SOAR
Eine Kognitive Architektur

AI Tools, Saarbrücken, Wintersemester 06/07

Peter Paul Kaczmarczyk,
66450 Bexbach, Peter.kaczmarczyk@web.de

Dozenten:

Dr. Michael Kipp,
Dr. Alassane Ndiaye,
Dr. Dominik Heckmann,
Michael Feld

Abstract

Das menschliche Denken wird schon lange von Wissenschaftlern untersucht. Soar ist als eine kognitive Architektur in der Lage menschliches Denken und Schließen zu simulieren. Bei diesem simuliertem Denkprozess treten dieselben Probleme auf, wie sie sich im Gedankengang eines Menschen stellen. Soar stellt Möglichkeiten und Lösungskonzepte vor, wie man mit solchen Problemen umgehen kann. Wissenschaftler aber auch große Militärprojekte profitieren von der kognitiven Architektur, die Soar bereitstellt, um das menschliche Denken so detailliert darstellen und simulieren zu können.

Inhaltsverzeichnis:

1 EINLEITUNG 3

2 KOGNITIVE ARCHITEKTUR 4

3 PRODUKTIONSSYSTEM 4

4 SOAR 4

5 PRODUKTIONSREGELN 4

6 ZUSTAND 5

7 OPERATOREN 6

8 ARBEITSSPEICHER 7

9 LANGZEITSPEICHER 9

9.1 PROZEDURALER LANGZEITSPEICHER 9
9.2 SEMANTISCHER LANGZEITSPEICHER 9
9.3 EPISODISCHER LANGZEITSPEICHER 10

10 SACKGASSEN 10

11 LERNMECHANISMEN 10

11.1 CHUNKING 10
11.2 REINFORCEMENT LEARNING 11
11.3 EPISODIC LEARNING 11

12 I/O-INTERFACE 11

13 ENTSCHEIDUNGSZYKLUS 12

14 UNIFIED THEORIES OF COGNITION 13

15 BEISPIELANWENDUNGEN VON SOAR 13

16 ZUSAMMENFASSUNG 14

17 REFERENZEN 16

1 Einleitung

Viele Wissenschaftler beschäftigen sich mit dem menschlichen Denken. Doch obwohl dieses Gebiet schon lange erforscht wird, gibt es noch kein Modell, das das menschliche Denken beschreiben kann.

Einige Theorien versuchen die einzelnen Eigenschaften des menschlichen Denkens zu beschreiben. Für diese Beschreibung werden Programme benutzt, die so handeln sollen, wie ein Mensch denken würde. Wenn diese Programme Ergebnisse erzeugen, die denen von Menschen ähneln, wird versucht Schlüsse auf die Denkprozesse beim Menschen aufzustellen.

Soar ist eine kognitive Architektur, die die Möglichkeit bietet, das menschliche Denken und Schließen zu simulieren. Die Simulation ist so präzise, dass bei diesem simulierten Denkprozess dieselben Probleme auftreten, wie sie sich im Gedankengang eines Menschen stellen. Soar stellt Möglichkeiten und Lösungskonzepte bereit mit solchen Problemen umzugehen. Wissenschaftler aber auch große Militärprojekte profitieren von dieser kognitiven Architektur, die Soar bereitstellt, um das menschliche Denken so detailliert darstellen und simulieren zu können. [SOAR]

Die Anwender von Soar sind sehr vielseitig, weil Soar nur eine Architektur bereitstellt, auf der die eigentlichen Programme laufen sollen. Alle Anwender haben jedoch die Kognition als verbindende Wissenschaft gemeinsam. Die Kognition ist ein Informationsverarbeitungsprozess, indem Neues gelernt und Wissen verarbeitet wird. Die Kognitionswissenschaft ist ein interdisziplinäres Unternehmen aus der Kognitionspsychologie, der Neurowissenschaft, der Linguistik, der Philosophie und der Informatik.

