll: „... vier Terrier, das müßte dann sein, *Permutation mit Wiederholung*, weil es ja vier Terrier gibt und sieben Mischlinge, denk ich mir mal so, aber wir haben ja sechs Plätze, weil es ja sechs Kinder sind, jetzt ist die Frage nach der Wahrscheinlichkeit, die vier Terrier auf die vier schwarzhaarige Kinder zu verteilen, mmmhh, das müßte dann, vier Terrier, *ich würd' sagen das ist Permutation mit Wiederholung*, ach da unten ist auch der Hund, schön, da hat man erst mal einen der vier, dann den nächsten, dann ist es schon mal mit Wiederholung, weil sie gleich sind, *Permutation* weil es auf die Reihenfolge ankommt, oder mmmhhh, zuerst ja doch müßte eigentlich so sein, das müßte dann *Permutation mit Wiederholung* sein ...“

Folgende Passagen bei der Bearbeitung der Ritteraufgabe weisen hingegen auf die Konstruktion eines Situationsmodells hin, weil die Versuchsperson überlegt, ob wirklich alle 12 genannten Ritter von Bedeutung sind oder ob nicht nur die drei Ritter, nach denen am Ende gefragt wird, wichtig für die Bestimmung der Anzahl der relevanten Auswahlen sind:

Aufgabentext: Beim 9. Königsturnier beteiligen sich 10 Ritter. Der König stellt für das Turnier 12 Pferde. Die Ritter beginnen mit verbundenen Augen, sich per Zufall ein Pferd auszuwählen. Zuerst wählt der schwerste Ritter, dann der zweitschwerste, usw. Wie berechnet sich die Wahrscheinlichkeit, dass der schwerste Ritter das größte Pferd, der zweitschwerste das zweitgrößte und der drittschwerste Ritter das drittgrößte Pferd bekommt?

Verbales Protokoll: „...schauen wir noch mal, das sind 12 Pferde und aus denen werden 10 ausgewählt und die Reihenfolge ist irrelevant, deshalb würde ich sagen, daß es 10 aus 12 sind, oder, und dann wie berechnet sich, dass der schwerste und dann der zweitschwerste das - ach so, es geht ja nur um die und um die anderen nicht ...“

Folgende Passagen bei der Bearbeitung der Angleraufgabe zeigen die Nutzung und Überprüfung der strukturellen Aufgabenmerkmale (Wiederholung, Reihenfolge, Auswahl) zur Aufgabenkategorisierung:

Aufgabentext: Der Verein vegetarischer Angler hat 4 Mitglieder. Alle Angler haben sich verpflichtet, gefangene Fische sofort wieder zurück in den Teich zu setzen. Eines Tages angeln die Vereinsmitglieder nacheinander an einem Teich von 8 Quadratmetern, in dem sich 5 Fische befinden: Ein Zander, ein Aal, eine Forelle, ein Hecht und ein Karpfen. Alle Mitglieder angeln in absteigender Altersreihenfolge jeweils einen Fisch. Wie berechnet sich die Wahrscheinlichkeit, dass per Zufall der älteste Angler den Aal geangelt hat und der zweitälteste die Forelle?

Verbales Protokoll: „...ist das nicht noch mal das Gleiche? Man hat fünf Fische, ah ne das ist ja *mit Zurücklegen*, das mit den vier ist ja nicht, und es *kommt auf die Reihenfolge an*, das müsste dann *Variation, Permutation sind alle, Variation nur ein paar*, es muß also

schon mal Variation sein; *Wiederholung, weil es mit Zurücklegen ist*. Denk ich jetzt mal so, also man hat fünf Fische, gefangene sofort wieder zurück in den Teich, ja das ist mit Zurücklegen ... also zwei Leute, zwei Fische, das wäre dann Variation mit Wiederholung n ist gleich 5, weil es fünf Fische sind und k ist gleich 2, weil zwei rausgezogen werden, so ...“

Um die relative Performanz der Schlüsselwort-Strategie und der Situationsmodell-Strategie einzuschätzen, haben wir Simulationsdaten für die beiden Bearbeitungsstrategien generiert und sie miteinander sowie mit empirisch erhobenen Daten verglichen. Zur Generierung der Simulationsdaten wurden für jedes Modell 20 *runs* ausgeführt. Für jeden *run* wurden die Fehler bei der Lösung der drei Klausuraufgaben (Codierung wie in Gerjets, Scheiter & Tack, 2000) sowie die Anzahl der ausgeführten Produktionen (Zyklen) und der gesamte Zeitbedarf des *runs* erfasst. Um eine gewisse Varianz der generierten Daten zu erzeugen, wurden bei beiden Modellen die Parameter *activation noise*, *permanent activation noise* und *expected-gain noise* gegenüber dem *default*-Wert von 0 geringfügig erhöht. Für beide Modelle wurden identische Werte eingestellt (0,1 / 0,1 / 1). Im ACT-R-Modell der Schlüsselwort-Strategie wurde zusätzlich das *base-level-learning* aktiviert und auf den *default*-Wert von 0,5 eingestellt, um die postulierten Prozesse der Aktivationssummation zu ermöglichen. Die Ergebnisse der Simulation sind in Tabelle 1 dargestellt.

Tabelle 1: Simulationsdaten: Mittelwerte (M) und Standardabweichungen (SD) für die ACT-R-Modelle der Schlüsselwort-Strategie und der Situationsmodell-Strategie

	Schlüsselwort-Strategie (n = 20)		Situationsmodell-Strategie (n = 20)	
	M	SD	M	SD
Zeitbedarf insgesamt in sec	91,69	13,17	295,79	12,32
Zyklen insgesamt	106,95	3,93	141,95	6,11
Klausurfehler in %	47,22	17,52	14,44	12,28

Ein quantitativer Vergleich der beiden Strategien ergibt, dass die Situationsmodell-Strategie mit 14,44% Klausurfehlern gegenüber der Schlüsselwort-Strategie mit 47,22% Klausurfehlern auf der einen Seite zu wesentlich besseren Leistungen führt ($t(38) = -6.85$; $p < .001$, zweiseitig). Auf der anderen Seite zeichnet die Schlüsselwort-Strategie sich mit 106,95 Zyklen bzw. mit 91,69 sec durch einen geringeren Zeitbedarf aus als die Situationsmodell-Strategie mit 141,95 Zyklen bzw. 295,79 sec ($t(38) = 21.56$; $p < .001$, zweiseitig bzw. $t(38) = 50.60$; $p < .001$, zweiseitig).

Empirische Daten aus einer vergleichbaren experimentellen Studie (Gerjets, Scheiter und Tack, 2000) sind in Tabelle 2 dargestellt. Versuchspersonen in dieser Studie erhielten als Instruktionsmaterial für jede der sechs Aufgabenkategorien eine abstrakte Darstellung und konnten zusätzlich jeweils ein

einfaches Urnenbeispiel in der HYPERCOMB-Umgebung abrufen. Die verwendeten Urnenbeispiele und Klausuraufgaben waren identisch mit denjenigen, die den Simulationsdaten zu Grunde liegen (vgl. Anhang D). In der experimentellen Studie wurden 40 Versuchspersonen untersucht und anhand der Werte in einem konzeptuellen Vortest per Mediansplit danach aufgeteilt, ob sie über hohes oder niedriges domänenspezifisches Vorwissen verfügen. Dabei wurde vermutet, dass Versuchspersonen mit hohem Vorwissen eher über die Voraussetzungen für die Anwendung der Situationsmodell-Strategie verfügen (Wissen darüber, welche Informationen in einem Situationsmodell für eine Kombinatorik-Textaufgabe enthalten sein müssen, Wissen über die für die verschiedenen Aufgabenkategorien charakteristischen strukturellen Aufgabenmerkmale) als Versuchspersonen mit niedrigem Vorwissen.

Tabelle 2: Empirische Daten: Mittelwerte (M) und Standardabweichungen (SD) für Versuchspersonen mit niedrigem und hohem Vorwissen (aus Gerjets, Scheiter und Tack, 2000)

	niedriges Vorwissen (n = 20)		hohes Vorwissen (n = 20)	
	M	SD	M	SD
Zeit auf Klausurseiten in sec	589,75	153,38	572,10	264,43
Klausurfehler in %	49,17	18,67	32,78	16,31

Es kann hypothetisch angenommen werden, dass die meisten Versuchspersonen mit niedrigem Vorwissen die Schlüsselwort-Strategie eingesetzt haben, während zumindest einige Versuchspersonen mit hohem Vorwissen die Situationsmodell-Strategie verwendet haben könnten. Das empirisch gefundene Datenmuster ist mit dieser Vermutung kompatibel. Versuchspersonen mit niedrigem Vorwissen entsprachen mit 49,17% Klausurfehlern den Simulationsdaten für die Schlüsselwort-Strategie. Auch die Standardabweichung für diese beiden Datensätze stimmen weitgehend überein. Gegenüber den Versuchspersonen mit niedrigem Vorwissen zeigen Versuchspersonen mit hohem Vorwissen mit 32,78% Klausurfehlern wesentlich bessere Bearbeitungsleistungen ($t(38) = -2,96$; $p < .005$, zweiseitig). Dieser Unterschied könnte darauf zurückgehen, dass zumindest ein Teil der Versuchspersonen mit hohem Vorwissen die Situationsmodell-Strategie einsetzt. In der Klausurbearbeitungszeit unterscheiden sich Versuchspersonen mit niedrigem (589,75 sec) und hohem (572, 10 sec) Vorwissen nicht signifikant ($t(38) = -0,26$; $p > .70$, zweiseitig). Die empirisch gefundene Klausurbearbeitungsdauer ist deutlich länger als der in den ACT-R-Modellen geschätzte Zeitbedarf. Dies geht vermutlich auch darauf zurück, dass die empirisch gefundenen Bearbeitungsdauern auch Verarbeitungsprozesse beinhalten, die in den ACT-R-Modellen nicht berücksichtigt werden, wie z.B. Prozesse des Lesens und der Textverarbeitung, der Hypertext-Navigation sowie des Eingebens der Aufgabenlösungen.

7.3 Ableitung neuer empirischer Vorhersagen

Die erstellten Performanzmodelle können nicht nur zur theoretischen Analyse von Ressourcenanforderungen und zum Vergleich mit vorliegenden empirischen Daten herangezogen werden, sondern erlauben auch die Ableitung von empirischen Vorhersagen für neue experimentelle Situationen. Beispielsweise können die beiden folgenden Experimentalanordnungen konstruiert werden, in denen es sinnvoll wäre, die Performanz von ACT-R Modellen mit derjenigen von Versuchspersonen zu vergleichen:

1. Es können unterschiedliche Sets von Klausuraufgaben untersucht werden, bei denen die Schlüsselwörter, die in der Lernphase aus Beispielaufgaben erworben werden konnten, in unterschiedlicher Form in den Klausuraufgaben auftreten (kongruent, inkongruent, indifferent). In der *kongruenten Bedingung* treten die erlernten Schlüsselwörter in den Klausuraufgaben in identischer Form wieder auf, und zwar bei den gleichen Aufgabenkategorien wie in der Lernphase. In der *inkongruenten Bedingung* treten die erlernten Schlüsselwörter in den Klausuraufgaben ebenfalls in identischer Form wieder auf, allerdings im Kontext anderer Aufgabenkategorien als in der Lernphase. In der *indifferenten Bedingung* treten die erlernten Schlüsselwörter in den Klausuraufgaben nicht in identischer Form auf. Für Versuchspersonen mit *hohem Vorwissen*, die überwiegend eine Situationsmodell-Strategie einsetzen sollten, bzw. *niedrigem Vorwissen*, die überwiegend eine Schlüsselwort-Strategie einsetzen sollten, ergeben sich die folgenden Hypothesen:

- Für Versuchspersonen mit hohem Vorwissen (bzw. für das ACT-R-Modell der Situationsmodell-Strategie) sollte die Manipulation der Klausuraufgaben keinen wesentlichen Einfluss auf die Performanz ausüben.
- Für Versuchspersonen mit niedrigem Vorwissen (bzw. für das ACT-R-Modell der Schlüsselwort-Strategie) sollten sich für kongruente Klausuraufgaben bessere Leistungen zeigen als für indifferente, und für indifferente Klausuraufgaben sollten sich bessere Leistungen zeigen als für inkongruente.

2. Es können unterschiedliche Sets von Klausuraufgaben untersucht werden, bei denen die Schlüsselwörter, die in der Lernphase aus Beispielaufgaben erworben werden konnten, jeweils in kongruenter Form auftreten. Die Klausuraufgaben unterscheiden sich jedoch dahingehend, ob ihnen eine konsistente Repräsentation der Problemsituation entnommen werden kann (konsistent, inkonsistent). In der *konsistenten Bedingung* enthält der Aufgabentext eine interpretierbare Beschreibung der Problemsituation. In der *inkonsistenten Bedingung* ist diese Beschreibung uneindeutig, widersprüchlich oder lückenhaft, so dass kein Situationsmodell inferiert werden kann. Für Versuchspersonen mit *hohem Vorwissen*, die überwiegend eine Situationsmodell-Strategie einsetzen sollten, bzw. *niedrigem Vorwissen*, die überwiegend eine Schlüsselwort-Strategie einsetzen sollten, ergeben sich die folgenden Hypothesen:

- Für Versuchspersonen mit hohem Vorwissen (bzw. für das ACT-R-Modell der Situationsmodell-Strategie) sollten sich für konsistente Klausuraufgaben bessere Leistungen zeigen als für inkonsistente.

- Für Versuchspersonen mit niedrigem Vorwissen (bzw. für das ACT-R-Modell der Schlüsselwort-Strategie) sollte die Manipulation der Klausuraufgaben keinen wesentlichen Einfluss auf die Performanz ausüben.

7.4 Implikationen für Lernmodelle und Strategiewahlmodelle

Aufbauend auf die in dieser Arbeit berichtete Modellierungen möglicher Strategien der Bearbeitung von Kombinatorik-Textaufgaben im Sinne einer Aufgabenanalyse sollen weitere Modelle konstruiert werden, um den Prozess des Wissenserwerbs und der Wissensnutzung zu modellieren, wie er empirisch im Rahmen der HYPERCOMB-Experimente untersucht wurde. Dabei sollen zunächst Lernmodelle erzeugt werden, die den Erwerb des Wissens simulieren, das für die Anwendung einer bestimmten Bearbeitungsstrategie erforderlich ist, wobei auch verschiedene Wissenserwerbsstrategien berücksichtigt werden. Im nächsten Schritt stehen dann Strategiewahlmodelle im Vordergrund, die die Wahl einer bestimmten Bearbeitungsstrategie bzw. Wissenserwerbsstrategie zum Gegenstand haben.

Lernmodelle: In Lernmodellen soll elaboriert werden, wie die Wissensvoraussetzungen der Schlüsselwort-Strategie und der Situationsmodell-Strategie mit Hilfe ausgearbeiteter Beispielaufgaben erworben werden können. Für die Schlüsselwort-Strategie betrifft dies vor allem den Erwerb von Schlüsselwörtern und Aufgabenkategorien sowie den Aufbau geeigneter assoziativer Verbindungsstrukturen. Für die Situationsmodell-Strategie ist hingegen das Wissen erforderlich, welche Informationen in einem Situationsmodell für eine Kombinatorik-Textaufgabe enthalten sein müssen und welche strukturellen Aufgabenmerkmale die verschiedenen Aufgabenkategorien charakterisieren. Zum Erwerb dieses Wissens ist der Vergleich strukturgleicher Lernbeispiele und die Entwicklung abstrakter Problemrepräsentationen Voraussetzung. Welche Lernergebnisse in der Wissenserwerbsphase resultieren und welche Problemlösestrategien daher in einer nachfolgenden Wissensanwendungsphase eingesetzt werden können, sollte dabei vor allem von den eingesetzten Strategien bei der Nutzung ausgearbeiteter Beispielaufgaben und anderer Instruktionmaterialien abhängen. Diese Strategien stellen ihrerseits unterschiedliche Anforderungen an die Verfügbarkeit von Ressourcen wie Zeit, Vorwissen und Lernmaterial. Es können vier Vorgehensweisen beim Wissenserwerb aus ausgearbeiteten Beispielen unterschieden werden, die sich jeweils durch eigene Lernmodelle darstellen lassen: Memorieren, Schlüsselwortidentifikation, Elaborieren und Schemaabstraktion.

- Beim *Memorieren* werden Beispielinformationen lediglich gespeichert. Memorieren verlangt wenig vorwissensbasierte Inferenzen, wenig Zeit und ist relativ unabhängig vom Umfang externer Information. Auf der anderen Seite ist diese Strategie anfällig für Einflüsse von Oberflächenmerkmalen und erfordert aufwändige Problemlöseprozesse bei der späteren Nutzung memorierter Beispiele.
- Die *Schlüsselwortidentifikation* beruht auf der Idee einer unvollständigen Beispielrepräsentation, die im Wesentlichen Informationen über Hinweise (Schlüsselwörter) auf die jeweilige Aufgabenkategorie enthält, welche beim Problemlösen zur Aufgabenkategorisierung genutzt werden können. Die Identifikation von Schlüsselwörtern setzt eine Vielzahl von Lernbeispielen voraus, die jedoch

nur oberflächlich verarbeitet werden müssen. Sie ist daher auch mit wenig Vorwissen und Lernzeit einsetzbar, jedoch anfällig für Einflüsse von irreführenden Assoziationen.

- Die *Elaboration* besteht darin, konkrete Beispielinformationen mit abstrakten Situationsrepräsentationen und Strukturmerkmalen zu verknüpfen, um zu einer elaborierten Beispielrepräsentation zu gelangen. Die entsprechenden abstrakten Informationen können in Form bereits vorhandenen Wissens vorliegen (Selbsterklärungsprozesse) oder external präsentiert bzw. neu inferiert werden (erklärungsbasiertes Lernen). Elaborationsstrategien setzen Vorwissen und Zeit voraus, sind hingegen relativ sparsam mit Bezug auf verfügbares Lernmaterial.
- Bei der *Schemaabstraktion* werden mehrere Beispiele hinsichtlich ihrer Gemeinsamkeiten und Unterschiede verglichen um aus übereinstimmenden Merkmalen ein abstraktes Schema zu generieren, das von ideosynkratischen Merkmalen einzelner Beispiele abstrahiert. Diese Strategie ist aufwändig und erfordert viel verfügbare Zeit, vorhandenes Vorwissen und verfügbare externe Informationen. Die resultierenden Problemlöseschemata erlauben jedoch eine hoch effiziente Aufgabebearbeitung.

Strategiewahlmodelle: Strategiewahlmodelle beschreiben die Wahl einer bestimmten Bearbeitungsstrategie bzw. Wissenserwerbsstrategie. Da Strategien als Sequenzen von Teilzielen und Produktionsregeln repräsentiert werden, lassen sich Prozesse der Strategiewahl sowohl auf die Wahl von Teilzielen als auch auf die Wahl entsprechender Produktionsregeln beziehen. Wir unterscheiden dabei zwischen deliberativer und automatischer Strategiewahl.

Eine *deliberative Strategiewahl* beruht auf *metakognitiven Zielen*, die eine explizite Entscheidungssituation zwischen Strategiealternativen repräsentieren. Wenn ein metakognitives Ziel verarbeitungssteuernd wird, ist die Bearbeitung der zuvor aktuellen Aufgabe unterbrochen und es werden nur Regeln angewandt, die sich auf die Auswahl einer Bearbeitungsstrategie beziehen. Deliberative Prozesse der Strategiewahl können überwiegend symbolisch mit Hilfe aufgabenunspezifischer *Interrupt-Produktionen* modelliert werden.