Jede Wissenschaft legt dabei jeweils einen anderen Focus. So versucht die Kognitions-Psychologie das Denken des Menschen zu beschreiben, indem sie Programme entwirft, die menschliches Denken simulieren. Dabei wird mit Soar nicht in erste Linie versucht die Probleme selbst zu lösen. Mit Hilfe von Soar wird versucht die Probleme so zu lösen, wie man denkt ein Mensch würde diese Probleme bearbeiten. Wenn man feststellt, dass Soar das Ergebnis ebenso erzeugt hat, wie ein Mensch es getan hätte, dann wird versucht Aussagen über die Denkweise des Menschen zu schließen.

Als Wissenschaft mit ganz anderem Focus wird hier die Informatik erwähnt. Speziell die KI Entwickler versuchen durch Simulation der Verhaltensweisen des Menschen konkrete Probleme effizienter zu lösen. Dabei geht es hauptsächlich um das Problem und die Lösung und weniger auf welche Weise man diese erreicht hat, oder ob sie wirklich so als Denkprozess im menschlichen Verstand entstanden wäre.

2 Kognitive Architektur

Eine kognitive Architektur ist eine Computerarchitektur, die nichtdeterministische, vielfache Interferenzprozesse, wie man sie in neuronalen Netzen findet einschließt. Kognitive Architekturen versuchen das menschliche Gehirn nach zu modellieren. Dabei unterscheiden sie sich bei der Verarbeitung von Aufgaben von Einzelprozessorcomputern. Sie simulieren parallele Abläufe, wie sie bei der Verarbeitung im menschlichen Gehirn gemessen werden können.

3 Produktionssystem

Ein Produktionssystem besteht aus einer Sammlung von Produktionen, einem Arbeitsspeicher und einem Algorithmus bekannt als „forward chaining". Der Arbeitsspeicher beinhaltet alle Tatsachen, die zurzeit dem Produktionssystem bekannt sind. Der Algorithmus benutzt die Produktionen, um neue Tatsachen im Arbeitsspeicher aus alten zu erzeugen. Die Produktionen werden auch Regeln genannt. Eine Regel wird berechtigt, um "zu schießen", wenn all ihre Bedingungen zutreffen. Dabei wird die Liste aller benötigten Speicherelemente mit dem Satz von Elementen verglichen, der zurzeit im Arbeitsspeicher vorhanden ist. Eine Konfliktlösungsstrategie bestimmt, welche Regel als nächstes schießt, wenn mehrere Regeln zur Verfügung stehen. Diese Liste von möglichen Regeln wird als Konfliktsatz bezeichnet.

4 Soar

Der Name Soar ist eine Abkürzung für **S**tate, **O**perator **A**nd **R**esult. Im Groben stellt der Name die Hauptteile von Soar dar, der Zustand auf dem Operatoren neue Ergebnisse erzeugen. Soar ist eine kognitive Architektur, die mit Hilfe von einem Produktionssystem, in der Lage ist menschliche Denkprozesse nachzuahmen.

Im Folgenden gehen wir tiefer in die eigentliche Architektur von Soar ein. Soar besteht aus Produktionsregeln, Zuständen, Operatoren, einem Speicher, einem Lernmechanismus, dem I/O-Interface und zuletzt dem Entscheidungszyklus. Die einzelnen Bestandteile werden nun näher untersucht.

5 Produktionsregeln

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

Jede Regel beginnt mit "sp": "Soar production". Danach folgt der Körper der Regel. Der Körper ist umschlossen mit den geschweiften Klammern, "{" … "}". Der Körper enthält den Regelnamen, gefolgt von mindestens einer Bedingung, dem Pfeilsymbol und mindestens einer Action.

Eine Bedingung testet das Vorhandensein von Daten in einem speziellen Teil vom Speicher, dem Arbeitsspeicher oder auch Working Memory (WM). Wenn alle Bedingungen zutreffen, feuert die Regel. Die Regel ist also erfüllt und wird ausgelöst. Die Aktionen der Regel werden ausgeführt.

Jede Aktion erzeugt oder verändert das WM. Sie erzeugt neue Speicherelemente oder löscht alte wieder heraus.

6 Zustand

Mit den Produktionsregeln werden Speicherobjekte verändert. Wie Objekte in Soar modelliert werden, betrachten wir nun näher.

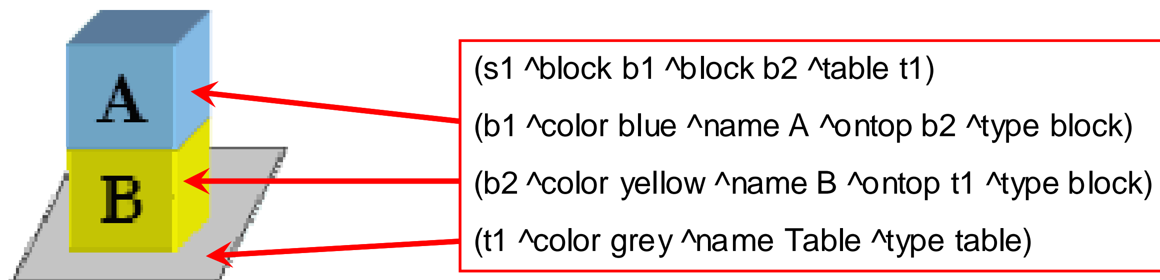

Abbildung 1 : Repräsentation eines Zustandes

Die Repräsentation in Abbildung 1 eines Zustands ist eher objektorientiert. Hier kann man erkennen, wie man sich die verschiedenen Objekte vorstellen kann. Soar nimmt diese einzelnen Objekte und versieht sie zusammen mit ihren Attributen zu einem vernetzen Graphen.

Im WM werden einzelne Wissensobjekte in einem Zustand zusammengefasst. Ein Zustand enthält alle Informationen über die momentane Situation. In Soar ist das ganze Wissen in einer Graphstruktur organisiert. Der Zustand ist die Wurzel des Graphen, von der aus man alle Objekte erreichen kann. Jedes Wissenselement ist entweder direkt oder indirekt mit solch einem Zustand verbunden.

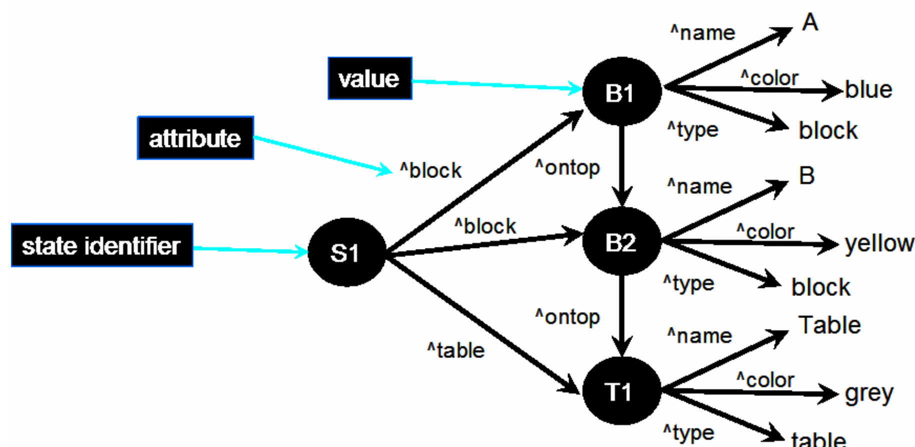

Abbildung 2 : Repräsentation eines Zustandes in Soar

Abbildung 2 gibt stellt den Zustand aus Abbildung 1 dar, wie er in Soar repräsentiert wird. Der Zustand besitzt Attribute mit Werten. Die Werte können wiederum Attribute mit Werten besitzen. Jedes so modellierte Objekt kann weiter verzweigen und mit anderen Objekten verbunden sein.