Prozesse der *automatischen Strategiewahl* lassen sich hingegen besser auf der Basis subsymbolischer Prozesse beschreiben. Dabei können sowohl in ACT-R implementierte Mechanismen benutzt werden wie auch spezifische Erweiterungen in Form exekutiver Kontrollproduktionen, die den Zugriff auf die subsymbolische Ebene gestatten. Die von ACT-R bereitgestellten Mechanismen umfassen vor allem Prozesse der Konfliktresolution bei der Entscheidung zwischen verschiedenen Produktionen sowie Mustervergleichsprozesse zur Auswahl deklarativer Wissensinhalte bei der Instantiierung selektierter Produktionen. Notwendige Erweiterungen betreffen vor allem die Manipulation subsymbolischer Parameter durch exekutive Kontrollproduktionen.

Da die hier interessierenden Bearbeitungs- und Wissenserwerbsstrategien unterschiedliche Ressourcenanforderungen stellen, sollte die Auswahl einer Strategie von der vorliegenden Ressourcenkonfiguration abhängig sein. Wir nehmen an, dass auf Ressourcenbeschränkungen mit automatischer Strategiewahl reagiert wird, während ein umfangreiches Ressourcenangebot eher zu deliberativer

Strategiewahl führt. Dabei liegt die Idee zu Grunde, dass die Anzahl einsetzbarer Strategien mit zunehmender Ressourcenverfügbarkeit ansteigt, sodass eine explizite Entscheidung für eine der alternativen Strategien notwendig wird (Gerjets, Scheiter & Tack, 2000).

Die automatische Adaption an knappe Ressourcen kann im Rahmen der ACT-R-Architektur einfach erklärt werden. Eine automatische Anpassung der Produktionsauswahl an die verfügbare Zeit zur Aufgabenbearbeitung wird durch den im System vorgesehenen Mechanismus der Konfliktresolution erreicht. Dieser bewirkt einen Geschwindigkeits-Genauigkeits-Abgleich, bei dem für Aufgaben, die in kurzer Zeit erledigt werden sollen, schnelle aber möglicherweise weniger Erfolg versprechende Bearbeitungsschritte ausgewählt werden, während für Aufgaben mit ausreichender Bearbeitungszeit kostenintensivere Alternativen gewählt werden, die jedoch mit einer hohen Erfolgswahrscheinlichkeit einhergehen. Dieser Konfliktresolutionsprozess schränkt damit die Menge verfügbarer Produktionen ein. Zur Anwendung dieses Mechanismus wird eine zusätzliche exekutive Kontrollproduktion benötigt, die auf Zeitdruckinstruktionen mit einer entsprechenden Absenkung des Systemparameters G reagiert. Analog führt ein Mangel an deklarativem oder prozeduralem Vorwissen oder ein Mangel an verfügbarem Lernmaterial dazu, dass bestimmte Strategien nicht ausgewählt werden können, da im Mustervergleichsprozess die benötigten Informationen zur Instantiierung der entsprechenden Produktionen nicht gebunden werden können.

Deliberative Strategiewahlprozesse sollten hingegen einsetzen, wenn (1) eine hohe Ressourcenverfügbarkeit dazu führt, dass mehrere alternative Strategien einsetzbar sind und wenn (2) subsymbolische Parameter keine hinreichenden Informationen liefern um über die Nützlichkeit dieser alternativen Strategien zu entscheiden. So kann beispielsweise angenommen werden, dass Lerner zunächst kein subsymbolisches Wissen über die Nützlichkeit verschiedener Lernmaterialien besitzen, so dass die Informationsselektion deliberative Strategiewahlen erfordert.

8. Literatur

- Anderson, J. R. (1993a). Problem solving and learning. *American Psychologist*, 48, 35-44.
- Anderson, J. R. (1993b). *Rules of the mind*. Hillsdale, NJ: Erlbaum.
- Anderson, J. R., Fincham, J. M. & Douglass, S. (1997). The role of examples and rules in the acquisition of a cognitive skill. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 23, 932-945.
- Anderson, J. R. & Lebière, C. (1998). *The atomic components of thought*. Hillsdale, NJ: Erlbaum.
- Bernardo, A. B. I. (1994). Problem-specific information and the development of problem-type schemata. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 20, 379-395.
- Carbonell, J. G. (1984). Learning by analogy: Generalizing plans from past experience. In R. S. Michalski, J. G. Carbonell & T. M. Mitchell (Eds.), *Machine Learning* (pp. 137-161). Berlin: Springer.
- Carbonell, J. G. (1986). Derivational analogy: A theory of reconstructive problem solving and expertise acquisition. In R. Michalski, J. G. Carbonell & T. M. Mitchell (Eds.), *Machine Learning: An artificial intelligence approach (Vol. 2)* (pp. 371-392). Los Altos, CA: Morgan Kaufmann.
- Carbonell, J. G. & Veloso, M. (1988). Integrating derivational analogy into a general problem solving architecture. In J.L.Kolodner (Ed.), *Proceedings of the DARPA-workshop on case-based reasoning, Clearwater, Florida* (pp. 104-124). San Mateo, CA: Morgan Kaufmann.
- Chi, M. T. H., Bassok, M., Lewis, M., Reimann, P. & Glaser, R. (1989). Self-explanations: How students study and use examples in learning to solve problems. *Cognitive Science*, 13, 145-182.
- Cooper, G. & Sweller, J. (1987). Effects of schema acquisition and rule automation of mathematical problem-solving transfer. *Journal of Educational Psychology*, 79, 347-362.
- Cummins, D. D. (1992). Role of analogical reasoning in the induction of problem categories. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 18, 1103-1124.
- Dörner, D. (1976). *Problemlösen als Informationsverarbeitung*. Stuttgart: Kohlhammer.
- Gentner, D. (1983). Structure Mapping: A theoretical framework for analogy. *Cognitive Science*, 7, 155-170.
- Gentner, D. & Markman, A. B. (1997). Structure mapping in analogy and similarity. *American Psychologist*, 52, 45-56.
- Gerjets, P., Scheiter, K. & Tack, W.H. (2000). Resource-adaptive selection of strategies in learning from worked-out examples. In L.R. Gleitman & A.K. Joshi (Eds.), *Proceedings of the 22nd Annual Conference of the Cognitive Science Society* (pp. 166-171). Mahwah, NJ: Erlbaum.
- Gerjets, P., Scheiter, K. & Tack, W.H. (2001). Problems of example selection and example processing in hypertext-based learning environments (Arbeitsbericht). Saarbrücken: Universität des Saarlandes.
- Gick, M. L. & Holyoak, K. J. (1980). Analogical problem solving. *Cognitive Psychology*, 12, 306-355.
- Gick, M. L. & Holyoak, K. J. (1983). Schema induction and analogical transfer. *Cognitive Psychology*, 15, 1-38.
- Holyoak, K. J. & Koh, K. (1987). Surface and structural similarity in analogical transfer. *Memory & Cognition*, 15, 332-340.

- Holyoak, K. J., Novick, L. R. & Melz, E. R. (1994). Component processes in analogical transfer: Mapping, pattern completion, and adaptation. In K. J. Holyoak & J. A. Barnden (Eds.), *Analogical Connections* (pp. 113-180). Norwood, NJ: Ablex.
- Kedar-Cabelli, S. (1988). Analogy - from a unified perspective. In D. Helman (Ed.), *Analogical reasoning* (pp. 65-103). Dordrecht: Kluwer.
- Kintsch, W. (1988). The use of knowledge in discourse processing: A construction-integration model. *Psychological Review*, *92*, 163-182.
- Kintsch, W. (1994). Text comprehension, memory, and learning. *American Psychologist*, *49*, 294-303.
- Koedinger, K. R. & Anderson, J. R. (1990). Abstract planning and perceptual chunks: Elements of expertise in geometry. *Cognitive Science*, *14*, 511-550.
- Kolodner, J. (1993). *Case-based reasoning*. San Mateo, CA: Morgan Kaufmann.
- Larkin, J., McDermott, J., Simon, D. & Simon, H. A. (1980). Expert and novice performance in solving physics problems. *Science*, *208*, 1335-1342.
- Lovett, M.C., Reder, L.M. & Lebiere C. (1999). Modeling working memory in a unified architecture: An ACT-R perspective. In A. Miyake & P. Shah (Eds.), *Models of working memory* (pp. 135-182). Cambridge: Cambridge University Press.
- Marshall, S. P. (1995). *Schemas in problem solving*. Cambridge, MA: Cambridge University Press.
- Michener, D. R. (1978). Understanding mathematics. *Cognitive Science*, *2*, 361-383.
- Nathan, M. J., Kintsch, W. & Young, E. (1992). A theory of algebra-word-problem comprehension and its implication for the design of learning environments. *Cognition and Instruction*, *9*, 329-389.
- Nelson, G., Thagard, P. & Hardy, S. (1994). Integration analogy with rules and explanations. In K. J. Holyoak & J. A. Barnden (Eds.), *Analogical connections* (pp. 181-206). Norwood, NJ: Ablex.
- Nesher, P. & Teubal, E. (1975). Verbal cues as an interfering factor in verbal problem solving. *Educational studies in mathematics*, *6*, 41-51.
- Newell, A. & Simon, H. A. (1972). *Human problem solving*. Englewood Cliffs, NJ: Prentice-Hall.
- Novick, L. R. (1988). Analogical transfer: Processes and individual differences. In D. Helman (Ed.), *Analogical reasoning* (pp. 125-145). Dordrecht: Kluwer.
- Pirolli, P. (1999). Cognitive engineering models and cognitive architectures in human-computer interaction. In F. T. Durso, R. S. Nickerson, R. W. Schvaneveldt, S. T. Dumais, D. S. Lindsay & M. T. H. Chi (Eds.), *Handbook of applied cognition* (pp. 443-477). New York: Wiley.
- Quilici, J. L. & Mayer, R. E. (1996). Role of examples in how students learn to categorize statistics word problems. *Journal of Educational Psychology*, *88*, 144-161.
- Reimann, P. & Chi, M. T. H. (1989). Human Expertise. In K. J. Gilhooly (Ed.), *Human and machine problem solving* (pp. 161-191). New York: Plenum.
- Reimann, P. (1997). *Lernprozesse beim Wissenserwerb aus Beispielen*. Bern: Huber.
- Riesbeck, C. K. & Schank, R. C. (1989). *Inside Case-based Reasoning*. Hillsdale, NJ: Erlbaum.
- Reed, S. K. (1999). *Word problems*. Mahwah, NJ: Erlbaum.
- Ross, B. H. (1984). Reminders and their effects in learning a cognitive skill. *Cognitive Psychology*, *16*, 371-416.
- Ross, B. H. (1987). This is like that: The use of earlier problems and the separation of similarity effects. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *13*, 629-639.

- Ross, B. H. (1989). Distinguishing types of superficial similarities: Different effects on the access and use of earlier problems. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 15, 456-468.
- Ross, B. H. & Kennedy, P. T. (1990). Generalizing from the use of earlier examples in problem solving. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 16, 42-55.
- Scheiter, K., Gerjets, P. & Heise, E. (2000). Hypertext navigation and conflicting goal intentions: Using log files to study distraction and volitional protection in learning and problem solving. In L.R. Gleitman & A.K. Joshi (Eds.), *Proceedings of the 22nd Annual Conference of the Cognitive Science Society* (pp. 441-446). Mahwah, NJ: Erlbaum.
- Schoenfeld, A. H. (1985). *Mathematical problem solving*. San Diego, CA: Academic Press.
- Schunn, C. D. & Dunbar, K. (1996). Priming, analogy, and awareness in complex reasoning. *Memory & Cognition*, 24, 271-284.
- Sowder, L. (1988). Children's solutions of story problems. *Journal of Mathematical Behavior*, 7, 227-238.
- Spiro, R. J. & Jehng, J.-C. (1990). Cognitive Flexibility and Hypertext: Theory and technology for the nonlinear and multidimensional traversal of complex subject matter. In D. Nix & R. J. Spiro (Eds.), *Cognition, Education, and Multimedia* (pp. 163-205). Hillsdale, NJ: Erlbaum.
- Staub, F. C. & Reusser, K. (1995). The role of presentational structure in understanding and solving mathematical word problems. In C. A. Weaver, S. Mannes & C. R. Fletcher (Eds.), *Discourse comprehension: Essays in honor of Walter Kintsch* (pp. 285-305). Hillsdale, NJ: Erlbaum.
- Strube, G. & Janetzko, D. (1990). Episodisches Wissen und fallbasiertes Schließen: Aufgaben für die Wissensdiagnostik und die Wissenspsychologie. *Schweizerische Zeitschrift für Psychologie*, 49, 211-221.
- Sweller, J. (1989). Cognitive technology: Some procedures for facilitating learning and problem solving in mathematics and science. *Journal of Educational Psychology*, 81, 457-466.
- Tack, W. H. (1987). Ziele und Methoden der Wissensrepräsentation. *Sprache & Kognition*, 3, 150-163.
- Thagard, P. (1996). *Mind*. Cambridge, MA: MIT-Press.
- VanLehn, K. (1989). Problem solving and cognitive skill acquisition. In M. I. Posner (Ed.), *Foundations of cognitive science* (pp. 527-579). Cambridge, MA: MIT-Press.
- VanLehn, K. (1996). Conceptual and meta learning during coached problem solving. In C. Frasson, G. Gauthier & A. Lesgold (Eds.), *ITS'96: Proceedings the Third International Conference on Intelligent Tutoring Systems* (pp. 29-47). New York: Springer.
- VanLehn, K. & Jones, R. M. (1993). Better learners use analogical problem solving sparingly. In P. E. Utgoff (Ed.), *Machine Learning: Proceedings of the Tenth Annual Conference* (pp. 338-345). San Mateo, CA: Morgan Kaufmann.
- Veloso, M. M. (1994). *Planning and learning by analogical reasoning*. Berlin: Springer.
- Weber, G. (1994). *Fallbasiertes Lernen und Analogien*. Weinheim: Beltz.

Teil III: Formaler Anhang

A. ACT-R: Definitionen und Gleichungen

1. Symbolic level	63
1.1 Declarative memory: chunks.....	63
Chunks.....	63
Beispiel (1): chunk-type.....	63
Beispiel (2): chunk.....	63
Beispiel (3): subtype.....	64
Lernen von chunks.....	64
Goal stack.....	64
Beispiel (4): goal-chunk.....	64
1.2 Procedural memory: productions.....	65
Anforderungen an Produktionsregeln.....	65
Struktur und Instantiierung von Produktionen.....	65
Beispiel (5): Produktionsinstantiierung.....	65
Typen von Produktionen.....	66
Production compilation.....	66
Beispiel (6): dependency-Produktion.....	66
Beispiel (7): dependency-chunk.....	66
Beispiel (8): erlernte Produktionsregel.....	67
2. Subsymbolic level	68
2.1 Declarative memory: retrieval.....	69
Activation A_i	69
Activation equation (1).....	69
Base-level activation B_i	69
Base-level activation equation (2).....	69
Base-level equation (3).....	69
Base-level learning equation (4).....	70
Approximative base-level equation (5).....	70
Kontextaktivierung $\sum_j W_j S_{ji}$	71
Beispiel (9): activation spreading.....	71
Beispiel (10): source activation.....	71
Associative strength equation (6).....	72
Approximative associative strength equation (7).....	72
Prior strength equation (8).....	73
Posterior strength equation (9).....	73
Chunk retrieval.....	74
Chunk retrieval probability.....	74
Retrieval probability equation (10).....	74
Partial matching.....	74
Beispiel (11): partial matching.....	74
Match equation (11).....	75
Chunk choice equation (12).....	75
Chunk retrieval time.....	76
Retrieval time equation (13).....	76
2.2 Procedural memory: conflict resolution.....	76
Conflict resolution mechanism.....	76

Expected gain E.....	77
Expected gain equation (14)	77
Probability of goal equation (15)	77
Probability learning equation (16)	77
Cost of goal equation (17)	77
Cost learning equation (18)	78
Subgoal value equation (19)	78
Boltzmann-Gleichung.....	78
Conflict-resolution equation (20).....	78
Instantiierung.....	79
Produktionsstärke S_p	79
Production strength equation (21).....	79

Die Theorie von ACT-R unterscheidet deklaratives von prozeduralem Gedächtnis sowie verschiedene Ebenen der Wissensrepräsentation (symbolisch vs. subsymbolisch). Lernprozesse finden in ACT-R sowohl auf symbolischer Ebene im Sinne von Neuerwerbungen von Wissenseinheiten oder Regeln als auch auf subsymbolischer Ebene als Anpassung bereits vorhandener Wissensstrukturen an damit gemachte Erfahrungen statt.

1. Symbolic level

1.1 Declarative memory: chunks

Chunks: Das deklarative Wissen wird in ACT-R in Form von chunks repräsentiert, die nach ihren chunk-types, der abstrakten Grundstruktur, unterschieden werden können. Die Anzahl von slots in dem jeweiligen type bestimmt, wieviel Information in einem chunk dieses types dargestellt werden kann bzw. mit welchen anderen chunks eine symbolische Verknüpfung besteht.

(chunk-type addition-fact addend1 addend2 sum)

Beispiel (1): chunk-type

Dementsprechend enthält ein chunk den speziellen *isa*-slot, welcher seinen type kennzeichnet (*addition-fact*), sowie weitere seinem type zugeordnete slots, in denen andere chunks als slot-filler gebunden sein können.

*(fact3+4
isa addition-fact
addend1 three
addend2 four
sum seven)*

Beispiel (2): chunk

Chunk-types können generell hierarchisch organisiert sein, indem zwischen den einzelnen types Beziehungen im Sinne von "Untertyp von" (*subtype*) bestehen können.

(chunk-type math-task difficulty)

Beispiel (3): subtype

(chunk-type addition-fact (:include math-task) addend1 addend2 sum)

Dabei enthält jeder Untertyp (*addition-fact*) neben den für diesen Typ definierten slots automatisch auch die slots des Obertyps (*math-task*), im Beispiel also den slot *difficulty*. Ungeklärt ist die Frage, wie chunk-types und auch deren hierarchische Organisation erlernt werden können. Es scheint bisher nicht möglich zu sein, dass ein Modell während eines runs selbständig neue Kategorien im Sinne von chunk-types generiert.

Lernen von chunks: Prinzipiell gibt es nur zwei Möglichkeiten für die Bildung von chunks (Lernen von Wissenseinheiten):

- (1) Ergebnis der erfolgreichen Ausführung von Produktionen: Beim Pop (with success) des aktuellen goal wird dieser goal-chunk dem deklarativen Gedächtnis als chunk hinzugefügt.
- (2) Resultat der Enkodierung externaler Reize (vgl. visual interface, ACT-R/PM)

Goal stack: Eine spezielle Kategorie innerhalb des deklarativen Gedächtnisses stellt der goal stack dar, auf welchem das aktuell zu bearbeitende prozesssteuernde Ziel abgelegt wird.

*(goal
isa addition-fact
addend1 three
addend2 four
sum nil)*

Beispiel (4): goal-chunk

Handelt es sich bei dem aktuellen Ziel um eine komplexe Aufgabe, so werden durch Produktionen Teilziele gebildet. Diese werden auf das ursprüngliche Ziel gestapelt und bestimmen den weiteren Zyklus von anzuwendenden Produktionen, bis sie wieder gepopt werden und damit das Ursprungsziel wieder aktuell wird. Dabei ist es auch möglich, dem Ursprungsziel abgerufenes oder neu erworbenes Wissen aus den subgoals zu übergeben (goal passing).