7 Operatoren

Zusätzlich zu den normalen Produktionsregeln können Regeln in Soar spezielle Aktionen erzeugen. Solch eine Aktion erzeugt einen Operator als Objekt im WM. Operatoren sind jedoch keine festen Speicherelemente. Operatoren können von Regeln in Soar nur vorgeschlagen werden. Danach muss durch einen Entscheidungsalgorithmus bestimmt werden, welcher Operator ausgewählt wird. Durch dieses Verfahren versucht man den menschlichen Gedankengang nachzuahmen. Ein Mensch kann zwar viele Ideen entwickeln, was er als nächstes machen will, danach muss er sich aber für eine dieser Ideen entscheiden und ihre Aktionen ausführen.

```
sp {propose*pick
(state <s> ^type state
           - ^holding
             ^block <b>)
-->
(<s> ^operator <o> +)
(<o> ^name pick
       ^obj <b>)}
```

Abbildung 3 : Regel um einen Operator vorzuschlagen

Eine Regel bietet einen Operator an, indem sie den Operator mit einer Präferenz versieht. In Soar wird eine Präferenz mit einem „+“ Zeichen erzeugt. Dieser Operator ist somit ein Kandidat für die nächste Phase im Soarzyklus, der Selektionsphase.

In unserem Beispiel würde die Regel aus Abbildung 3 durch mehrere Objekte erfüllt werden, block b1 und block b2. Man würde sich jetzt fragen, auf welchen der beiden Blöcke jetzt der Operator pick angewendet würde. Soar regelt diese Frage, indem sie den Vorschlag für eine Handlung und die Konsequenzen einer Handlung trennt.

```
sp {apply*pick
(state <s> ^operator <o>)
(<o> ^name pick
       ^obj <b>)
-->
(<b> ^ontop <x> -)
(<s> ^holding <b>)}
```

Abbildung 4: Regel um einen ausgewählten Operator anzuwenden

Um diese Trennung zu realisieren, existieren zwei Arten von Regeln, die Operatoranwendungsregeln und die Nicht-Operatoranwendungsregeln.

Operatoranwendungsregeln, „operator application rules“, enthalten in der Bedingung mindestens einen Operator. Als Beispiel sehen wir in Abbildung 4 die Ausführung des pick-Operators. Diese Regel wird erst angewendet, wenn der Operator pick ausgewählt wurde.

Nicht-Operatoranwendungsregeln, „non-operator application rules“, werden alle restlichen Regeln bezeichnet.

8 Arbeitsspeicher

Der Arbeitsspeicher, oder auch Working Memory (WM) genannt, wird von Soar benutzt um eine interne Repräsentation der Welt zu erstellen. Eine wichtige Eigenschaft von Soar ist es, neben der Trennung von Auswahl und Ausführung einer Handlung, die Möglichkeit sein WM immer aktuell zu halten. Die Unterscheidung der Produktionsregeln gibt Soar die Möglichkeit das Wissen über die Objekte zu trennen. So sind nur Aktionen von operator application rules dauerhaft. Speicherobjekte, die durch solche Aktionen erzeugt wurden, bleiben auch dann bestehen, wenn die Bedingungen der Regel nicht mehr erfüllt sind. Die Aktionen von operator application rules haben einen so genannten operator-support. Alle anderen Speicherobjekte, die durch Aktionen von non-operator application rules erzeugt wurden, haben einen instatiation-support. Diese Objekte bleiben nur solange im WM bestehen, wie auch die Bedingungen der Produktionsregel erfüllt sind. Dieser Mechanismus verhilft SOAR zu ihrer eigenem Truth-Maintenance System (TMS). So sind nur die Speicherobjekte im WM, die durch einen Operator erzeugt wurden, oder diejenigen, die sich aus dem jetzigen Zustand der Welt schließen lassen.

Das WM enthält das gesamte Wissen, das Soar von der Welt besitzt. Wie schon erwähnt wurde, ist dieses Wissen in einer Graphstruktur aufgebaut, in der alle Objekte Attribute besitzen, die als Wert wiederum andere Objekte haben können. Dabei gibt es bei der Verknüpfung zwischen den einzelnen Speicherobjekten keine Regeln, bis auf den Zustand. Der Zustand repräsentiert die Wurzel, ist somit direkt oder über andere Objekte mit allen Speicherobjekten verbunden.