Der goal stack ist also nach dem Prinzip „last in – first out“ konzipiert, wobei es sich um ein perfektes Gedächtnis handelt, d.h. innerhalb dieser Zielhierarchie sind keinerlei Fehler bzgl. der Reihenfolge oder des Inhalts der Ziele vorgesehen. Diese sehr starke Annahme wird momentan innerhalb der ACT-R community diskutiert und erste Modellierungen mit einer geänderten Konzeption (perfektes Erinnern von nur drei goals; Verwerfen des stack) wurden vorgestellt, so dass mit einer Modifikation der Theorie bzgl. dieser Problematik zu rechnen ist.

1.2 Procedural memory: productions

Anforderungen an Produktionsregeln: Prozedurales Wissen wird in ACT-R in Form von Produktionsregeln repräsentiert, welche aus condition- und action-part bestehen. Nach Anderson (1993) müssen Produktionsregeln den folgenden Forderungen genügen:

- (1) Modularität: Produktionsregeln als atomare Einheit des Wissenserwerbs
- (2) Abstraktion: Spezifität der Regeln für bestimmte Muster, nicht für bestimmte Werte
- (3) Zielbestimmung: Verantwortung des internen Zustands für die Vorgehensweise
- (4) Asymmetrie zwischen Bedingung und Aktion: Kontrollfluss von Bedingung zu Aktion nicht umkehrbar

Struktur und Instantiierung von Produktionen: Eine Produktion kann durch folgende allgemeine Struktur beschrieben werden:

goal condition + chunk retrieval ==> goal transformations

Der Bedingungsteil enthält als erstes die goal-condition, welche das Ziel repräsentiert, zu dessen Bearbeitung die Regel eingesetzt werden kann. Außerdem können im condition-part weitere chunks abgerufen werden (chunk retrieval). Dabei werden chunks aus dem deklarativen Gedächtnis, die den Spezifikationen der Produktion entsprechen, an die Produktion gebunden und das Wissen steht für die Bearbeitung des aktuellen Ziels zur Verfügung. Dieser Prozess der Instantiierung wird auch als pattern matching-Zyklus bezeichnet (vgl. auch S. 20).

Nur wenn der Bedingungsteil das aktuelle goal matcht und Instantiierungen für den chunk-retrieval gefunden werden, kann die Produktion feuern, d.h. der Aktionsteil wird ausgeführt.

<pre>(p find-sum =goal> isa addition-fact addend1 =addend1 addend2 =addend2 sum nil =addition-fact> isa addition-fact addend1 =addend1 addend2 =addend2 sum =sum ==> =goal> sum =sum !pop! !output! („The sum of =addend1 and =addend2 equals =sum.“)</pre>	<p>Instantiierung:</p> <pre>(p find-sum goal> isa addition-fact addend1 three addend2 four sum nil fact3+4> isa addition-fact addend1 three addend2 four sum seven ==> goal> sum seven !pop! !output! („The sum of three and four equals seven.“)</pre>	<p>Beispiel (5): Produktionsinstantiierung</p>
---	---	---

Typen von Produktionen: Grundsätzlich lassen sich nach Art der ausgeführten Aktionen sechs (3 x 2) verschiedene Typen von Produktionen klassifizieren, die sich aus Kombination der folgenden Merkmale ergeben.

- (1) Modifikation des goal stack: push (a new subgoal); pop (the current goal); no modification
- (2) Modifikation des Ziels: ja; nein

Production compilation: Damit in ACT-R neue Produktionen erlernt werden können, muss dies durch den expliziten Aufruf einer entsprechenden „Lern-Produktion“ initiiert werden (study-dependency). Damit soll ein Schritt im Problemlöseprozess bzw. die bewusste Reflexion über einen solchen symbolisiert werden.

```
(p study-dependency
 =goal>
  isa dependency
 ==>
  !push!
```

Beispiel (6): dependency-Produktion

Dependency-chunks stellen den goal-chunk in abstrakter Form dar, d.h. sie repräsentieren das darin enthaltene Wissen in variabilisierter Form.

```
(example
 isa dependency
 goal goal1
 modified goal2
 constraints fact34)
```

Beispiel (7): dependency-chunk

```
(goal1
 isa add-column
 number1 three
 number2 four
 sum nil)
```

```
(goal2
 isa add-column
 number1 three
 number2 four
 sum seven)
```

```
(fact34
 isa addition-fact
 arg1 three
 arg2 four
 sum seven)
```

Sobald dieses dependency-goal gepopt ist, steht es – wie alle goal-chunks - als chunk im deklarativen Gedächtnis zur Verfügung. Zusätzlich wird aus dem dependency-chunk die entsprechende Produktionsregel gebildet.

```

(p add-numbers
 =goal>
  isa add-column
  number1 =number1
  number2 =number2
  sum nil
 =addition-fact>
  isa addition-fact
  arg1 =number1
  arg2 =number2
  sum =sum
 ==>
 =goal>
  sum =sum)

```

Beispiel (8): erlernte Produktionsregel

Die automatisierte Variabilisierung des im goal-chunk repräsentierten Wissens für die Darstellung im dependency-chunk erfolgt, wenn wiederholt die gleichen chunks an slots gebunden sind. Entweder sind also slots des Bedingungs- und des Aktionsteils mit denselben fillers belegt (=sum), oder innerhalb des Bedingungs- oder des Aktionsteils wird derselbe filler mehrfach an slots gebunden.

Ein dependency-chunk enthält folgende slots:

- goal: ursprüngliches Ziel
- constraints: chunk-retrieval als Übergang vom Bedingungs- zum Aktionsteil
- modified: Ziel im Endzustand
- stack: Änderungen des stack (push subgoal, success, failure)

Dadurch wird gewährleistet, dass mittels der production compilation alle Arten von Produktionsregeln (s.o.) erzeugt werden können (2x3 für goal-Modifikation: ja/nein; Änderung des stack: push subgoal, success, failure). Die Annahme einer automatisierten Variabilisierung kann in einer Modellierung auch zu unerwünschten Ergebnissen führen. Zur kontrollierten Steuerung des Prozesses können daher zusätzliche Modifikationen in der Modellierung vorgenommen werden, welche in speziellen slots gebunden werden.

- specifics: Verhinderung einer Variabilisierung trotz mehrfacher Realisierung
- generals: Erzwingen einer Variabilisierung trotz einfachen Auftretens der Realisierung
- don't cares: Ignorieren des slots bei Erzeugen der Produktionsregel
- difference: Wert eines slots muss sich von einem vorgegebenen Wert unterscheiden

Auch bei der production compilation soll der Anspruch aufrechterhalten werden, die atomic components of thought nachzubilden, d.h. nur einfache, leicht nachvollziehbare Schritte sollen jeweils modelliert werden, welche im Gesamten ein komplexes Muster ergeben können. Daher wird per Konvention bzgl. der compiled productions gefordert, dass nur eine begrenzte Zahl von Abrufen (i.d.R. nur ein chunk-retrieval) auf der left-hand-side stattfindet und dass keine Modifikationen von chunks – mit Ausnahme des goal-chunks – auf der right-hand-side vorgenommen werden.

Der hier skizzierte Mechanismus, der ACT-R das Erlernen neuer Produktionsregeln ermöglicht, wird innerhalb der ACT-R community stark diskutiert, da er in einigen Annahmen durchaus problematisch zu sehen ist. Dazu gehört, dass es eines speziellen Aufrufs bedarf, d.h. ein unbewusstes Lernen prozeduralen Wissens ist nicht vorgesehen. Außerdem bedarf der Mechanismus der primär automatischen, z.T. aber auch kontrollierten Variabilisierung noch einer genaueren Prüfung. Das größte Problem besteht aber wohl darin, dass ein direkter Vergleich von zwei gegebenen Beispielen nicht zur Bildung einer Produktionsregel verwendet werden kann.

2. Subsymbolic level

Die symbolische Wissensrepräsentation wird in ACT-R durch eine subsymbolische Ebene ergänzt, da eine ausschließlich symbolische Repräsentation mit verschiedenen Nachteilen verbunden wäre. Beispielsweise läge eine perfekte Erinnerung des deklarativen Wissens vor, d.h. es gälte ein „Alles oder nichts“-Prinzip. Die Instantiierung einer Produktion, d.h. das binding von chunks aus dem deklarativen Gedächtnis in der left-hand-side einer Produktionsregel, könnte daher nur fehlschlagen, wenn ein chunk als Wissensseinheit nicht existierte. Ein Vergessen bzw. ein vorübergehendes Nicht-Erinnern von chunks wäre bei einer Beschränkung auf die symbolische Ebene von ACT-R also nicht möglich. Weiterhin könnten mehrere Produktionen auf der left-hand-side das aktuelle goal matchen, d.h. gleichermaßen zur Bearbeitung des Ziels zur Verfügung stehen. Bei einer rein symbolischen Repräsentation würde dann automatisch die Produktion instantiiert und ausgeführt, die als erste in der Modellierung programmiert wurde. Es existierte damit keine Möglichkeit, die Auswahl zwischen diesen Produktionen nach weiteren festgelegten Kriterien oder auch nur zufällig zu gestalten.

Wegen dieser Nachteile einer rein symbolischen Repräsentation verfügt ACT-R zusätzlich zu der symbolischen Ebene über eine elaborierte subsymbolische Ebene, wodurch die oben genannten Probleme vermieden werden können. Subsymbologische Parameter, welche als Wahrscheinlichkeits-schätzungen konzipiert sind, regeln die Verfügbarkeit der Inhalte des deklarativen Gedächtnisses und ermöglichen damit die Modellierung von fehlerhaften Erinnerungen. Außerdem können subsymbologische Parameter als Kriterien genutzt werden, anhand derer aus konkurrierenden Produktionsregeln eine Produktion zur Bearbeitung des aktuellen Ziels ausgewählt wird.

Mit der in ACT-R vorliegenden Konzeption einer subsymbolischen Ebene ist es möglich, auf Erfahrung beruhende Lernprozesse zu modellieren, da sich die Parameter als Wahrscheinlichkeitsschätzungen den bisher gemachten Erfahrungen anpassen (statistical learning). Damit erhöhen bzw. erniedrigen sich die Chancen, dass die entsprechenden Wissensseinheiten oder Produktionsregeln in Zukunft wieder erinnert bzw. eingesetzt werden.

2.1 Declarative memory: retrieval

Im vorangegangenen Kapitel wurde bereits erläutert, dass Produktionen aus drei Elementen bestehen: goal condition, chunk retrieval (beides left-hand-side) und goal transformations (right-hand-side). Für die Instantiierung einer Produktion, deren goal condition mit dem aktuellen Verarbeitungsziel übereinstimmt, kann also – bei entsprechender Spezifikation im Bedingungsteil – der Abruf von chunks aus dem deklarativen Gedächtnis notwendig sein.

Activation A_i : Ob und wie schnell ein Gedächtnisabruf erfolgen kann, hängt von der Aktivierung A_i des chunks ab, welche sich folgendermaßen zusammensetzt:

$$A_i = B_i + \sum_j W_j \cdot S_{ji} \quad \text{Activation equation (1)}$$

mit B_i = base level activation
 W_j = source activation
 und S_{ji} = associative strength

BASE-LEVEL ACTIVATION B_i : Mit B_i wird die Basisaktivierung des chunks bezeichnet, welche kontextunabhängig ist, d.h. nicht in Beziehung zu dem aktuell zu bearbeitenden Ziel steht. Wahrscheinlichkeitstheoretisch betrachtet wird in der Konzeption der base-level activation die Wahrscheinlichkeit, dass ein chunk benutzt wird, mit der Wahrscheinlichkeit, dass er nicht benutzt wird, in Bezug zueinander gesetzt. Demzufolge wird die base-level activation auch als Komponente angesehen, welche die allgemeine Brauchbarkeit eines chunks beschreibt.

$$B_i = \ln \left(\frac{P(C_i)}{P(\bar{C}_i)} \right) \quad \text{Base-level activation equation (2)}$$

mit C_i : chunk i wird benutzt

Zur Schätzung der base-level activation müssen zwei Fälle unterschieden werden:

(1) a priori-Schätzung

Bei der Bildung eines neuen chunks werden die Annahmen getroffen, dass dessen base-level activation den Wert einer Zufallsvariablen einnimmt, dem decay unterworfen ist und durch weitere additive Zufallsvariablen modifiziert wird. Damit ergibt sich für die base-level eines chunks zum Zeitpunkt t nach seiner Bildung (unter der Annahme keiner weiteren Nutzung):

$$B(t) = \beta - d \ln(t) + \varepsilon_1 + \varepsilon_2 \quad \text{Base-level equation (3)}$$

expected base-level
decay with time d=decay rate
permanent fluctuations
moment-to-moment fluctuations

(2) a posteriori-Schätzung

Bei der Berechnung der base-level activation eines chunks werden die Zeitpunkte der Nutzung dieses chunks berücksichtigt und ein Aktivationsverfall über die Zeit angenommen. Weiterhin wird eine additive Konstante β eingeführt, welche die base-level activation zusätzlich zu der durch Nutzung erreichten Aktivierung erhöhen kann. Durch die Konzeption der base-level activation werden sowohl frequency- als auch recency-Effekte einbezogen.

$$B_i = \ln \left(\sum_{j=1}^n t_j^{-d} \right) + \beta \quad \text{mit } d = \text{decay rate} \quad \text{Base-level learning equation (4)}$$

t_j = Zeitspanne (s.u.)
 n = Anzahl der Nutzungen des chunks
 $\beta = \ln(\alpha)$ mit α = scaling constant

Bezüglich der Konzeption der Zeitspanne t_j müssen wieder zwei Fälle unterschieden werden:

(a) erste Nutzung des chunks nach seiner Bildung

Für die Entstehung eines chunks gibt es zwei Möglichkeiten. Zum einen kann ein chunk das Ergebnis der Enkodierung externer Reize sein. Dann beschreibt t_j die Zeitspanne zwischen dem Moment, als die Aufmerksamkeit auf den entsprechenden Reiz gelenkt wurde, und dem aktuellen Zeitpunkt der ersten Nutzung des chunks. Zum anderen kann es sich bei dem chunk um einen goal-chunk handeln, welcher nach seiner erfolgreichen Abarbeitung (*!pop!*) im deklarativen Gedächtnis zur Verfügung steht. In diesem Fall beschreibt t_j die Zeitspanne zwischen dem Moment, als dieser goal-chunk erzeugt wurde, und dem aktuellen Zeitpunkt der ersten Nutzung des chunks.

(b) jede weitere Nutzung des chunks

Wurde ein chunk bereits mindestens einmal genutzt, d.h. bei der Instantiierung einer Produktion gebunden, so beschreibt t_j jeweils die Zeitspanne zwischen dem Start des vorangegangenen pattern matching-Zyklus und dem Start des nachfolgenden pattern matching-Zyklus.

Mittels der a posteriori-Schätzung der base-level learning equation kann das Power Law of Learning (Newell & Rosenbloom, 1981) vorhergesagt werden, wenn die n Nutzungen des chunks als gleichverteilt angenommen werden können. Dann kann die base-level activation approximativ durch folgende Gleichung beschrieben werden.

$$B_i = \ln \left(\frac{n}{1-d} \right) - d \ln T \quad \text{Approximative base-level equation (5)}$$

mit d = decay rate
 T = Zeitspanne zwischen Entstehen des chunks und aktueller Nutzung
 n = Anzahl der Nutzungen des chunks

KONTEXTAKTIVATION $\sum_j W_j \cdot S_{ji}$: Der Term $\sum_j W_j \cdot S_{ji}$ bezeichnet eine kontextspezifische

Aktivation, d.h. eine Aktivation, die in Abhängigkeit des aktuellen Ziels zu sehen ist. Von diesem Ziel – genauer von den slot-fillers in diesem goal-chunk – geht Quellaktivierung W_j aus. Diese gewichtet die assoziative Stärke S_{ji} , mit der diese slot-fillers mit anderen chunks verbunden sind. Diese chunks erhalten dadurch zusätzlich zu ihrer base-level activation weitere Aktivierung, so dass die allgemeine Aktivierung A_i der chunks jeweils erhöht ist und letztlich ihr Abruf damit wahrscheinlicher (vgl. chunk retrieval). Der hier dargestellte Mechanismus wird als ‘spreading activation’-Mechanismus bezeichnet.

Als default-Einstellung im ACT-R-Environment werden chunks mit folgender Spezifikation automatisch als mit den slot-fillers eines goal assoziiert angesehen und erhalten damit automatisch spreading activation:

- (1) die slot-filler chunks selbst
- (2) chunks, bei denen diese slot-filler chunks ebenfalls als slot-fillers fungieren

goal-chunk	<pre>goal isa addition-fact addend1 three addend2 four sum nil</pre>	Beispiel (9): activation spreading
activation spreading z.B. zu	(1)	<pre>three four isa integer isa integer value 3 value 4</pre>
	(2)	<pre>drei fact3+4 isa integer isa addition-fact name three addend1 three value 3 addend2 four sum seven</pre>

Die spreading activation breitet sich von diesen chunks nicht mehr weiter zu (mit diesen) assoziierten chunks aus; die Konzeption sieht also ein sogenanntes ‘one-step activation spreading’ vor.

Source activation W_j : Die source activation stellt die Grundlage für das Konzept der spreading activation dar. Wie bereits dargestellt geht von den slot-fillers eines goal-chunk eine Quellaktivierung W_j aus, welche die jeweilige Assoziationsstärke zu assoziierten chunks gewichtet. Dabei wird von einer konstanten Kapazität der source activation W_j (default-value 1) ausgegangen, welche sich auf alle betrachteten slot-fillers gleich verteilt. Je mehr slot-fillers also ein goal-chunk enthält, umso weniger source activation entfällt auf jeden einzelnen slot-filler.

<pre>=goal> isa addition-fact addend1 three addend2 four sum nil</pre>	W_j 1/2 1/2	<pre>=goal> isa addition-fact arg1 three arg2 four operator plus result nil</pre>	W_j 1/3 1/3 1/3	Beispiel (10): source activation
---	---------------------	--	----------------------------	----------------------------------

Die source activation wird auf Grund der Annahme einer konstanten Kapazität und der Gleichverteilung auf die slot-fillers des goal-chunk von Lernmechanismen nicht beeinflusst.

Überlegungen, interindividuelle Unterschiede bzgl. der kognitiven Komplexität von Menschen als Unterschiede im Ausmaß der Quellaktivierung zu beschreiben (und damit die Annahme der Konstanz zu verwerfen), treffen Lovett, Reder & Lebiere (1999)⁶. Die Annahme der Gleichverteilung stellen Anderson & Reder (1999)⁷ in Frage, wobei sie die Verteilung von Quellaktivierung auf die slot-fillers eines goal-chunk als strategische Variable betrachten.