Es gibt also zwei Sorten von Speicherobjekten. Die erste Sorte sind die o-supported WM-Objekte. Diese sind persistent und werden durch operator application rules erzeugt. Die zweite Sorte sind i-supported WM-Objekte. Diese sind abhängig von anderen Objekten im WM und werden nach jeder Veränderung einer dieser Objekte neu berechnet. Falls die Bedingung für ein i-supported WM-Objekt nicht mehr erfüllt ist, wird dieses aus dem WM gelöscht.

Wir wollen uns zum besseren Verständnis zwei weitere Beispiele anschauen.

```
sp {elaborate*free
(state <s> ^block <a>
           ^block <b>)
(<a> ^ontop <b>)
-->
(<b> ^occupied)
(<b> ^free -)}
```

Abbildung 6 : i-support Operator

```
sp {select*operator*pick
(state <s> ^operator <o> +)
(<o> ^obj <a>)
(<a> ^free)
-->
(<s> ^operator <o>  >  )}
```

Abbildung 5 : Operatorauswahlregel

Abbildung 6 stellt ein Beispiel aus der Klasse der Auswertungsregeln dar. Wenn wir weitere Speicherobjekte erzeugen wollen und sicherstellen möchten, dass es immer aktuell ist, gibt es zwei Möglichkeiten. Die erste wäre spezielle Regeln zu schreiben, die alle Speicherobjekte löschen würden, wenn die Bedingungen für sie nicht mehr erfüllt sind. Die andere Möglichkeit, die Soar bietet besteht darin dieses Speicherobjekt mit instatiation-support zu erzeugen. „Instantiation-supported Knowledge" werden diese Objekte in Soar genannt. Durch unser Beispiel und die Regel aus Abbildung 6 wissen wir, welche Blöcke frei sind und welche nicht, ohne andere Regeln anpassen zu müssen. Auch Aktionen, die die Blöcke bewegen, müssen sich nicht um das Attribut free kümmern.

Die zweite Regel in Abbildung 5 zeigt uns ein Beispiel aus der Klasse der „operator application rules". In unserem Beispiel mit den Klötzchen hatte Soar die Möglichkeit zwischen beiden Klötzchen zu wählen, da der Operator pick für beide Objekte vorgeschlagen wurde. Wenn wir jetzt aber die Klötzchen bevorzugen wollen, die nicht durch andere belegt sind, können wir dies durch unser aus Abbildung 6 erzeugtes i-supported Speicherobject, „free", sicherstellen. Die Regel in Abbildung 5 sagt, dass der Operator pick, der auf ein freies Objekt vorgeschlagen wurde, die höchste Priorität bekommt. Er wird somit gegenüber einem anderen Operator mit niedrigerer Priorität bevorzugt und somit ausgewählt.

9 Langzeitspeicher

Aufbauend auf dem Abeitsspeicher (WM) existiert in Soar der Langzeitspeicher oder auch Long-term Memory (LTM) genannt. Das LTM baut auf dem WM auf. Es liefert weitere Funktionen Wissen abzuspeichern. Das LTM besteht aus drei Bereichen wie sie in der folgenden Abbildung zu sehen sind.