Associative strength S_{ji} : Entscheidend dafür, ob die Aktivierung eines chunks durch spreading activation erhöht werden kann, ist die assoziative Verbindung des chunks zu einem slot-filler im goal-chunk. Die Assoziationsstärke S_{ji} eines chunks ist konzipiert als dessen spezifische Brauchbarkeit bzw. bedingte Nutzungswahrscheinlichkeit und reflektiert die Nützlichkeit eines chunks im Kontext eines bestimmten cues. Konkret wird bzgl. der Assoziationsstärke von chunk i nach der Wahrscheinlichkeit gefragt, mit der chunk j benutzt wird (slot-filler ist), wenn chunk i benötigt wird – relativiert an der Wahrscheinlichkeit, mit der chunk j benutzt wird, wenn chunk i nicht benötigt wird.

$$S_{ji} = \ln \left(\frac{P(C_j | N_i)}{P(C_j | \bar{N}_i)} \right) \quad \text{Associative strength equation (6)}$$

mit C_j : chunk j wird benutzt (slot-filler)
 N_i : chunk i wird benötigt

Die Assoziationsstärke nach dieser Konzeption wird in ACT-R wie folgt approximiert, wobei R_{ji} als Maß dafür angesehen werden kann, inwiefern chunk i – im Vergleich zu seiner base-level activation – mehr oder weniger wahrscheinlich im Kontext von chunk j ist.

$$R_{ji} = \frac{P(C_j | N_i)}{P(C_j)} = \frac{P(N_i | C_j)}{P(N_i)} \quad \text{Approximative associative strength equation (7)}$$

wobei gilt: $S_{ji} = \ln(R_{ji})$

Besteht zwischen chunks keine assoziative Beziehung, so ist $P(C_j | N_i) = P(C_j)$ bzw. $P(N_i | C_j) = P(N_i)$, so dass $R_{ji} = 1$ und damit $S_{ji} = 0$.

⁶ Lovett, M.C., Reder, L.M. & Lebiere C. (1999). Modeling working memory in a unified architecture: An ACT-R perspective. In A. Miyake & P. Shah (Eds.), *Models of working memory* (pp. 135-182). Cambridge: Cambridge University Press.

⁷ Anderson, J.R. & Reder, L.M. (1999). The fan effect: New results and new theories. *Journal of Experimental Psychology: General*, 128, 186-197.

Ebenso wie bei der Schätzung der base-level activation B_i müssen auch bei der Schätzung der associative strength zwei Fälle unterschieden werden:

(1) a priori-Schätzung

Die a priori Schätzung geht aus von der Anzahl n der mit chunk j (slot-filler im goal-chunk) verbundenen chunks sowie der Gesamtzahl m von chunks im deklarativen Gedächtnis. Zusätzlich werden folgende Annahmen getroffen:

$$(1) \quad P(N_i | C_j) = \frac{1}{n} \quad \text{also Gleichwahrscheinlichkeit aller mit chunk } j \text{ verbundenen chunks,}$$

wenn chunk j im goal-chunk gebunden ist

$$(2) \quad P(N_i) = \frac{1}{m} \quad \text{also Gleichwahrscheinlichkeit aller chunks im deklarativen Gedächtnis}$$

Unter diesen Voraussetzungen gilt:

$$R_{ji}^* = \frac{m}{n} \quad \text{bzw. } S_{ji}^* = \ln(m) - \ln(n) \quad \text{Prior strength equation (8)}$$

mit m = number of chunks in declarative memory
 n = number of chunks connected to j

Wenn m als Anzahl von chunks im deklarativen Gedächtnis nicht bekannt ist, wird in der Prior strength equation $\ln(m)$ durch die geschätzte Konstante S^* ersetzt.

(2) a posteriori-Schätzung

Liegen bereits Erfahrungen bzgl. der Nutzbarkeit eines chunks i in Abhängigkeit von dem slot-filler j im aktuellen goal vor, hat also bereits mindestens ein retrieval von chunk i seit dessen Bildung im Kontext von chunk j stattgefunden, so werden diese Erfahrungen zur Schätzung der Assoziationsstärke herangezogen. Diese Kombination der a priori Schätzung R_{ji}^* mit den empirischen Erfahrungen ist in der posterior strength equation dargestellt.

$$R_{ji} = \frac{assoc * R_{ji}^* + F(C_j) E_{ji}}{assoc + F(C_j)} \quad \text{wobei } S_{ji} = \ln(R_{ji}) \quad \text{Posterior strength equation (9)}$$

mit $assoc$ = Konstante (als Gewichtung)

R_{ji}^* = a priori-Schätzung (vgl. prior strength equation (8))

$F(C_j)$ = Häufigkeit der Bindung von chunk j im goal-chunk

$$E_{ji} = \frac{P_e(N_i | C_j)}{P_e(N_i)} = \text{empirical ratio}$$

mit:

$$P_e(N_i | C_j) = \frac{F(N_i \cap C_j)}{F(C_j)} = \frac{\text{Häufigkeit, daß chunk } i \text{ gebraucht, wenn chunk } j \text{ im goal - chunk gebunden}}{\text{Häufigkeit der Bindung von chunk } j \text{ im goal - chunk}}$$

$$P_e(N_i) = \frac{F(N_i)}{F(i)} = \frac{\text{Häufigkeit, daß chunk } i \text{ gebraucht}}{\text{Häufigkeit des Feuerns von Produktionen seit Bildung von } i}$$

Chunk retrieval: Zur Instantiierung von Produktionen kann der Abruf von chunks aus dem deklarativen Gedächtnis notwendig sein.

CHUNK RETRIEVAL PROBABILITY: Damit ein chunk aus dem deklarativen Gedächtnis abgerufen werden kann, muss seine Aktivierung einen angenommenen Schwellenwert überschreiten. Dafür kann zu jedem gegebenen Zeitpunkt lediglich eine Wahrscheinlichkeitsschätzung angegeben werden, da bereits die Schätzung der Aktivierung eines chunks Zufallsmomente beinhaltet. Unter Annahme einer logistischen Verteilung für die noise distribution kann die Wahrscheinlichkeit dafür, dass ein chunk abgerufen werden kann, folgendermaßen berechnet werden:

$$\text{probability} = \frac{1}{1 + e^{-\frac{(A-\tau)}{s}}} \quad \text{Retrieval probability equation (10)}$$

mit A = Aktivierung des chunks
 τ = Schwellenwert (retrieval treshold)

$$s = \frac{\sqrt{3}\sigma}{\pi} \quad \text{mit } \sigma^2 = \text{combined temporary and permanent variance ("noise")}$$

Ist der Abruf eines chunks, der im Bedingungsteil einer Produktion matchen würde, wegen zu geringer Aktivierung im Vergleich zum Schwellenwert nicht möglich, nennt man dies einen 'error of omission'.

PARTIAL MATCHING: Der Mechanismus des partial matching sieht vor, dass beim Matchen eines chunks für die Instantiierung einer Produktionsregel der ausgewählte chunk nur teilweise der production-rule specification entspricht. Die mögliche Abweichung bezieht sich allerdings nur auf die Spezifikation der slot-filler, während der chunk-type des ausgewählten chunks dem in der Produktionsregel spezifizierten entsprechen muss.

goal-chunk	<i>goal ></i> <i>isa addition-fact</i> <i>addend1 three</i> <i>addend2 four</i> <i>sum nil</i>	Beispiel (11): partial matching	
chunk-retrieval	<i>=addition-fact></i> <i>isa addition-fact</i> <i>addend1 =addend1</i> <i>addend2 =addend2</i> <i>sum =sum</i>	gesuchter chunk	<i>fact3+4></i> <i>isa addition-fact</i> <i>addend1 three</i> <i>addend2 four</i> <i>sum seven</i>
		partial matching z.B.	<i>fact2+4></i> <i>isa addition-fact</i> <i>addend1 two</i> <i>addend2 four</i> <i>sum six</i>
		partial matching unmöglich	<i>fact3*4></i> <i>isa multiplication-fact</i> <i>addend1 three</i> <i>addend2 four</i> <i>sum twelve</i>

Der partial matching-Mechanismus ermöglicht also bei der Instantiierung einer Produktion ein fehlerhaftes binding eines chunks (Mismatch). Dabei wird ein "falscher" – d.h. der production-rule specification nicht vollständig entsprechender – Gedächtnisinhalte genutzt; dies wird als 'error of commission' bezeichnet.

Unter Berücksichtigung des partial matching-Mechanismus kann beim Abruf eines chunks zur Instantiierung einer Produktion nicht nur dessen Aktivierung entscheidend sein, sondern muss vielmehr auch eine produktionsspezifische Größe beachtet werden. Als solche Größe wird der degree of mismatch D_{ip} eingeführt. Dieser stellt ein Maß für die Unterschiede zwischen einem chunk und der production-rule specification dar. Konkret gibt der degree of mismatch die Zahl der slots des abzurufenden chunks an, in denen Mismatches vorliegen, d.h. deren slot-filler nicht mit der Spezifikation in der Produktion übereinstimmen.

Aus dem Zusammenspiel der beiden für den Abruf eines chunks aus dem deklarativen Gedächtnis relevanten Parameter, nämlich der Aktivierung und dem degree of mismatch, lässt sich der sogenannte match score M_{ip} bestimmen⁸.

$$M_{ip} = A_i - D_{ip} \quad \text{Match equation (11)}$$

mit A_i = Aktivierung eines chunks i
 D_{ip} = degree of mismatch eines chunks i bzgl. einer bestimmten Produktion p

Mit Hilfe dieses match scores kann nun die Wahrscheinlichkeit dafür, dass ein chunk i zum match einer Produktion p abgerufen wird, approximativ bestimmt werden.

$$\text{probability} = \frac{e^{\frac{M_{ip}}{t}}}{\sum_j e^{\frac{M_{jp}}{t}}} \quad \text{Chunk choice equation (12)}^9$$

mit M_{ip} = match score von chunk i relativ zur Produktion p

$$t = \frac{\sqrt{6} \sigma}{\pi} = \sqrt{2} s \quad (\text{"noise"})$$

j = Index für alle chunks im deklarativen Gedächtnis

⁸ Im Prinzip müsste in der ,retrieval probability equation (10)' der match score M_{ip} an Stelle der Aktivierung A_i eingesetzt werden, da es sich bei dem partial matching-Mechanismus auf der Theorieebene um ein generelles Konzept handelt. Allerdings wurde der Mechanismus in ACT-R 4.0 als Vorschlag an die Community eingeführt, der in der praktischen Modellierung eher selten umgesetzt wird. Insofern ist diese Inkonsistenz bei der Verwendung des match score M_{ip} wohl als „Mismatch“ zwischen Theorie und Praxis von ACT-R zu verstehen.

⁹ Der genaue Bezug zwischen der ,retrieval probability equation (10)' und der ,chunk choice equation (12)' ist unklar.

Im Gegensatz zur 'retrieval probability equation (10)' wird hierbei kein einzelner chunk betrachtet, sondern ein chunk i immer in Bezug zu allen anderen chunks im deklarativen Gedächtnis gesehen. Die Wahrscheinlichkeitsschätzung wird gemäß der Boltzmann-Gleichung vorgenommen, so dass meist, auf Grund des "Rauschens" (vgl. Parameter t) aber nicht immer der chunk mit der höchsten Wahrscheinlichkeit ausgewählt wird ('soft-max rule').

CHUNK RETRIEVAL TIME: Der Zeitbedarf für das Ausführen einer Produktionsregel ist abhängig von der Zeit, die für den Abruf der im Bedingungsteil benötigten chunks erforderlich ist¹⁰. Diese 'chunk retrieval time' wird bestimmt durch die Zahl der erforderlichen Gedächtnisabrufe. Dabei ergibt sich die total retrieval time einer Produktion – auf Grund der Annahme der Serialität beim chunk-retrieval – als Summe der Abrufzeiten aller im Bedingungsteil zu instantiierenden chunks.

Die zum Abruf eines bestimmten chunks benötigte Zeit ist abhängig von seinem match score (damit auch von der Aktivierung) sowie der Produktionsstärke der zu instantiierenden Produktion. Der Zeitbedarf wird unter Annahme einer zu Grunde liegenden Exponentialfunktion mit den freien Parametern F (latency scale factor) und f (latency exponent) geschätzt mit

$$Time_{ip} = Fe^{-f(M_{ip} + S_p)} \quad \text{Retrieval time equation (13)}$$

mit F = latency scale factor

f = latency exponent

M_{ip} = match score von chunk i relativ zur Produktion p

S_p = Produktionsstärke der Produktion p

2.2 Procedural memory: Conflict resolution

Ebenso wie für das deklarative Gedächtnis kann auch für das prozedurale Gedächtnis die subsymbolische Ebene betrachtet werden. Auf dieser finden die zentralen Prozesse der Auswahl einer Produktion für die Bearbeitung des aktuellen Ziels statt, welchen ein kosten-nutzen-analytisches Kalkül zu Grunde liegt.

Conflict resolution mechanism: Es kann vorkommen, dass mehrere Produktionen die aktuelle goal-condition matchen und damit prinzipiell – sofern die Instantiierung möglich ist – zur Bearbeitung des current goal eingesetzt werden können. Diese Menge von Produktionen bildet das sogenannte 'conflict resolution set'. Aus dieser Menge muss letztlich eine Produktion ausgewählt werden, wobei dies durch Parameter gesteuert wird. Dieser Prozess wird als 'conflict resolution mechanism' bezeichnet wird.

¹⁰ Für das matching des aktuellen Ziels im Bedingungsteil wird keine Zeit beansprucht.

EXPECTED GAIN E: Dazu wird für alle das aktuelle Ziel matchende Produktionen parallel der zu erwartende Nutzen (expected gain E) abgeschätzt, welcher wie folgt konzipiert ist.

$$E = PG - C \qquad \text{Expected gain equation (14)}$$

mit P = Wahrscheinlichkeit des Erreichens des aktuellen Ziels bei Anwendung der Produktion
 G = goal value
 C = Kosten des Erreichens des aktuellen Ziels (operationalisiert in Zeit)

Wahrscheinlichkeit P: Die Wahrscheinlichkeit P lässt sich in zwei Komponenten zerlegen, welche jeweils die aktuell betrachtete Produktion betreffen sowie nachfolgende Produktionen, deren Einsatz im Anschluss an die aktuelle Produktion bis zur endgültigen (erfolgreichen) Bearbeitung des current goal notwendig ist.

$$P = q * r \qquad \text{Probability of goal equation (15)}$$

mit q = Wahrscheinlichkeit der Zielerreichung bei Einsatz der aktuell betrachteten Produktion
 r = Wahrscheinlichkeit der Zielerreichung durch Einsatz nachfolgender Produktionen

Die Parameter q und r reflektieren sowohl a priori-Erwartungen als auch empirische Erfahrungen – soweit vorhanden – bzgl. des Einsatzes der Produktion. Durch dieses Zusammenspiel theoretischer Erwartungen und empirischer Erfahrungen unterliegt die Schätzung der Wahrscheinlichkeit, dass eine Produktion zum Erreichen eines Ziels beitragen kann, einem Lernprozess. Dabei wird der Einsatz einer Produktion als Erfolg gewertet, wenn das aktuelle Ziel durch die Produktion (q) bzw. durch nachfolgende Produktionen (r) erreicht werden konnte; ein Misserfolg bezieht sich entsprechend auf das Nichterreichen des Ziels (pop with failure). Diese Konzeption spiegelt sich in der folgenden 'probability learning equation' wider.

$$q = \frac{\text{successes}}{\text{successes} + \text{failures}} = \frac{\alpha + m}{\alpha + \beta + m + n} \qquad \text{Probability learning equation (16)}$$

(für r entsprechend)

mit α = theoretische Erwartungen bzgl. des Erfolgs (a priori) → default value = 1
 β = theoretische Erwartungen bzgl. des Mißerfolgs (a priori) → default value = 0
 m = empirische Erfahrungen des Erfolgs (a posteriori)
 n = empirische Erfahrungen des Mißerfolgs (a posteriori)

Kosten C: Dieselbe Aufteilung in zwei Komponenten findet sich auch bei den Kosten C des Einsatzes einer Produktion.

$$C = a + b \qquad \text{Cost of goal equation (17)}$$

mit a = Kosten (Zeitbedarf) beim Einsatz der aktuell betrachteten Produktion
 b = Kosten (Zeitbedarf) beim Einsatz nachfolgender Produktionen bis zur Zielerreichung

Auch bei der Kostenschätzung finden a priori-Erwartungen und a posteriori-Erfahrungen gleichermaßen Berücksichtigung, so dass auch hier Lerneffekte auftreten. Zur Schätzung des

mittleren Zeitbedarfs einer Produktion wird der benötigte Aufwand mit Erfolgs- bzw. Mißerfolgserwartungen und –erfahrungen in Zusammenhang gestellt.

$$a = \frac{\text{efforts}}{\text{successes} + \text{failures}} = \frac{Z + \sum_i^m \text{effort}_i}{\alpha + \beta + m + n} \quad \text{Cost learning equation (18)}$$

(für b entsprechend)

- mit Z = a priori-Schätzung der Kosten
 effort_i = tatsächliche Kosten beim Einsatz der Produktion i
 α = theoretische Erwartungen bzgl. des Erfolgs (a priori) → default value = 1
 β = theoretische Erwartungen bzgl. des Mißerfolgs (a priori) → default value = 0
 m = empirische Erfahrungen des Erfolgs (a posteriori)
 n = empirische Erfahrungen des Mißerfolgs (a posteriori)

Goal value G : Der Wert, der einer Produktion beigemessen wird, ist wie die Kosten C als Zeit operationalisiert, also als die Zeitspanne, die man für das Erreichen des Ziels zu investieren bereit ist. Als default value wird ein goal value von 20 [msec] angenommen.

Wird durch eine Produktion ein subgoal zur Erreichung des aktuellen Ziels (topgoal) gesetzt, so ist dessen goal value wie folgt konzipiert.

$$\text{Wert eines Teilzieles: } G' = rG - b \quad \text{Subgoal value equation (19)}$$

- mit r = Wahrscheinlichkeit der Zielerreichung durch Einsatz nachfolgender Produktionen
 G = goal value des topgoal
 b = Kosten (Zeitbedarf) beim Einsatz nachfolgender Produktionen bis zur Zielerreichung

BOLTZMANN-GLEICHUNG: Würde durch den conflict resolution mechanism immer die Produktion mit dem höchsten expected gain ausgewählt werden, so könnten mögliche Fehleinschätzungen oder spontane Entscheidungen im menschlichen Problemlöseprozess nicht modelliert werden. Um eine solche streng deterministische Sichtweise zu vermeiden, ist eine Randomisierung durch eine Zufallsvariable mit logistischer Verteilung vorgesehen. Die Wahrscheinlichkeit für die Auswahl einer Produktion i mit expected gain E_i aus dem conflict resolution set T möglicher Regeln ergibt sich gemäß folgender conflict resolution equation, welche der Boltzmann-Gleichung entspricht:

$$p(i|T) = \frac{e^{\frac{E_i}{t}}}{\sum_{j \in T} e^{\frac{E_j}{t}}} \quad \text{Conflict-resolution equation (20)}$$

- mit i = Auswahl der Produktion i
 T = conflict resolution set
 E_i = expected gain der Produktion i
 j = Index für alle Produktionen des conflict resolution set
 $t = \frac{\sqrt{6} \sigma}{\pi} = \sqrt{2} s$ ("noise")

INSTANTIIERUNG: Ist eine Produktion i mit der höchsten Auswahlwahrscheinlichkeit gemäß der conflict-resolution equation ausgewählt, erfolgt deren Instantiierung, also der Abruf und das Binden von chunks aus dem deklarativen Gedächtnis an die Produktion. Dabei können slots, an die bereits chunks gebunden wurden, nicht neu besetzt werden. Ein backtracking ist also (im Gegensatz zu der Konzeption in ACT*) nicht möglich, d.h. ein Neubeginn des Instantiierungsprozesses unter Beibehaltung der Produktion findet nicht statt. Dies bedeutet, dass eine Produktion nicht eingesetzt wird, wenn für einen slot kein chunk zum binding zur Verfügung steht. In diesem Fall wird zur Bearbeitung des aktuellen Ziels die Produktion mit der nächstgrößten Auswahlwahrscheinlichkeit im conflict resolution set herangezogen und instantiiert. Ein pop with failure des current goal erfolgt, wenn keine Produktionsregel des conflict resolution set erfolgreich instantiiert werden kann. (Dadurch wird die Größe des r -Parameter der Produktion, welche dieses goal auf den stack gelegt hat, verringert.)