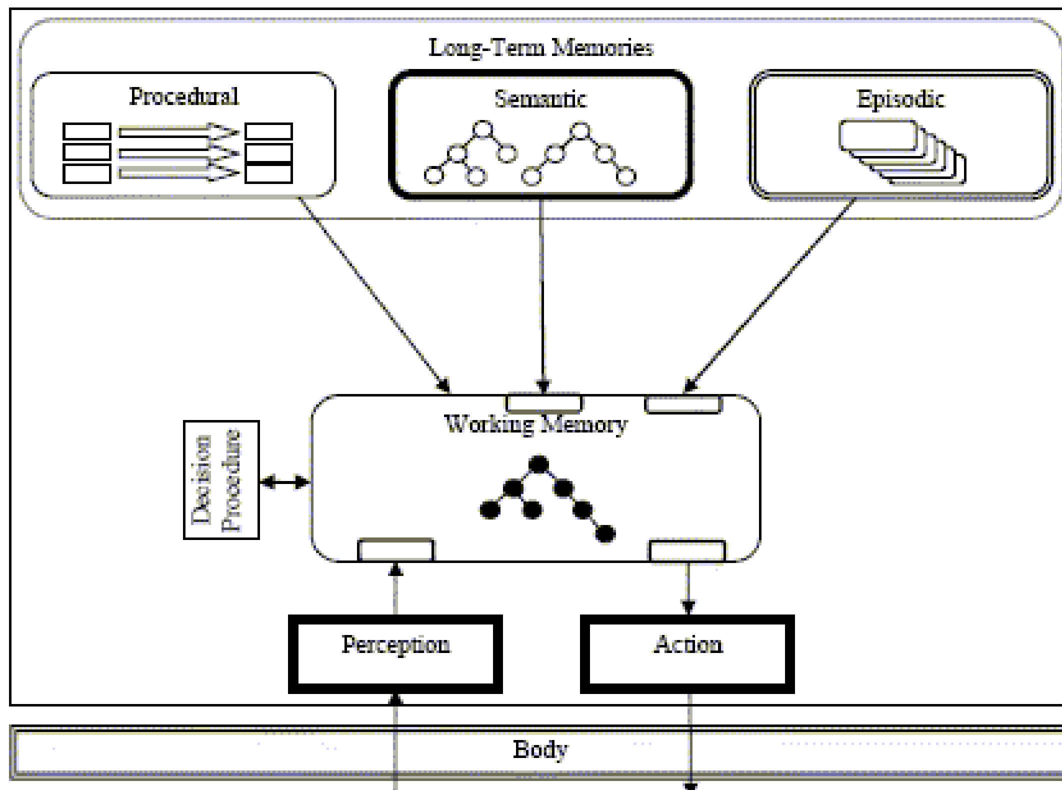

Abbildung 7: Soar Strukturdarstellung

9.1 Prozeduraler Langzeitspeicher

Im procedural LTM wird das ganze Wissen über alle definierten Regeln abgelegt. Das procedural LTM enthält eine Liste aller definierten Regeln. So würden alle Produktionsregeln, die wir bisher als Beispiele gesehen haben in Soar im procedural LTM gespeichert werden.

9.2 Semantischer Langzeitspeicher

Das semantic LTM repräsentiert die Zusammengehörigkeit von Strukturen im WM. Dieses Wissen beschreibt deklarative Strukturen, die ausdrücken was man „weiß" im Unterschied zu dem, an das man sich erinnern kann. Es ist ein Gebiet der aktuellen Forschung, wann solche Strukturen im semantischen Speicher abgespeichert werden sollen. Soll jede Struktur, die im WM auftaucht gespeichert werden? Als Beispiel kann man hier die Eigenschaften der Blöcke erwähnen. So könnte man im semantic LTM speichern, dass alle Blöcke eine Farbe

und einen Namen haben. Diese Attribute könnte man dann im WM in bestimmten Bedingungen benutzen.

9.3 Episodischer Langzeitspeicher

Der letzte Bereich ist das episodic LTM. In Soar werden Episoden automatisch aufgezeichnet, wenn ein Problem gelöst wird. Solche Probleme können die Auswahl der richtigen Operatoren sein, wenn mehrere zur Wahl standen und keine Wahl zwischen den Operatoren getroffen werden konnte. Eine Episode beinhaltet eine Teilmenge der WM-Objekte, die zum Zeitpunkt der Aufnahmen existierten. So kann Soar beim nächsten Mal, wenn dieselben Operatoren vorgeschlagen und dieselben Bedingungen im WM erfüllt werden, wie sie in der Episode aufgezeichnet wurde, eine Regel definieren, die den in der Episode ausgewählten Operator allen anderen vorzieht.

10 Sackgassen

In unserem einfachen Beispiel mit den Klötzchen haben wir gesehen, dass es zu Problemen während der Auswahl von Regeln kommen kann. So gab es die Regel, die einen Operator vorgeschlagen hat, um eines der Klötzchen aufzuheben. Da es jedoch zwei Klötzchen gab, wurden zwei Operatoren für jeweils eines der Klötzchen vorgeschlagen. Zwischen diesen Operatoren gab es jedoch keine Entscheidung. Sackgassen („impasses“) entstehen in Soar automatisch, wenn die Entscheidung zwischen Operatoren nicht getroffen werden kann.

Sobald eine Sackgasse durch Soar gefunden wird, werden spezielle Regeln getestet, die sich nicht mehr mit konkreten Weltdarstellungen beschäftigen, sondern mehr über die Entscheidungen und Darstellungen schließen. So kann man Regeln definieren, die alle WM-Objekte in eine Episode abspeichern. Man kann auch Regeln definieren, die solche Sackgassen, oder auch andere Probleme lösen, indem sie eine Lösung zufällig auswählen und diese statistisch im LTM speichern. Die Möglichkeiten sind an dieser Stelle speziell sehr offen gestaltet, da Soar die Möglichkeit bieten will, den Denkprozess und das Schließen eines Menschen so gut und komplex wie es nur geht zu beschreiben und zu simulieren.

Impasses stellen eine Wissenslücke dar. Diese Wissenslücken bieten eine Möglichkeit um zu lernen.

11 Lernmechanismen

Wir haben gesehen, dass Soar nicht nur Operatoren auf Zustande in der Welt anwenden kann, um bestimmte Aktionen zu erzeugen, sondern dass Soar auch spezielles Wissen über Fehler oder Probleme in der Welt anlegen kann. Aufbauend auf diesem Wissen wollen wir uns nun einige Möglichkeiten ansehen, die es Soar ermöglichen das menschliche Lernen zu simulieren.

11.1 Chunking

Das Chunking erzeugt neue Regeln und speichert diese im LTM, wenn eine Sackgasse gelöst wurde. Das Chunking erzeugt eine Regel, die als Bedingung die Operatoren beinhaltet, zwischen denen Soar keine Wahl treffen konnte, und alle Bedingungen, die während der

Auflösung der Sackgase benutzt wurden. Die Aktion der Regel, die das Chunking erzeugt, ist eine erhöhte Priorität für den ausgewählten Operator gegenüber den restlichen Operatoren, so dass die anderen nicht ausgewählt werden.

11.2 Reinforcement learning

Das Reinforcement learning geht nun näher auf die Prioritäten zwischen Operatoren ein. Nachdem Aktionen ausgewählt wurden und ein Feedback in Form von verändertem Wissen vorliegt, wird dieses Wissen mit den letzten Aktionen verglichen. Wenn Ziele erreicht wurden, oder nicht mehr erreicht werden können, kann man dies als positives oder negatives Feedback auffassen. Solch ein Feedback kann durch ein WM-Objekt dargestellt werden. Bei negativem Feedback würde man die Prioritäten des letzten Operators verringern und somit bei der nächsten Entscheidung anderen Aktionen den Vorrang erteilen. Hier muss man beachten, dass das Chunking diese veränderten Prioritäten nicht wieder verändern würde. Sackgassen treten nur auf, wenn Operatoren dieselben Prioritäten besitzen.

11.3 Episodic learning

Im Episodic learning werden die so genannten Episodes aufgezeichnet, um sie dann im LTM abzulegen. Es dient mehr einer statistischen Erfassung aller früheren Aktionen. Das Episodic learning kann bei längeren und komplexeren Problemlösungen helfen sie beim nächsten Mal durch das vergleichen mit einer Episode in einem Schritt zu lösen.