Produktionsstärke S_p : Analog zu der Aktivationsstärke A von chunks gibt die Produktionsstärke S_p die Aktivierung einer Produktionsregel an. Diese beeinflusst – wie es in der ‘retrieval time equation (13)’ (vgl. S. 17) deutlich wird – die Geschwindigkeit der Instantiierung einer Produktionsregel. Allerdings besteht kein Zusammenhang zur Auswahl einer Produktionsregel, da dies über das expected gain E der Produktion bestimmt wird. Die Produktionsstärke wird nach der production strength equation geschätzt (vgl. ‘base level learning equation (4)’).

$$S_p = \ln \left(\sum_{j=1}^n t_j^{-d} \right) + \beta \quad \text{Production strength equation (21)}$$

mit d = decay rate
 t_j = Zeitspanne (s.u.)
 n = Anzahl der Nutzungen der Produktion
 $\beta = \ln(\alpha)$ mit α = scaling constant

Bezüglich der Konzeption der Zeitspanne t_j müssen zwei Fälle unterschieden werden:

(a) erste Nutzung der Produktion nach ihrer Bildung

Für die erste Nutzung einer Produktion beschreibt t_j die Zeitspanne zwischen dem Moment der Erzeugung (vgl. production compilation) und dem aktuellen Zeitpunkt der ersten Nutzung der Produktion.

(b) jede weitere Nutzung der Produktion

Wurde eine Produktion bereits mindestens einmal eingesetzt, so beschreibt t_j jeweils die Zeitspanne zwischen dem Start des vorangegangenen pattern matching-Zyklus (Auswahl im conflict resolution set) und dem Start des nachfolgenden pattern matching-Zyklus.

B. ACT-R-Quellcode für die Schlüsselwort-Strategie

```
(clear-all)
(sgp
 :ANS 0.1
 :PAS 0.1
 :EGS 1
 :ER T
 :RT -1.0
 :ERA T
 :BLL 0.5
)
(chunk-type combinatoric-test task1 task2 task3)
(chunk-type task tasknumber status solution)
(chunk-type solution tasknumber principle n k)
(chunk-type taskstring code string count end tasknumber number status)
(chunk-type order first next)
(chunk-type number value tasknumber)
(chunk-type keyword code content tasknumber number)
(chunk-type status)
(chunk-type principle name)
(chunk-type parameter name tasknumber number)
(chunk-type solution_element)
(chunk-type tasknumber)
(add-dm
(goal isa combinatoric-test task1 angler task2 hunde task3 ritter)
(angler isa task tasknumber task1 status unsolved)
(hunde isa task tasknumber task2 status unsolved)
(ritter isa task tasknumber task3 status unsolved)
(string1_1 ISA taskstring code string1_1 tasknumber task1 count 1 number 4 string (Der Verein vegetarischer Angler hat 4 Mitglieder))
(string1_2 ISA taskstring code string1_2 tasknumber task1 count 2 string (Alle Angler haben sich verpflichtet))
(string1_3 ISA taskstring code string1_3 tasknumber task1 count 3 string (gefangene Fische sofort wieder zurueck in den Teich zu setzen))
(string1_4 ISA taskstring code string1_4 tasknumber task1 count 4 string (Eines Tages angeln die Vereinsmitglieder nacheinander))
(string1_5 ISA taskstring code string1_5 tasknumber task1 count 5 number 8 string (an einem Teich von 8 Quadratmetern))
(string1_6 ISA taskstring code string1_6 tasknumber task1 count 6 number 5 string (in dem sich 5 Fische befinden))
(string1_7 ISA taskstring code string1_7 tasknumber task1 count 7 number 5 string (Ein Zander ein Aal eine Forelle ein Hecht und ein Karpfen))
(string1_8 ISA taskstring code string1_8 tasknumber task1 count 8 string (Alle Mitglieder angeln))
(string1_9 ISA taskstring code string1_9 tasknumber task1 count 9 string (in absteigender Altersreihenfolge jeweils einen Fisch))
(string1_10 ISA taskstring code string1_10 tasknumber task1 count 10 string (Wie berechnet man die Wahrscheinlichkeit))
(string1_11 ISA taskstring code string1_11 tasknumber task1 count 11 string (dass per Zufall))
(string1_12 ISA taskstring code string1_12 tasknumber task1 count 12 string (der aelteste Angler))
(string1_13 ISA taskstring code string1_13 tasknumber task1 count 13 string (den Aal geangelt hat))
(string1_14 ISA taskstring code string1_14 tasknumber task1 count 14 number 2 string (und der zweitaelteste))
(string1_15 ISA taskstring code string1_15 tasknumber task1 count 15 end t string (die Forelle))
(string2_1 ISA taskstring code string2_1 tasknumber task2 count 1 number 11 string (Ein Tierheim sucht fuer 11 Hunde ein Zuhause))
(string2_2 ISA taskstring code string2_2 tasknumber task2 count 2 number 4 string (4 der Hunde sind Terrier))
(string2_3 ISA taskstring code string2_3 tasknumber task2 count 3 number 7 string (die restlichen 7 sind Mischlinge))
```

(string2_4 ISA taskstring code string2_4 tasknumber task2 count 4 number 2 string (Es melden sich 2 blonde))

(string2_5 ISA taskstring code string2_5 tasknumber task2 count 5 number 4 string (und 4 schwarzhaarige))

(string2_6 ISA taskstring code string2_6 tasknumber task2 count 6 string (Kinder))

(string2_7 ISA taskstring code string2_7 tasknumber task2 count 7 string (die ein Haustier suchen))

(string2_8 ISA taskstring code string2_8 tasknumber task2 count 8 string (Um Streit zu vermeiden))

(string2_9 ISA taskstring code string2_9 tasknumber task2 count 9 string (werden die Hunde per Zufall ausgelost))

(string2_10 ISA taskstring code string2_10 tasknumber task2 count 10 string (zuerst ziehen die schwarzhaarigen Kinder jeweils ein Los))

(string2_11 ISA taskstring code string2_11 tasknumber task2 count 11 string (Wie berechnet man die Wahrscheinlichkeit))

(string2_12 ISA taskstring code string2_12 tasknumber task2 count 12 string (dass jedes schwarzhaarige Kind))

(string2_13 ISA taskstring code string2_13 tasknumber task2 count 13 end t string (einen Terrier bekommt))

(string3_1 ISA taskstring code string3_1 tasknumber task3 count 1 number 9 string (beim 9. Koenigsturnier))

(string3_2 ISA taskstring code string3_2 tasknumber task3 count 2 number 10 string (beteiligen sich 10 Ritter))

(string3_3 ISA taskstring code string3_3 tasknumber task3 count 3 number 12 string (Der Koenig stellt fuer das Turnier 12 Pferde))

(string3_4 ISA taskstring code string3_4 tasknumber task3 count 4 string (Die Ritter beginnen mit verbundenen Augen))

(string3_5 ISA taskstring code string3_5 tasknumber task3 count 5 string (sich per Zufall ein Pferd))

(string3_6 ISA taskstring code string3_6 tasknumber task3 count 6 string (auszuwaehlen))

(string3_7 ISA taskstring code string3_7 tasknumber task3 count 7 string (Zuerst waehlt der schwerste Ritter))

(string3_8 ISA taskstring code string3_8 tasknumber task3 count 8 string (dann der zweitschwerste usw.))

(string3_9 ISA taskstring code string3_9 tasknumber task3 count 9 string (Wie berechnet man die Wahrscheinlichkeit))

(string3_10 ISA taskstring code string3_10 tasknumber task3 count 10 string (dass der schwerste Ritter))

(string3_11 ISA taskstring code string3_11 tasknumber task3 count 11 string (das groesste Pferd))

(string3_12 ISA taskstring code string3_12 tasknumber task3 count 12 string (der zweitschwerste))

(string3_13 ISA taskstring code string3_13 tasknumber task3 count 13 string (das zweitgroesste))

(string3_14 ISA taskstring code string3_14 tasknumber task3 count 14 number 3 string (und der drittschwerste Ritter))

(string3_15 ISA taskstring code string3_15 tasknumber task3 count 15 end t string (das drittgroesste Pferd bekommt))

(order1_2 isa order first 1 next 2)

(order2_3 isa order first 2 next 3)

(order3_4 isa order first 3 next 4)

(order4_5 isa order first 4 next 5)

(order5_6 isa order first 5 next 6)

(order6_7 isa order first 6 next 7)

(order7_8 isa order first 7 next 8)

(order8_9 isa order first 8 next 9)

(order9_10 isa order first 9 next 10)

(order10_11 isa order first 10 next 11)

(order11_12 isa order first 11 next 12)

(order12_13 isa order first 12 next 13)

(order13_14 isa order first 13 next 14)

(order14_15 isa order first 14 next 15)

(num1_1 isa number value 4 tasknumber task1)

(num1_2 isa number value 8 tasknumber task1)

(num1_3 isa number value 5 tasknumber task1)

(num1_4 isa number value 2 tasknumber task1)

(num1_5 isa number value (nicht benoetigt) tasknumber task1)

(num2_1 isa number value 11 tasknumber task2)

(num2_2 isa number value 4 tasknumber task2)
 (num2_3 isa number value 7 tasknumber task2)
 (num2_4 isa number value 2 tasknumber task2)
 (num2_5 isa number value (nicht benoetigt) tasknumber task2)
 (num3_1 isa number value 9 tasknumber task3)
 (num3_2 isa number value 10 tasknumber task3)
 (num3_3 isa number value 12 tasknumber task3)
 (num3_4 isa number value 3 tasknumber task3)
 (num3_5 isa number value (nicht benoetigt) tasknumber task3)
 (keyword1_1_1 isa keyword code keyword1_1_1 content (einige elemente aus einer menge auswaehlen))
 (keyword1_1_2 isa keyword code keyword1_1_2 content (3 kugeln von 5 gezogen))
 (keyword1_1_3 isa keyword code keyword1_1_3 content (2 kugeln von 5 gezogen))
 (keyword1_1_4 isa keyword code keyword1_1_4 content (2 kugeln von 4 gezogen))
 (keyword1_1_5 isa keyword code keyword1_1_5 content (5 kugeln zur auswahl))
 (keyword1_1_6 isa keyword code keyword1_1_6 content (3 von 5 kugeln ausgewaehlt))
 (keyword1_1_7 isa keyword code keyword1_1_7 content (auswahl aus einer menge von 4 kugeln))
 (keyword1_2_1 isa keyword code keyword1_2_1 content (alle elemente einer menge))
 (keyword1_2_2 isa keyword code keyword1_2_2 content (alle kugeln gezogen))
 (keyword2_1_1 isa keyword code keyword2_1_1 content (anordnung))
 (keyword2_1_2 isa keyword code keyword2_1_2 content (reihenfolge))
 (keyword2_1_3 isa keyword code keyword2_1_3 content (nacheinander))
 (keyword2_1_4 isa keyword code keyword2_1_4 content (zuerst-dann-als letztes))
 (keyword2_1_5 isa keyword code keyword2_1_5 content (zuerst-dann))
 (keyword2_1_6 isa keyword code keyword2_1_6 content (das erste mal-dann))
 (keyword2_2_1 isa keyword code keyword2_2_1 content (reihenfolge irrelevant))
 (keyword2_2_2 isa keyword code keyword2_2_2 content (gleichzeitig))
 (keyword2_2_3 isa keyword code keyword2_2_3 content (einmal-einmal))
 (keyword3_1_1 isa keyword code keyword3_1_1 content (teilmenge von gleichartigen elementen))
 (keyword3_1_2 isa keyword code keyword3_1_2 content (elemente nicht voneinander unterscheidbar))
 (keyword3_1_3 isa keyword code keyword3_1_3 content (elemente mehrfach))
 (keyword3_1_4 isa keyword code keyword3_1_4 content (sofort wieder in die urne zurueckgelegt))
 (keyword3_1_5 isa keyword code keyword3_1_5 content (direkt nach dem ziehen wieder zurueckgelegt))
 (keyword3_2_1 isa keyword code keyword3_2_1 content (alle elemente voneinander unterscheidbar))
 (keyword3_2_2 isa keyword code keyword3_2_2 content (nicht mehr in die urne zurueckgelegt))
 (keyword3_2_3 isa keyword code keyword3_2_3 content (kein element mehrfach))
 (keyword4_1 isa keyword code keyword4_1 content (anzahl von elementen die in anordnung gebracht))
 (keyword4_2 isa keyword code keyword4_2 content (menge deren n elemente in reihenfolge gebracht))
 (keyword4_3 isa keyword code keyword4_3 content (anzahl von elementen aus denen ausgewaehlt))
 (keyword4_4 isa keyword code keyword4_4 content (menge aus der ausgewaehlt))
 (keyword5_1 isa keyword code keyword5_1 content (anzahl gleichartiger elemente))
 (keyword5_2 isa keyword code keyword5_2 content (anzahl ausgewaehlter elemente))
 (keyword5_3 isa keyword code keyword5_3 content (anzahl ausgewaehlter kugeln))
 (unsolved isa status)
 (solved isa status)
 (search_solution isa status)
 (read isa status)
 (principle1 isa principle name pow)
 (principle2 isa principle name pmw)
 (principle3 isa principle name vow)
 (principle4 isa principle name vmw)
 (principle5 isa principle name kow)
 (principle6 isa principle name kmw)
 (parameter1 isa parameter name n)
 (parameter2 isa parameter name k)
 (n isa solution_element)
 (k isa solution_element)
 (pow isa solution_element)

```
(pmw isa solution_element)
(vow isa solution_element)
(vmw isa solution_element)
(kow isa solution_element)
(kmw isa solution_element)
(task1 isa tasknumber)
(task2 isa tasknumber)
(task3 isa tasknumber)
)
```

```
(p start_task
=goal>
    isa combinatoric-test
=task>
    isa task
    status unsolved
==>
!push! =task
!output! (ich beginne jetzt mit der =task -aufgabe)
)
```

```
(p end_test
=goal>
    isa combinatoric-test
    task1 =task1
    task2 =task2
    task3 =task3
=task1>
    isa task
    status solved
=task2>
    isa task
    status solved
=task3>
    isa task
    status solved
==>
!pop!
!output! (ich habe den kombinatorik-test vollstaendig bearbeitet)
)
```

```
(p prepare_solution
=goal>
    isa task
    tasknumber =tasknumber
    status unsolved
==>
=solution>
    isa solution
    tasknumber =tasknumber
=goal>
    status search_solution
    solution =solution
!output! (ich merke mir die ergebnisse zu aufgabe =tasknumber in =solution)
)
```

```
(p start_reading
=goal>
    isa task
    tasknumber =tasknumber
    status search_solution
=taskstring>
```

```

        isa taskstring
        string =string
        count 1
        tasknumber =tasknumber
-       status read
==>
!push! =taskstring
!output! (ich fange an die =goal -aufgabe zu lesen)
)

(p end_reading
=goal>
        isa task
        tasknumber =tasknumber
        status search_solution
=last-string>
        isa taskstring
        tasknumber =tasknumber
        end t
        status read
==>
=goal>
        status read
!output! (ich habe aufgabe =tasknumber jetzt ganz gelesen)
)

(p choose_principle
=goal>
        isa task
        tasknumber =tasknumber
        status read
        solution =solution
=solution>
        isa solution
        tasknumber =tasknumber
        principle nil
=principle>
        isa principle
        name =name
==>
=solution>
        principle =name
!output! (ich nehme =name in aufgabe =tasknumber an)
)

(p answer_task
=goal>
        isa task
        tasknumber =tasknumber
        status read
        solution =solution
=solution>
        isa solution
        tasknumber =tasknumber
        principle =principle
        n =n
        k =k
==>
=goal>
        status solved
!pop!
!output! (ich beantworte aufgabe =tasknumber mit =principle =n fuer n und =k fuer k)

```

```

)
(p associate_principle
=goal>
    isa taskstring
    tasknumber =tasknumber
    string =string
    count =count
-   end t
    status nil
    number nil
=keyword>
    isa keyword
    content =content
=order>
    isa order
    first =count
    next =next
=nextstring>
    isa taskstring
    count =next
    tasknumber =tasknumber
==>
=goal>
    status read
=keyword>
    tasknumber =tasknumber
!focus-on! =nextstring
!push! =keyword
!output! (ich lese =string und denke dabei an =content)
)

```

```

(p associate_parameter
=goal>
    isa taskstring
    tasknumber =tasknumber
    string =string
    count =count
-   end t
    status nil
    number =number
=keyword>
    isa keyword
    content =content
=order>
    isa order
    first =count
    next =next
=nextstring>
    isa taskstring
    count =next
    tasknumber =tasknumber
==>
=goal>
    status read
=keyword>
    tasknumber =tasknumber
    number =number
!focus-on! =nextstring
!push! =keyword
!output! (ich lese =string und denke dabei an =content)
)

```

```

(p read_task
=goal>
    isa taskstring
    string =string
    count =count
-    end t
    tasknumber =tasknumber
    status nil
=order>
    isa order
    first =count
    next =next
=nextstring>
    isa taskstring
    count =next
    tasknumber =tasknumber
==>
=goal>
    status read
!focus-on! =nextstring
!output! (ich lese =string)
)

```

```

(p stop_associate_principle
=goal>
    isa taskstring
    tasknumber =tasknumber
    string =string
    end t
    status nil
    number nil
=keyword>
    isa keyword
    content =content
==>
=goal>
    status read
=keyword>
    tasknumber =tasknumber
!focus-on! =keyword
!output! (ich lese =string und denke dabei an =content und hoere dann auf zu lesen)
)

```

```

(p stop_associate_parameter
=goal>
    isa taskstring
    tasknumber =tasknumber
    string =string
    end t
    status nil
    number =number
=keyword>
    isa keyword
    content =content
==>
=goal>
    status read
=keyword>
    tasknumber =tasknumber
    number =number
!focus-on! =keyword

```

```
!output! (ich lese =string und denke dabei an =content und hoere dann auf zu lesen)
)
```

```
(p stop_reading
=goal>
```

```
  isa taskstring
  string =string
  count =count
  end t
  tasknumber =tasknumber
  status nil
```

```
==>
```

```
=goal>
```

```
  status read
```

```
!pop!
```

```
!output! (ich lese =string und hoere dann auf zu lesen)
)
```

```
(p interpret_principle
=goal>
```

```
  isa keyword
  content =content
  tasknumber =tasknumber
  number nil
```

```
=principle1>
```

```
  isa principle
  name =name1
```

```
=principle2>
```

```
  isa principle
  name =name2
-   name =name1
```

```
=principle3>
```

```
  isa principle
  name =name3
-   name =name1
-   name =name2
```

```
==>
```

```
!pop!
```

```
!output! (=content erinnert mich an =name1 und =name2 und =name3)
)
```

```
(p keyword_parameter
=goal>
```

```
  isa keyword
  content =content
  tasknumber =tasknumber
  number =number
```

```
=parameter>
```

```
  isa parameter
  name =name
```

```
==>
```

```
=parameter>
```

```
  tasknumber =tasknumber
  number =number
```

```
=goal>
```

```
  tasknumber nil
  number nil
```

```
!focus-on! =parameter
```

```
!output! (ich denke an =name wegen =content in aufgabe =tasknumber)
)
```

```
(p n
```

```

=goal>
    isa parameter
    name n
    tasknumber =tasknumber
    number =number
=solution>
    isa solution
    tasknumber =tasknumber
==>
=goal>
    tasknumber nil
    number nil
=solution>
    n =number
!pop!
!output! (ich merke mir =number fuer n in aufgabe =tasknumber)
)

```

```

(p k
=goal>
    isa parameter
    name k
    tasknumber =tasknumber
    number =number
=solution>
    isa solution
    tasknumber =tasknumber
==>
=goal>
    tasknumber nil
    number nil
=solution>
    k =number
!pop!
!output! (ich merke mir =number fuer k in aufgabe =tasknumber)
)

```

```

(p guess_n
=goal>
    isa task
    tasknumber =tasknumber
    status read
    solution =solution
=solution>
    isa solution
    tasknumber =tasknumber
    n nil
=number>
    isa number
    value =value
    tasknumber =tasknumber
==>
=solution>
    n =value
!output! (ich rate =value fuer n in aufgabe =tasknumber)
)

```

```

(p guess_k
=goal>
    isa task
    tasknumber =tasknumber
    status read

```

```

        solution =solution
=solution>
        isa solution
        tasknumber =tasknumber
        k nil
=number>
        isa number
        value =value
        tasknumber =tasknumber
==>
=solution>
        k =value
!output! (ich rate =value fuer k in aufgabe =tasknumber)
)
(spp READ_TASK :R 0.9)
(spp STOP_READING :R 0.9)
(add-ia
(task1 string1_1 3)
(task2 string2_1 3)
(task3 string3_1 3)
(string1_1 string1_2 3)
(string1_2 string1_3 3)
(string1_3 string1_4 3)
(string1_4 string1_5 3)
(string1_5 string1_6 3)
(string1_6 string1_7 3)
(string1_7 string1_8 3)
(string1_8 string1_9 3)
(string1_9 string1_10 3)
(string1_10 string1_11 3)
(string1_11 string1_12 3)
(string1_12 string1_13 3)
(string1_13 string1_14 3)
(string1_14 string1_15 3)
(string2_1 string2_2 3)
(string2_2 string2_3 3)
(string2_3 string2_4 3)
(string2_4 string2_5 3)
(string2_5 string2_6 3)
(string2_6 string2_7 3)
(string2_7 string2_8 3)
(string2_8 string2_9 3)
(string2_9 string2_10 3)
(string2_10 string2_11 3)
(string2_11 string2_12 3)
(string2_12 string2_13 3)
(string3_1 string3_2 3)
(string3_2 string3_3 3)
(string3_3 string3_4 3)
(string3_4 string3_5 3)
(string3_5 string3_6 3)
(string3_6 string3_7 3)
(string3_7 string3_8 3)
(string3_8 string3_9 3)
(string3_9 string3_10 3)
(string3_10 string3_11 3)
(string3_11 string3_12 3)
(string3_12 string3_13 3)
(string3_13 string3_14 3)
(string3_14 string3_15 3)
(string1_1 order1_2 3)
(string1_2 order2_3 3)

```

(string1_3 order3_4 3)
(string1_4 order4_5 3)
(string1_5 order5_6 3)
(string1_6 order6_7 3)
(string1_7 order7_8 3)
(string1_8 order8_9 3)
(string1_9 order9_10 3)
(string1_10 order10_11 3)
(string1_11 order11_12 3)
(string1_12 order12_13 3)
(string1_13 order13_14 3)
(string1_14 order14_15 3)
(string2_1 order1_2 3)
(string2_2 order2_3 3)
(string2_3 order3_4 3)
(string2_4 order4_5 3)
(string2_5 order5_6 3)
(string2_6 order6_7 3)
(string2_7 order7_8 3)
(string2_8 order8_9 3)
(string2_9 order9_10 3)
(string2_10 order10_11 3)
(string2_11 order11_12 3)
(string2_12 order12_13 3)
(string3_1 order1_2 3)
(string3_2 order2_3 3)
(string3_3 order3_4 3)
(string3_4 order4_5 3)
(string3_5 order5_6 3)
(string3_6 order6_7 3)
(string3_7 order7_8 3)
(string3_8 order8_9 3)
(string3_9 order9_10 3)
(string3_10 order10_11 3)
(string3_11 order11_12 3)
(string3_12 order12_13 3)
(string3_13 order13_14 3)
(string3_14 order14_15 3)
(string1_1 keyword4_1 10)
(string1_1 keyword4_2 10)
(string1_1 keyword4_3 10)
(string1_1 keyword4_4 10)
(string1_2 keyword1_2_1 10)
(string1_2 keyword1_2_2 10)
(string1_3 keyword3_1_1 10)
(string1_3 keyword3_1_2 10)
(string1_3 keyword3_1_3 10)
(string1_3 keyword3_1_4 10)
(string1_3 keyword3_1_5 10)
(string1_4 keyword2_1_1 10)
(string1_4 keyword2_1_2 10)
(string1_4 keyword2_1_3 10)
(string1_4 keyword2_1_4 10)
(string1_4 keyword2_1_5 10)
(string1_4 keyword2_1_6 10)
(string1_6 keyword4_1 10)
(string1_6 keyword4_2 10)
(string1_6 keyword4_3 10)
(string1_6 keyword4_4 10)
(string1_7 keyword4_1 10)
(string1_7 keyword4_2 10)
(string1_7 keyword4_3 10)

(string1_7 keyword4_4 10)
(string1_8 keyword1_2_1 10)
(string1_8 keyword1_2_2 10)
(string1_9 keyword2_1_1 10)
(string1_9 keyword2_1_2 10)
(string1_9 keyword2_1_3 10)
(string1_9 keyword2_1_4 10)
(string1_9 keyword2_1_5 10)
(string1_9 keyword2_1_6 10)
(string1_12 keyword2_1_1 10)
(string1_12 keyword2_1_2 10)
(string1_12 keyword2_1_3 10)
(string1_12 keyword2_1_4 10)
(string1_12 keyword2_1_5 10)
(string1_12 keyword2_1_6 10)
(string1_13 keyword1_1_1 10)
(string1_13 keyword1_1_2 10)
(string1_13 keyword1_1_3 10)
(string1_13 keyword1_1_4 10)
(string1_13 keyword1_1_5 10)
(string1_13 keyword1_1_6 10)
(string1_13 keyword1_1_7 10)
(string1_14 keyword2_1_1 10)
(string1_14 keyword2_1_2 10)
(string1_14 keyword2_1_3 10)
(string1_14 keyword2_1_4 10)
(string1_14 keyword2_1_5 10)
(string1_14 keyword2_1_6 10)
(string1_14 keyword5_1 10)
(string1_14 keyword5_2 10)
(string1_14 keyword5_3 10)
(string1_15 keyword1_1_1 10)
(string1_15 keyword1_1_2 10)
(string1_15 keyword1_1_3 10)
(string1_15 keyword1_1_4 10)
(string1_15 keyword1_1_5 10)
(string1_15 keyword1_1_6 10)
(string1_15 keyword1_1_7 10)
(string2_1 keyword4_1 10)
(string2_1 keyword4_2 10)
(string2_1 keyword4_3 10)
(string2_1 keyword4_4 10)
(string2_2 keyword5_1 10)
(string2_2 keyword5_2 10)
(string2_2 keyword5_3 10)
(string2_3 keyword5_1 10)
(string2_3 keyword5_2 10)
(string2_3 keyword5_3 10)
(string2_4 keyword5_1 10)
(string2_4 keyword5_2 10)
(string2_4 keyword5_3 10)
(string2_5 keyword5_1 10)
(string2_5 keyword5_2 10)
(string2_5 keyword5_3 10)
(string2_9 keyword3_2_1 10)
(string2_9 keyword3_2_2 10)
(string2_9 keyword3_2_3 10)
(string2_10 keyword2_1_1 10)
(string2_10 keyword2_1_2 10)
(string2_10 keyword2_1_3 10)
(string2_10 keyword2_1_4 10)
(string2_10 keyword2_1_5 10)

(string2_10 keyword2_1_6 10)
(string2_12 keyword2_2_1 10)
(string2_12 keyword2_2_2 10)
(string2_12 keyword2_2_3 10)
(string2_13 keyword1_1_1 10)
(string2_13 keyword1_1_2 10)
(string2_13 keyword1_1_3 10)
(string2_13 keyword1_1_4 10)
(string2_13 keyword1_1_5 10)
(string2_13 keyword1_1_6 10)
(string2_13 keyword1_1_7 10)
(string3_2 keyword4_1 10)
(string3_2 keyword4_2 10)
(string3_2 keyword4_3 10)
(string3_2 keyword4_4 10)
(string3_2 keyword5_1 10)
(string3_2 keyword5_2 10)
(string3_2 keyword5_3 10)
(string3_3 keyword4_1 10)
(string3_3 keyword4_2 10)
(string3_3 keyword4_3 10)
(string3_3 keyword4_4 10)
(string3_5 keyword3_2_1 10)
(string3_5 keyword3_2_2 10)
(string3_5 keyword3_2_3 10)
(string3_6 keyword1_1_1 10)
(string3_6 keyword1_1_2 10)
(string3_6 keyword1_1_3 10)
(string3_6 keyword1_1_4 10)
(string3_6 keyword1_1_5 10)
(string3_6 keyword1_1_6 10)
(string3_6 keyword1_1_7 10)
(string3_7 keyword2_1_1 10)
(string3_7 keyword2_1_2 10)
(string3_7 keyword2_1_3 10)
(string3_7 keyword2_1_4 10)
(string3_7 keyword2_1_5 10)
(string3_7 keyword2_1_6 10)
(string3_8 keyword2_1_1 10)
(string3_8 keyword2_1_2 10)
(string3_8 keyword2_1_3 10)
(string3_8 keyword2_1_4 10)
(string3_8 keyword2_1_5 10)
(string3_8 keyword2_1_6 10)
(string3_10 keyword2_1_1 10)
(string3_10 keyword2_1_2 10)
(string3_10 keyword2_1_3 10)
(string3_10 keyword2_1_4 10)
(string3_10 keyword2_1_5 10)
(string3_10 keyword2_1_6 10)
(string3_11 keyword1_1_1 10)
(string3_11 keyword1_1_2 10)
(string3_11 keyword1_1_3 10)
(string3_11 keyword1_1_4 10)
(string3_11 keyword1_1_5 10)
(string3_11 keyword1_1_6 10)
(string3_11 keyword1_1_7 10)
(string3_12 keyword2_1_1 10)
(string3_12 keyword2_1_2 10)
(string3_12 keyword2_1_3 10)
(string3_12 keyword2_1_4 10)
(string3_12 keyword2_1_5 10)

(string3_12 keyword2_1_6 10)
(string3_13 keyword1_1_1 10)
(string3_13 keyword1_1_2 10)
(string3_13 keyword1_1_3 10)
(string3_13 keyword1_1_4 10)
(string3_13 keyword1_1_5 10)
(string3_13 keyword1_1_6 10)
(string3_13 keyword1_1_7 10)
(string3_14 keyword2_1_1 10)
(string3_14 keyword2_1_2 10)
(string3_14 keyword2_1_3 10)
(string3_14 keyword2_1_4 10)
(string3_14 keyword2_1_5 10)
(string3_14 keyword2_1_6 10)
(string3_14 keyword5_1 10)
(string3_14 keyword5_2 10)
(string3_14 keyword5_3 10)
(string3_15 keyword1_1_1 10)
(string3_15 keyword1_1_2 10)
(string3_15 keyword1_1_3 10)
(string3_15 keyword1_1_4 10)
(string3_15 keyword1_1_5 10)
(string3_15 keyword1_1_6 10)
(string3_15 keyword1_1_7 10)
(string3_15 keyword3_2_1 10)
(string3_15 keyword3_2_2 10)
(string3_15 keyword3_2_3 10)
(keyword1_1_1 principle3 10)
(keyword1_1_2 principle3 10)
(keyword1_1_3 principle3 10)
(keyword1_1_4 principle3 10)
(keyword1_1_5 principle3 10)
(keyword1_1_6 principle3 10)
(keyword1_1_7 principle3 10)
(keyword1_1_1 principle4 10)
(keyword1_1_2 principle4 10)
(keyword1_1_3 principle4 10)
(keyword1_1_4 principle4 10)
(keyword1_1_5 principle4 10)
(keyword1_1_6 principle4 10)
(keyword1_1_7 principle4 10)
(keyword1_1_1 principle5 10)
(keyword1_1_2 principle5 10)
(keyword1_1_3 principle5 10)
(keyword1_1_4 principle5 10)
(keyword1_1_5 principle5 10)
(keyword1_1_6 principle5 10)
(keyword1_1_7 principle5 10)
(keyword1_1_1 principle6 10)
(keyword1_1_2 principle6 10)
(keyword1_1_3 principle6 10)
(keyword1_1_4 principle6 10)
(keyword1_1_5 principle6 10)
(keyword1_1_6 principle6 10)
(keyword1_1_7 principle6 10)
(keyword1_2_1 principle1 10)
(keyword1_2_2 principle1 10)
(keyword1_2_1 principle2 10)
(keyword1_2_2 principle2 10)
(keyword2_1_1 principle1 10)
(keyword2_1_2 principle1 10)
(keyword2_1_3 principle1 10)

(keyword2_1_4 principle1 10)
(keyword2_1_5 principle1 10)
(keyword2_1_6 principle1 10)
(keyword2_1_1 principle2 10)
(keyword2_1_2 principle2 10)
(keyword2_1_3 principle2 10)
(keyword2_1_4 principle2 10)
(keyword2_1_5 principle2 10)
(keyword2_1_6 principle2 10)
(keyword2_1_1 principle3 10)
(keyword2_1_2 principle3 10)
(keyword2_1_3 principle3 10)
(keyword2_1_4 principle3 10)
(keyword2_1_5 principle3 10)
(keyword2_1_6 principle3 10)
(keyword2_1_1 principle4 10)
(keyword2_1_2 principle4 10)
(keyword2_1_3 principle4 10)
(keyword2_1_4 principle4 10)
(keyword2_1_5 principle4 10)
(keyword2_1_6 principle4 10)
(keyword2_2_1 principle5 10)
(keyword2_2_2 principle5 10)
(keyword2_2_3 principle5 10)
(keyword2_2_1 principle6 10)
(keyword2_2_2 principle6 10)
(keyword2_2_3 principle6 10)
(keyword3_1_1 principle2 10)
(keyword3_1_2 principle2 10)
(keyword3_1_3 principle2 10)
(keyword3_1_4 principle2 10)
(keyword3_1_5 principle2 10)
(keyword3_1_1 principle4 10)
(keyword3_1_2 principle4 10)
(keyword3_1_3 principle4 10)
(keyword3_1_4 principle4 10)
(keyword3_1_5 principle4 10)
(keyword3_1_1 principle6 10)
(keyword3_1_2 principle6 10)
(keyword3_1_3 principle6 10)
(keyword3_1_4 principle6 10)
(keyword3_1_5 principle6 10)
(keyword3_2_1 principle1 10)
(keyword3_2_2 principle1 10)
(keyword3_2_3 principle1 10)
(keyword3_2_1 principle3 10)
(keyword3_2_2 principle3 10)
(keyword3_2_3 principle3 10)
(keyword3_2_1 principle5 10)
(keyword3_2_2 principle5 10)
(keyword3_2_3 principle5 10)
(keyword4_1 parameter1 10)
(keyword4_2 parameter1 10)
(keyword4_3 parameter1 10)
(keyword4_4 parameter1 10)
(keyword5_1 parameter2 10)
(keyword5_2 parameter2 10)
(keyword5_3 parameter2 10)
)

(goal-focus goal)

C. ACT-R-Quellcode für die Situationsmodell-Strategie

```
(clear-all)
(sgp
 :ANS 0.1
 :EGS 1
 :ER T
 :RT -1
 :ERA T
 :PAS 0.1
 )
(sgp
 :DMT NIL
 )
(chunk-type combinatoric-test task1 task2 task3)
(chunk-type task tasknumber status sm solution)
(chunk-type sm tasknumber status action subject subject_all object object_all mode special)
(chunk-type solution tasknumber status principle selection order repetition n k source source_category)
(chunk-type principle selection order repetition)
(chunk-type taskstring tasknumber count end status string code)
(chunk-type interpret tasknumber base content status)
(chunk-type task_content tasknumber content number)
(chunk-type problem_element)
(chunk-type order first next)
(chunk-type status)
(chunk-type attribute category)
(chunk-type feature)
(add-dm
(goal isa combinatoric-test task1 angler task2 hunde task3 ritter)
(angler isa task tasknumber 1 status unsolved)
(hunde isa task tasknumber 2 status unsolved)
(ritter isa task tasknumber 3 status unsolved)
(pow isa principle selection select_no order ordered_yes repetition repeated_no)
(pmw isa principle selection select_no order ordered_yes repetition repeated_yes)
(vow isa principle selection select_yes order ordered_yes repetition repeated_no)
(vmw isa principle selection select_yes order ordered_yes repetition repeated_yes)
(kow isa principle selection select_yes order ordered_no repetition repeated_no)
(kmw isa principle selection select_yes order ordered_no repetition repeated_yes)
(string1_1 ISA taskstring tasknumber 1 count 1 string (Der Verein vegetarischer Angler hat 4 Mitglieder))
(string1_2 ISA taskstring tasknumber 1 count 2 string (Alle Angler haben sich verpflichtet))
(string1_3 ISA taskstring tasknumber 1 count 3 string (gefangene Fische sofort wieder zurueck in den Teich zu setzen))
(string1_4 ISA taskstring tasknumber 1 count 4 string (Eines Tages angeln die Vereinsmitglieder nacheinander))
(string1_5 ISA taskstring tasknumber 1 count 5 string (an einem Teich von 8 Quadratmetern))
(string1_6 ISA taskstring tasknumber 1 count 6 string (in dem sich 5 Fische befinden))
(string1_7 ISA taskstring tasknumber 1 count 7 string (Ein Zander ein Aal eine Forelle ein Hecht und ein Karpfen))
(string1_8 ISA taskstring tasknumber 1 count 8 string (Alle Mitglieder angeln))
(string1_9 ISA taskstring tasknumber 1 count 9 string (in absteigender Altersreihenfolge jeweils einen Fisch))
(string1_10 ISA taskstring tasknumber 1 count 10 string (Wie berechnet man die Wahrscheinlichkeit))
(string1_11 ISA taskstring tasknumber 1 count 11 string (dass per Zufall))
(string1_12 ISA taskstring tasknumber 1 count 12 string (der aelteste Angler))
(string1_13 ISA taskstring tasknumber 1 count 13 string (den Aal geangelt hat))
(string1_14 ISA taskstring tasknumber 1 count 14 string (und der zweitaelteste))
(string1_15 ISA taskstring tasknumber 1 count 15 end t string (die Forelle))
```

(string2_1 ISA taskstring tasknumber 2 count 1 string (Ein Tierheim sucht fuer 11 Hunde ein Zuhause))
(string2_2 ISA taskstring tasknumber 2 count 2 string (4 der Hunde sind Terrier))
(string2_3 ISA taskstring tasknumber 2 count 3 string (die restlichen 7 sind Mischlinge))
(string2_4 ISA taskstring tasknumber 2 count 4 string (Es melden sich 2 blonde))
(string2_5 ISA taskstring tasknumber 2 count 5 string (und 4 schwarzhaarige))
(string2_6 ISA taskstring tasknumber 2 count 6 string (Kinder))
(string2_7 ISA taskstring tasknumber 2 count 7 string (die ein Haustier suchen))
(string2_8 ISA taskstring tasknumber 2 count 8 string (Um Streit zu vermeiden))
(string2_9 ISA taskstring tasknumber 2 count 9 string (werden die Hunde per Zufall ausgelost))
(string2_10 ISA taskstring tasknumber 2 count 10 string (zuerst ziehen die schwarzhaarigen Kinder jeweils ein Los))
(string2_11 ISA taskstring tasknumber 2 count 11 string (Wie berechnet man die Wahrscheinlichkeit))
(string2_12 ISA taskstring tasknumber 2 count 12 string (dass jedes schwarzhaarige Kind))
(string2_13 ISA taskstring tasknumber 2 count 13 end t string (einen Terrier bekommt))
(string3_1 ISA taskstring tasknumber 3 count 1 string (beim 9. Koenigsturnier))
(string3_2 ISA taskstring tasknumber 3 count 2 string (beteiligen sich 10 Ritter))
(string3_3 ISA taskstring tasknumber 3 count 3 string (Der Koenig stellt fuer das Turnier 12 Pferde))
(string3_4 ISA taskstring tasknumber 3 count 4 string (Die Ritter beginnen mit verbundenen Augen))
(string3_5 ISA taskstring tasknumber 3 count 5 string (sich per Zufall ein Pferd))
(string3_6 ISA taskstring tasknumber 3 count 6 string (auszuwaehlen))
(string3_7 ISA taskstring tasknumber 3 count 7 string (Zuerst waehlt der schwerste Ritter))
(string3_8 ISA taskstring tasknumber 3 count 8 string (dann der zweitschwerste usw.))
(string3_9 ISA taskstring tasknumber 3 count 9 string (Wie berechnet man die Wahrscheinlichkeit))
(string3_10 ISA taskstring tasknumber 3 count 10 string (dass der schwerste Ritter))
(string3_11 ISA taskstring tasknumber 3 count 11 string (das groesste Pferd))
(string3_12 ISA taskstring tasknumber 3 count 12 string (der zweitschwerste))
(string3_13 ISA taskstring tasknumber 3 count 13 string (das zweitgroesste))
(string3_14 ISA taskstring tasknumber 3 count 14 string (und der drittschwerste Ritter))
(string3_15 ISA taskstring tasknumber 3 count 15 end t string (das drittgroesste Pferd bekommt))
(interpret1_1 isa interpret tasknumber 1 base string1_1 content content1_1)
(interpret1_2 isa interpret tasknumber 1 base string1_3 content content1_2)
(interpret1_3 isa interpret tasknumber 1 base string1_4 content content1_3)
(interpret1_4 isa interpret tasknumber 1 base string1_6 content content1_4)
(interpret1_5 isa interpret tasknumber 1 base string1_9 content content1_5)
(interpret1_6 isa interpret tasknumber 1 base string1_14 content content1_6)
(interpret1_7 isa interpret tasknumber 1 base string1_15 content content1_7)
(interpret2_1 isa interpret tasknumber 2 base string2_1 content content2_1)
(interpret2_2 isa interpret tasknumber 2 base string2_2 content content2_2)
(interpret2_3 isa interpret tasknumber 2 base string2_3 content content2_3)
(interpret2_4 isa interpret tasknumber 2 base string2_4 content content2_4)
(interpret2_5 isa interpret tasknumber 2 base string2_5 content content2_5)
(interpret2_6 isa interpret tasknumber 2 base string2_6 content content2_6)
(interpret2_7 isa interpret tasknumber 2 base string2_9 content content2_7)
(interpret2_8 isa interpret tasknumber 2 base string2_10 content content2_8)
(interpret2_9 isa interpret tasknumber 2 base string2_12 content content2_9)
(interpret2_10 isa interpret tasknumber 2 base string2_13 content content2_10)
(interpret3_1 isa interpret tasknumber 3 base string3_2 content content3_1)
(interpret3_2 isa interpret tasknumber 3 base string3_3 content content3_2)
(interpret3_3 isa interpret tasknumber 3 base string3_5 content content3_3)
(interpret3_4 isa interpret tasknumber 3 base string3_6 content content3_4)
(interpret3_5 isa interpret tasknumber 3 base string3_8 content content3_5)
(interpret3_6 isa interpret tasknumber 3 base string3_14 content content3_6)
(interpret3_7 isa interpret tasknumber 3 base string3_15 content content3_7)
(content1_1 isa task_content tasknumber 1 content (4 angler) number 4)
(content1_2 isa task_content tasknumber 1 content (fische werden wieder in den teich gesetzt))
(content1_3 isa task_content tasknumber 1 content (angler angeln nacheinander))
(content1_4 isa task_content tasknumber 1 content (5 fische) number 5)
(content1_5 isa task_content tasknumber 1 content (angler angeln in reihenfolge))
(content1_6 isa task_content tasknumber 1 content (2 angler) number 2)
(content1_7 isa task_content tasknumber 1 content (2 fische) number 2)
(content2_1 isa task_content tasknumber 2 content (11 hunde) number 11)

```

(content2_2 isa task_content tasknumber 2 content (4 terrier) number 4)
(content2_3 isa task_content tasknumber 2 content (7 mischlinge) number 7)
(content2_4 isa task_content tasknumber 2 content (2 blonde kinder) number 2)
(content2_5 isa task_content tasknumber 2 content (4 schwarzhaarige kinder) number 4)
(content2_6 isa task_content tasknumber 2 content (6 kinder) number 6)
(content2_7 isa task_content tasknumber 2 content (auslosen der hunde))
(content2_8 isa task_content tasknumber 2 content (zuerst ziehen schwarzhaarige kinder))
(content2_9 isa task_content tasknumber 2 content (4 kinder) number 4)
(content2_10 isa task_content tasknumber 2 content (4 terrier) number 4)
(content3_1 isa task_content tasknumber 3 content (10 ritter) number 10)
(content3_2 isa task_content tasknumber 3 content (12 pferde) number 12)
(content3_3 isa task_content tasknumber 3 content (jeder ritter bekommt ein pferd))
(content3_4 isa task_content tasknumber 3 content (ritter waehlen pferde aus))
(content3_5 isa task_content tasknumber 3 content (ritter waehlen in reihenfolge))
(content3_6 isa task_content tasknumber 3 content (3 ritter) number 3)
(content3_7 isa task_content tasknumber 3 content (3 pferde) number 3)
(action isa problem_element)
(subject isa problem_element)
(subject_all isa problem_element)
(object isa problem_element)
(object_all isa problem_element)
(mode isa problem_element)
(special isa problem_element)
(order0_1 isa order first 0 next 1)
(order1_2 isa order first 1 next 2)
(order2_3 isa order first 2 next 3)
(order3_4 isa order first 3 next 4)
(order4_5 isa order first 4 next 5)
(order5_6 isa order first 5 next 6)
(order6_7 isa order first 6 next 7)
(order7_8 isa order first 7 next 8)
(order8_9 isa order first 8 next 9)
(order9_10 isa order first 9 next 10)
(order10_11 isa order first 10 next 11)
(order11_12 isa order first 11 next 12)
(order12_13 isa order first 12 next 13)
(order13_14 isa order first 13 next 14)
(order14_15 isa order first 14 next 15)
(unsolved isa status)
(solved isa status)
(search_solution isa status)
(read isa status)
(inferred isa status)
(used isa status)
(construct_sm isa status)
(instantiate_solution isa status)
(solution_instantiated isa status)
(select_yes isa attribute category selection)
(select_no isa attribute category selection)
(ordered_yes isa attribute category order)
(ordered_no isa attribute category order)
(repeated_yes isa attribute category repetition)
(repeated_no isa attribute category repetition)
(parameter1 isa attribute category n)
(parameter2 isa attribute category k)
(selection isa feature)
(order isa feature)
(repetition isa feature)
(n isa feature)
(k isa feature)
)
(p start_task

```

```

=goal>
    isa combinatoric-test
=task>
    isa task
    status unsolved
    solution nil
==>
!push! =task
!output! (ich beginne jetzt mit der =task -aufgabe)
)

(p end_test
=goal>
    isa combinatoric-test
    task1 =task1
    task2 =task2
    task3 =task3
=task1>
    isa task
    status solved
=task2>
    isa task
    status solved
=task3>
    isa task
    status solved
==>
!pop!
!output! (ich habe den kombinatorik-test vollstaendig bearbeitet)
)

(p prepare_sm_and_solution
=goal>
    isa task
    tasknumber =tasknumber
    status unsolved
    sm nil
    solution nil
==>
=sm>
    isa sm
    tasknumber =tasknumber
=solution>
    isa solution
    tasknumber =tasknumber
=goal>
    status search_solution
    sm =sm
    solution =solution
!output! (ich merke mir die aufgabensituation in =sm und die ergebnisse von aufgabe =tasknumber in
=solution)
)

(p start_reading
=goal>
    isa task
    tasknumber =tasknumber
    status search_solution
=taskstring>
    isa taskstring
    tasknumber =tasknumber
    string =string

```

```

        count 1
-       status read
==>
!push! =taskstring
!output! (ich fange an die =goal -aufgabe zu lesen)
)

(p end_reading
=goal>
    isa task
    tasknumber =tasknumber
    status search_solution
=last_string>
    isa taskstring
    tasknumber =tasknumber
    end t
    status read
==>
=goal>
    status read
!output! (ich habe aufgabe =tasknumber jetzt ganz gelesen)
)

(p start_construct_sm
=goal>
    isa task
    tasknumber =tasknumber
    status read
=sm>
    isa sm
    tasknumber =tasknumber
==>
=goal>
    status construct_sm
=sm>
    status =status
=goal>
    status =status
!push! =sm
!output! (ich beginne mit der erstellung des situationsmodells)
)

(p start_solution
=goal>
    isa task
    tasknumber =tasknumber
    status instantiate_solution
=solution>
    isa solution
    tasknumber =tasknumber
    status nil
==>
=solution>
    status =status
=goal>
    status =status
!push! =solution
!output! (ich beginne jetzt mit der instantiierung von =solution)
)

(p infer_principle
=goal>

```

```

        isa task
        tasknumber =tasknumber
        status solution_instantiated
        solution =solution
=solution>
        isa solution
        tasknumber =tasknumber
        principle nil
        selection =selection
        order =order
        repetition =repetition
=principle>
        isa principle
        selection =selection
        order =order
        repetition =repetition
==>
=solution>
        principle =principle
!output! (ich nehme =principle in aufgabe =tasknumber an wegen =selection =order =repetition)
)

```

```

(p guess_principle
=goal>
        isa task
        tasknumber =tasknumber
        status solution_instantiated
        solution =solution
=solution>
        isa solution
        tasknumber =tasknumber
        principle nil
=principle>
        isa principle
==>
=solution>
        principle =principle
!output! (ich rate =principle in aufgabe =tasknumber)
)

```

```

(p answer_task
=goal>
        isa task
        tasknumber =tasknumber
        status solution_instantiated
        solution =solution
=solution>
        isa solution
        tasknumber =tasknumber
        principle =principle
        n =n
        k =k
==>
=goal>
        status solved
!pop!
!output! (ich beantworte aufgabe =tasknumber mit =principle =n fuer n und =k fuer k)
)

```

```

(p interpret_task
=goal>
        isa taskstring

```

```

        tasknumber =tasknumber
        count =count
        string =string
        code nil
        status nil
=code>
        isa interpret
        base =goal
        content =content
        status nil
=order>
        isa order
        first =count
        next =next
=otherstring>
        isa taskstring
        tasknumber =tasknumber
        count =next
==>
=code>
        status inferred
=goal>
        code =code
        status read
!focus-on! =otherstring
!output! (ich lese =string und codiere als =code mit =content)
)

```

```

(p read_task
=goal>
        isa taskstring
        tasknumber =tasknumber
        count =count
        string =string
        status nil
=order>
        isa order
        first =count
        next =next
=otherstring>
        isa taskstring
        tasknumber =tasknumber
        count =next
==>
=goal>
        status read
!focus-on! =otherstring
!output! (ich lese =string)
)

```

```

(p interpret_stop_reading
=goal>
        isa taskstring
        tasknumber =tasknumber
        end t
        string =string
        code nil
        status nil
=code>
        isa interpret
        base =goal
        content =content

```

```

        status nil
==>
=code>
        status inferred
=goal>
        code =code
        status read
!pop!
!output! (ich lese =string und codiere als =code mit =content und hoere dann auf zu lesen)
)

(p stop_reading
=goal>
        isa taskstring
        tasknumber =tasknumber
        end t
        string =string
        status nil
==>
=goal>
        status read
!pop!
!output! (ich lese =string und hoere dann auf zu lesen)
)

(p sm_search_action
=goal>
        isa sm
        tasknumber =tasknumber
        action nil
-       subject subject
-       subject_all subject_all
-       object object
-       object_all object_all
-       mode mode
-       special special
action>
        isa problem_element
==>
=goal>
        action action
!output! (ich starte die suche nach action fuer =goal)
)

(p sm_find_action
=goal>
        isa sm
        tasknumber =tasknumber
        action action
=interpret>
        isa interpret
        tasknumber =tasknumber
        content =content
        status inferred
==>
=goal>
        action =content
=interpret>
        status used
!output! (ich merke mir =content als action in =goal fuer aufgabe =tasknumber)
)

```

```

(p sm_search_subject
=goal>
    isa sm
    tasknumber =tasknumber
    subject nil
-   action action
-   subject_all subject_all
-   object object
-   object_all object_all
-   mode mode
-   special special

subject>
    isa problem_element
==>
=goal>
    subject subject
!output! (ich starte die suche nach subject fuer =goal)
)

(p sm_find_subject
=goal>
    isa sm
    tasknumber =tasknumber
    subject subject
=interpret>
    isa interpret
    tasknumber =tasknumber
    content =content
    status inferred
=content>
    isa task_content
    number =number
==>
=goal>
    subject =content
=interpret>
    status used
!output! (ich merke mir =content als subject in =goal fuer aufgabe =tasknumber)
)

(p sm_search_subject_all
=goal>
    isa sm
    tasknumber =tasknumber
    subject_all nil
-   action action
-   subject subject
-   object object
-   object_all object_all
-   mode mode
-   special special
subject_all>
    isa problem_element
==>
=goal>
    subject_all subject_all
!output! (ich starte die suche nach subject_all fuer =goal)
)

(p sm_find_subject_all
=goal>

```

```

        isa sm
        tasknumber =tasknumber
        subject_all subject_all
=interpret>
        isa interpret
        tasknumber =tasknumber
        content =content
        status inferred
=content>
        isa task_content
        number =number
==>
=goal>
        subject_all =content
=interpret>
        status used
!output! (ich merke mir =content als subject_all in =goal fuer aufgabe =tasknumber)
)

```

```

(p sm_search_object
=goal>
        isa sm
        tasknumber =tasknumber
        object nil
-        action action
-        subject subject
-        subject_all subject_all
-        object_all object_all
-        mode mode
-        special special
object>
        isa problem_element
==>
=goal>
        object object
!output! (ich starte die suche nach object fuer =goal)
)

```

```

(p sm_find_object
=goal>
        isa sm
        tasknumber =tasknumber
        object object
=interpret>
        isa interpret
        tasknumber =tasknumber
        content =content
        status inferred
=content>
        isa task_content
        number =number
==>
=goal>
        object =content
=interpret>
        status used
!output! (ich merke mir =content als object in =goal fuer aufgabe =tasknumber)
)

```

```

(p sm_search_object_all
=goal>
        isa sm

```

```

        tasknumber =tasknumber
        object_all nil
-       action action
-       subject subject
-       subject_all subject_all
-       object object
-       mode mode
-       special special
object_all>
        isa problem_element
==>
=goal>
        object_all object_all
!output! (ich starte die suche nach object_all fuer =goal)
)

(p sm_find_object_all
=goal>
        isa sm
        tasknumber =tasknumber
        object_all object_all
=interpret>
        isa interpret
        tasknumber =tasknumber
        content =content
        status inferred
=content>
        isa task_content
        number =number
==>
=goal>
        object_all =content
=interpret>
        status used
!output! (ich merke mir =content als object_all in =goal fuer aufgabe =tasknumber)
)

(p sm_search_mode
=goal>
        isa sm
        tasknumber =tasknumber
        mode nil
-       action action
-       subject subject
-       subject_all subject_all
-       object object
-       object_all object_all
-       special special
mode>
        isa problem_element
==>
=goal>
        mode mode
!output! (ich starte die suche nach mode fuer =goal)
)

(p sm_find_mode
=goal>
        isa sm
        tasknumber =tasknumber
        mode mode
=interpret>

```

```

        isa interpret
        tasknumber =tasknumber
        content =content
        status inferred
==>
=goal>
    mode =content
=interpret>
    status used
!output! (ich merke mir =content als mode in =goal fuer aufgabe =tasknumber)
)

(p sm_search_special
=goal>
    isa sm
    tasknumber =tasknumber
    special nil
-    action action
-    subject subject
-    subject_all subject_all
-    object object
-    object_all object_all
-    mode mode
special>
    isa problem_element
==>
=goal>
    special special
!output! (ich starte die suche nach special fuer =goal)
)

(p sm_find_special
=goal>
    isa sm
    tasknumber =tasknumber
    special special
=interpret>
    isa interpret
    tasknumber =tasknumber
    content =content
    status inferred
==>
=goal>
    special =content
=interpret>
    status used
!output! (ich merke mir =content als special in =goal fuer aufgabe =tasknumber)
)

(p sm_constructed
=goal>
    isa sm
    tasknumber =tasknumber
    status nil
==>
=goal>
    status instantiate_solution
!pop!
!output! (ich beende die konstruktion von =goal)
)

(p selection

```

```

=goal>
    isa solution
    tasknumber =tasknumber
    selection nil
    source nil
    source_category nil
=sm>
    isa sm
    tasknumber =tasknumber
    action =action
==>
=goal>
    source =action
    source_category action
!output! (um etwas ueber selection herauszufinden nutze ich =action von =sm)
)

```

```

(p instantiate_selection
=goal>
    isa solution
    tasknumber =tasknumber
    selection nil
    source =source
    source_category action
=attribute>
    isa attribute
    category selection
==>
=goal>
    selection =attribute
    source nil
    source_category nil
!output! (ich nehme =attribute fuer selection in aufgabe =tasknumber an)
)

```

```

(p order
=goal>
    isa solution
    tasknumber =tasknumber
    order nil
    source nil
    source_category nil
=sm>
    isa sm
    tasknumber =tasknumber
    mode =mode
==>
=goal>
    source =mode
    source_category mode
!output! (um etwas ueber order herauszufinden nutze ich =mode von =sm)
)

```

```

(p instantiate_order
=goal>
    isa solution
    tasknumber =tasknumber
    order nil
    source =source
    source_category mode
=attribute>
    isa attribute

```

```

category order
==>
=goal>
    order =attribute
    source nil
    source_category nil
!output! (ich nehme =attribute fuer order in aufgabe =tasknumber an)
)

(p repetition
=goal>
    isa solution
    tasknumber =tasknumber
    repetition nil
    source nil
    source_category nil
=sm>
    isa sm
    tasknumber =tasknumber
    special =special
==>
=goal>
    source =special
    source_category special
!output! (um etwas ueber repetition herauszufinden nutze ich =special von =sm)
)

(p instantiate_repetition
=goal>
    isa solution
    tasknumber =tasknumber
    repetition nil
    source =source
    source_category special
=attribute>
    isa attribute
    category repetition
==>
=goal>
    repetition =attribute
    source nil
    source_category nil
!output! (ich nehme =attribute fuer repetition in aufgabe =tasknumber an)
)

(p n
=goal>
    isa solution
    tasknumber =tasknumber
    n nil
    source nil
    source_category nil
=sm>
    isa sm
    tasknumber =tasknumber
    object_all =object_all
==>
=goal>
    source =object_all
    source_category object_all
!output! (um etwas ueber n herauszufinden nutze ich =object_all von =sm)
)

```

```

(p instantiate_n
=goal>
    isa solution
    tasknumber =tasknumber
    n nil
    source =source
    source_category object_all
=attribute>
    isa attribute
    category n
=source>
    isa task_content
    content =content
    number =number
==>
=goal>
    n =number
    source nil
    source_category nil
!output! (ich nehme =number fuer n in aufgabe =tasknumber an)
)

(p k
=goal>
    isa solution
    tasknumber =tasknumber
    k nil
    source nil
    source_category nil
=sm>
    isa sm
    tasknumber =tasknumber
    object =object
==>
=goal>
    source =object
    source_category object
!output! (um etwas ueber k herauszufinden nutze ich =object von =sm)
)

(p instantiate_k
=goal>
    isa solution
    tasknumber =tasknumber
    k nil
    source =source
    source_category object
=attribute>
    isa attribute
    category k
=source>
    isa task_content
    content =content
    number =number
==>
=goal>
    k =number
    source nil
    source_category nil
!output! (ich nehme =number fuer k in aufgabe =tasknumber an)
)

```

```

(p guess_selection
=goal>
    isa solution
    tasknumber =tasknumber
    selection nil
    source nil
=sm>
    isa sm
    tasknumber =tasknumber
    action nil
=attribute>
    isa attribute
    category selection
==>
=goal>
    selection =attribute
!output! (ich rate =attribute als selection in aufgabe =tasknumber)
)

```

```

(p guess_order
=goal>
    isa solution
    tasknumber =tasknumber
    order nil
    source nil
=sm>
    isa sm
    tasknumber =tasknumber
    mode nil
=attribute>
    isa attribute
    category order
==>
=goal>
    order =attribute
!output! (ich rate =attribute als order in aufgabe =tasknumber)
)

```

```

(p guess_repetition
=goal>
    isa solution
    tasknumber =tasknumber
    repetition nil
    source nil
=sm>
    isa sm
    tasknumber =tasknumber
    special nil
=attribute>
    isa attribute
    category repetition
==>
=goal>
    repetition =attribute
!output! (ich rate =attribute als repetition in aufgabe =tasknumber)
)

```

```

(p guess_n
=goal>
    isa solution
    tasknumber =tasknumber

```

```

        n nil
        source nil
=sm>
        isa sm
        tasknumber =tasknumber
        object_all nil
=task_content>
        isa task_content
        tasknumber =tasknumber
        number =number
==>
=goal>
        n =number
!output! (ich rate =number als n in aufgabe =tasknumber)
)

```

```

(p guess_k
=goal>
        isa solution
        tasknumber =tasknumber
        k nil
        source nil
=sm>
        isa sm
        tasknumber =tasknumber
        object nil
=task_content>
        isa task_content
        tasknumber =tasknumber
        number =number
==>
=goal>
        k =number
!output! (ich rate =number als k in aufgabe =tasknumber)
)

```

```

(p guess_n_object_all
=goal>
        isa solution
        tasknumber =tasknumber
        n nil
        source object_all
        source_category object_all
=task_content>
        isa task_content
        tasknumber =tasknumber
        number =number
==>
=goal>
        n =number
        source nil
        source_category nil
!output! (ich rate =number als n in aufgabe =tasknumber)
)

```

```

(p guess_k_object
=goal>
        isa solution
        tasknumber =tasknumber
        k nil
        source object
        source_category object

```

```

=task_content>
  isa task_content
  tasknumber =tasknumber
  number =number
==>
=goal>
  k =number
  source nil
  source_category nil
!output! (ich rate =number als k in aufgabe =tasknumber)
)

(p stop_instantiate_solution
=goal>
  isa solution
  tasknumber =tasknumber
  status nil
  selection =selection
  order =order
  repetition =repetition
  n =n
  k =k
==>
=goal>
  status solution_instantiated
!pop!
!output! (ich habe alle informationen in =goal instantiiert)
)
(spp INFER_PRINCIPLE :B 0 :A 0)
(spp INTERPRET_STOP_READING :B 1.0)
(spp INTERPRET_TASK :B 1.0)
(spp READ_TASK :B 1.0 :R 0.7)
(spp STOP_READING :B 1.0 :R 0.7)
(spp SM_CONSTRUCTED :R 0.5 :Q 1.0)
(spp GUESS_PRINCIPLE :Q 0.8)
(add-ia
(subject_all interpret1_1 3)
(special interpret1_2 3)
(action interpret1_3 3)
(object_all interpret1_4 3)
(mode interpret1_5 3)
(subject interpret1_6 3)
(object interpret1_7 3)
(object_all interpret2_1 3)
(subject_all interpret2_6 3)
(special interpret2_7 3)
(action interpret2_8 3)
(subject interpret2_9 3)
(object interpret2_10 3)
(subject_all interpret3_1 3)
(object_all interpret3_2 3)
(special interpret3_3 3)
(action interpret3_4 3)
(mode interpret3_5 3)
(subject interpret3_6 3)
(object interpret3_7 3)
(content1_2 repeated_yes 3)
(content1_3 select_yes 3)
(content1_4 n 3)
(content1_5 ordered_yes 3)
(content1_7 k 3)
(content2_1 n 3)

```

(content2_7 repeated_no 3)
(content2_8 select_yes 3)
(content2_10 k 3)
(content3_2 n 3)
(content3_3 repeated_no 3)
(content3_4 select_yes 3)
(content3_5 ordered_yes 3)
(content3_7 k 3)
)

(goal-focus goal)

D. Kombinatorik-Aufgabenkategorien in HYPERCOMB

1. Permutation ohne Wiederholung

Bei Permutationsaufgaben geht es um die Anzahl der Möglichkeiten, alle Elemente einer Menge in eine Anordnung oder Reihenfolge zu bringen ("Permutationen").

Können alle Elemente der Menge voneinander unterschieden werden, so spricht man von Permutation ohne Wiederholung.

Die Anzahl A möglicher Permutationen ohne Wiederholung lässt sich durch folgende Formel berechnen: $A = n!$

Dabei bezeichnet n die Anzahl von Elementen, die in eine Anordnung gebracht werden sollen ($n! = n * (n-1) * (n-2) \dots * 1$).

Ein einfaches Urnenbeispiel

In einer Urne befinden sich eine gelbe, eine rote, eine grüne und eine blaue Kugel. Nacheinander werden vier Kugeln gezogen, die nicht mehr in die Urne zurückgelegt werden. Wie groß ist die Wahrscheinlichkeit zuerst die blaue, dann die rote, dann die gelbe und als letztes die grüne Kugel zu ziehen?

In der geschilderten Beispielaufgabe geht es um eine bestimmte Anordnung von 4 Kugeln. Dies ist die Menge, deren n Elemente in eine Reihenfolge gebracht werden ($n = 4$). Setzt man diese Zahl in die Formel für Permutation ohne Wiederholung ein, also $A = n!$, so ergeben sich $4! = 4 \times 3 \times 2 \times 1 = 24$ Permutationen. Die Wahrscheinlichkeit für eine bestimmte Reihenfolge beträgt damit $1/24 = 4,17\%$.

2. Permutation mit Wiederholung

Bei Permutationsaufgaben geht es um die Anzahl der Möglichkeiten, Elemente einer Menge in eine bestimmte Anordnung oder Reihenfolge zu bringen ("Permutationen").

Gibt es in der anzuordnenden Menge eine Teilmenge von gleichartigen Elementen, die nicht voneinander unterschieden werden, so spricht man von Permutation mit Wiederholung.

Die Anzahl A möglicher Permutationen mit Wiederholung lässt sich durch folgende Formel berechnen: $A = n! / k!$

Dabei bezeichnet n die Anzahl von Elementen, die in eine Anordnung gebracht werden sollen, und k die Anzahl der gleichartigen Elemente ($n! = n * (n-1) * (n-2) \dots * 1$).

Ein einfaches Urnenbeispiel

In einer Urne befinden sich vier rote, eine grüne und eine blaue Kugel. Nacheinander werden alle Kugeln gezogen. Wie groß ist die Wahrscheinlichkeit zuerst alle roten, dann die grüne und dann die blaue Kugel zu ziehen?

In der geschilderten Beispielaufgabe geht es um eine bestimmte Anordnung von 6 Kugeln. Dies ist die Menge, deren n Elemente in eine Reihenfolge gebracht werden ($n = 6$). 4 dieser Kugeln sind rot. Dies sind die gleichartigen Elemente ($k = 4$). Setzt man diese Zahl in die Formel für Permutation mit Wiederholung ein, also $A = n! / k!$, so ergeben sich $6! / 4! = 30$ Permutationsmöglichkeiten. Die Wahrscheinlichkeit für eine bestimmte Reihenfolge (erst alle roten, dann die grüne und dann die blaue Kugel) beträgt demnach $1/30 = 3,3\%$.

3. Variation ohne Wiederholung

Bei Variationsaufgaben geht es um die Anzahl der Möglichkeiten, aus einer Menge von Elementen einige Elemente in einer bestimmten Reihenfolge auszuwählen ("Variationen").

Kann in der Menge der ausgewählten Elemente kein Element mehrfach auftreten, so spricht man von Variation ohne Wiederholung.

Die Anzahl A möglicher Variationen ohne Wiederholung läßt sich durch folgende Formel berechnen:

$$A = n! / (n - k)!$$

Dabei bezeichnet n die Anzahl von Elementen, aus denen ausgewählt werden kann, und k die Anzahl der ausgewählten Elemente ($n! = n * (n-1) * (n-2) \dots * 1$).

Ein einfaches Urnenbeispiel

In einer Urne befinden sich eine weiße, eine gelbe, eine rote, eine grüne und eine blaue Kugel. Nacheinander werden drei Kugeln gezogen, die nicht mehr in die Urne zurückgelegt werden. Wie groß ist die Wahrscheinlichkeit, daß zuerst die blaue, dann die weiße und dann die gelbe Kugel gezogen wird?

In der geschilderten Situation stehen 5 Kugeln zur Auswahl. Dies ist die Menge, aus der ausgewählt wird ($n = 5$). Da die Wahrscheinlichkeit gefragt ist, zuerst die blaue, dann die weiße und dann die gelbe Kugel zufällig auszuwählen, ist die Reihenfolge der Auswahl von Bedeutung. Aus der Gruppe von 5 Kugeln werden drei ausgewählt. Die Anzahl ausgewählter Kugeln beträgt also $k = 3$. Setzt man diese Zahlen in die Formel für Variation ohne Wiederholung ein, also $A = n! / (n-k)!$, so ergeben sich $5! / (5 - 3)! = 60$ Variationen. Die Wahrscheinlichkeit für eine dieser Möglichkeiten (zuerst die blaue, dann die weiße und dann die gelbe Kugel auszuwählen) beträgt $1/60 = 1,7\%$.

4. Variation mit Wiederholung

Bei Variationsaufgaben geht es um die Anzahl der Möglichkeiten, aus einer Menge von Elementen einige Elemente in einer bestimmten Reihenfolge auszuwählen ("Variationen").

Können in der Menge der ausgewählten Elemente Elemente mehrfach auftreten, so spricht man von Variation mit Wiederholung.

Die Anzahl A möglicher Variationen mit Wiederholung läßt sich durch folgende Formel berechnen:

$$A = n^k$$

Dabei bezeichnet n die Anzahl von Elementen, aus denen ausgewählt werden kann, und k die Anzahl der ausgewählten Elemente.

Ein einfaches Urnenbeispiel

In einer Urne befindet sich eine weiße, eine gelbe, eine rote, eine grüne und eine blaue Kugel. Nacheinander werden drei Kugeln gezogen, die jeweils sofort wieder in die Urne zurückgelegt werden. Wie groß ist die Wahrscheinlichkeit zuerst die rote, dann die blaue und dann wieder die rote Kugel zu ziehen?

In der geschilderten Situation stehen 5 Kugeln zur Auswahl. Dies ist die Menge, aus der ausgewählt wird ($n = 5$). Es werden insgesamt drei Kugeln gezogen, also geht es um eine Auswahl von 3 Kugeln aus 5 Kugeln. Die Anzahl ausgewählter Kugeln beträgt also $k = 3$. Da die Wahrscheinlichkeit gefragt ist, das erste Mal die rote, das zweite Mal die blaue und dann wieder die rote Kugeln zu ziehen, ist die Reihenfolge der Auswahl von Bedeutung. Setzt man diese Zahlen in die Formel für Variation mit Wiederholung ein, also $A = n^k$, so ergeben sich 53 Variationen. Die Wahrscheinlichkeit für eine dieser Möglichkeiten (erst rote, dann blaue, dann wieder rote Kugel) beträgt $1/125 = 0,8\%$.

5. Kombination ohne Wiederholung

Bei Kombinationsaufgaben geht es um die Anzahl der Möglichkeiten, aus einer Menge von Elementen einige Elemente auszuwählen, wobei die Reihenfolge der ausgewählten Elemente irrelevant ist ("Kombinationen").

Kann in der Menge der ausgewählten Elemente kein Element mehrfach auftreten, so spricht man von Kombination ohne Wiederholung.

Die Anzahl A möglicher Kombinationen ohne Wiederholung läßt sich durch folgende Formel berechnen: $A = n! / [(n - k)! \cdot k!]$

Dabei bezeichnet n die Anzahl von Elementen, aus denen ausgewählt werden kann, und k die Anzahl der ausgewählten Elemente ($n! = n \cdot (n-1) \cdot (n-2) \dots \cdot 1$).

Ein einfaches Urnenbeispiel

In einer Urne befinden sich eine weiße, eine gelbe, eine rote, eine blaue und eine grüne Kugel. Es werden gleichzeitig zwei Kugeln gezogen. Wie groß ist die Wahrscheinlichkeit, die rote und die blaue Kugel zu ziehen?

In der geschilderten Beispielaufgabe geht es um eine Auswahl aus einer Menge von 5 Kugeln. Dies ist die Menge, aus der ausgewählt wird ($n = 5$). Es ist die Wahrscheinlichkeit gefragt, aus dieser Menge zufällig die rote und die blaue Kugel auszuwählen, wobei es irrelevant ist, in welcher Reihenfolge die Kugeln ausgewählt werden. Die Anzahl ausgewählter Elemente beträgt also $k = 2$. Setzt man diese Zahlen in die Formel für Kombination ohne Wiederholung ein, also $A = n! / (n-k)! \cdot k!$, so ergeben sich $5! / (5-2)! \cdot 2! = 10$ Kombinationen. Die Wahrscheinlichkeit für eine dieser Möglichkeiten (nämlich die rote und die blaue Kugel auszuwählen) beträgt demnach $1/10 = 10\%$.

6. Kombination mit Wiederholung

Bei Kombinationsaufgaben geht es um die Anzahl der Möglichkeiten, aus einer Menge von Elementen einige Elemente auszuwählen, wobei die Reihenfolge der ausgewählten Elemente irrelevant ist ("Kombinationen").

Können in der Menge der ausgewählten Elemente Elemente mehrfach auftreten, so spricht man von Kombination mit Wiederholung.

Die Anzahl A möglicher Kombinationen mit Wiederholung läßt sich durch folgende Formel berechnen:

$$A = (n + k - 1)! / [(n - 1)! * k!]$$

Dabei bezeichnet n die Anzahl von Elementen, aus denen ausgewählt werden kann, und k die Anzahl der ausgewählten Elemente (mit $n! = n * (n-1) * (n-2) \dots * 1$).

Ein einfaches Urnenbeispiel

In einer Urne befinden sich eine rote, eine weiße, eine gelbe und eine grüne Kugel. Es werden zwei Kugeln gezogen, die aber jeweils direkt nach dem Ziehen wieder zurückgelegt werden. Wie groß ist die Wahrscheinlichkeit einmal die gelbe und einmal die grüne Kugel zu ziehen?

In der geschilderten Beispielaufgabe geht es um eine Auswahl aus einer Menge von 4 Kugeln. Dies ist die Menge, aus der ausgewählt wird ($n = 4$). Es ist die Wahrscheinlichkeit gefragt, aus dieser Menge zufällig die gelbe und die grüne Kugel auszuwählen, wobei die Reihenfolge der Auswahl irrelevant ist und nach jeder Ziehung zurückgelegt wird. Die Anzahl ausgewählter Elemente beträgt also $k = 2$. Setzt man diese Zahlen in die Formel für Kombination mit Wiederholung ein, also $A = (n+k-1)!/(n-1)!k!$, so ergeben sich $(4+2-1)!/(4-1)!2! = 10$ Kombinationen. Die Wahrscheinlichkeit für eine dieser Möglichkeiten (nämlich einmal die gelbe Kugel und einmal die grüne Kugel auszuwählen) berechnet sich etwas anders als bei den fünf anderen Aufgabentypen, weil die verschiedenen Kombinationen mit Wiederholung nicht mit gleicher Häufigkeit auftreten (siehe nächste Seite).

Würden die 10 möglichen Auswahlen von 2 aus 4 Kugeln in unserem Beispiel gleich häufig auftreten, so wäre die Wahrscheinlichkeit für jede Auswahl $1/10 = 10\%$. Diese Gleichhäufigkeit der 10 Auswahlmöglichkeiten ist jedoch nicht gegeben: Da insgesamt 4 verschiedenfarbige Kugeln zur Auswahl stehen, sind unter den 10 Möglichkeiten für die Auswahl von zwei Kugeln mit Wiederholung 4 Fälle, in denen beide Kugeln die gleiche Farbe haben. In den restlichen 6 Fällen haben beide Kugeln eine unterschiedliche Farbe. Um zwei Kugeln von gleicher Farbe auszuwählen (z.B. rot-rot) gibt es jeweils genau eine Möglichkeit: Die erste ausgewählte Kugel muß rot sein und die zweite ausgewählte Kugel muß ebenfalls rot sein. Um zwei Kugeln von unterschiedlicher Farbe auszuwählen (z.B. gelb-grün) gibt es hingegen zwei Möglichkeiten: Entweder wird erst eine gelbe und dann eine grüne Kugel ausgewählt, oder es wird erst eine grüne und dann eine gelbe Kugel ausgewählt! Auswahlen von Kugeln mit gleicher Farbe sind also jeweils halb so wahrscheinlich wie Auswahlen von Kugeln mit unterschiedlicher Farbe. Da unter den 10 möglichen Kombinationen also 4 Fälle sind, die halb so wahrscheinlich sind wie die restlichen 6 Fälle (unter denen sich auch der günstige Fall befindet, nämlich gelb-grün), müssen diese 4 Fälle bei der Wahrscheinlichkeitsberechnung mit $1/2$ gewichtet werden. Es ergibt sich eine Wahrscheinlichkeit von $1 / (6 + (4 \times 1/2)) = 1/8 = 12,5\%$ für die Auswahl einer gelben und einer grünen Kugel.