Das Semantic Learning versucht Zusammenhänge zwischen Wissenselementen aufzufinden. Das Semantic Learning ist ein noch offenes Forschungsgebiet. Man kann alle Wissenselemente, die zu einer Aktion führten zusammenfassen, doch ob es einem ein „mehr an Wissen" erzeugt, ist man sich noch nicht im Klaren. Semantische Strukturen können am leichtesten von Hand geschrieben werden, um Zusammenhänge zwischen Objekten zu beschreiben. Meistens will man gewisse Sachverhalte dem Programm vorgeben, ohne dass es diese erst lernen muss.

12 I/O-Interface

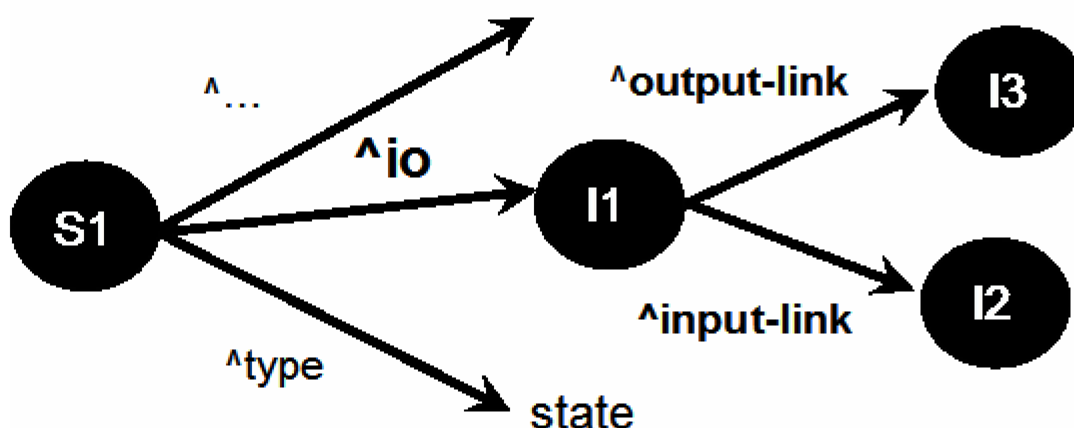

Abbildung 8: I/O-Interface

Jeder Zustand bekommt durch Soar ein weiteres ^io Attribut. Dieses ^io Attribut hat einen Ein- und Ausgang. Andere Programme können durch diese festgelegten Schnittstellen mit Soar kommunizieren. Alle Objekte im Input-link werden von einer Klasse in Soar oder durch ein externes Programm gelöscht und neu Objekte erzeugt. Auf diese Weise können in Soar Regeln auf neue Eindrücke aus der Welt matchen und das Programm auf den neuen Input reagieren.

An den Output-link werden all die Aktionen gelegt, die ausgegeben werden sollen. So müsste man in unserem Klötzchenbeispiel noch Regeln schreiben, die nicht nur in der Soar-Repräsentation Klötzchen aufnehmen und diese bewegen, sondern auch die nötigen Bewegungen und Befehle an die Außenwelt abgeben. Dies gehört jedoch weniger zu Soar als mehr zur Schnittstelle der jeweiligen Anwendung, z.B. einer Robotersteuerung. Regeln in Soar würden Anweisungen in Objekten auf den ^output-link legen. Diese Anweisungen müssten nach vorgegebenen Schnittstellenbeschreibungen spezielle Attribute tragen, die dann der Maschine sagen könnten, wie weit sie den Arm bewegen soll, oder an welcher Position sie das Klötzchen fallen soll.

13 Entscheidungszyklus

Nachdem wir die einzelnen Teile von Soar gesehen haben, wollen wir uns anschauen, wie diese Einzelteile in Soar zu einem Ganzen werden. Soar ist in fünf Einzelschritte aufgeteilt, die jeweils eigene Aufgaben erfüllen. In Abbildung 9 sehen wir alle Einzelschritte im Entscheidungszyklus von Soar.

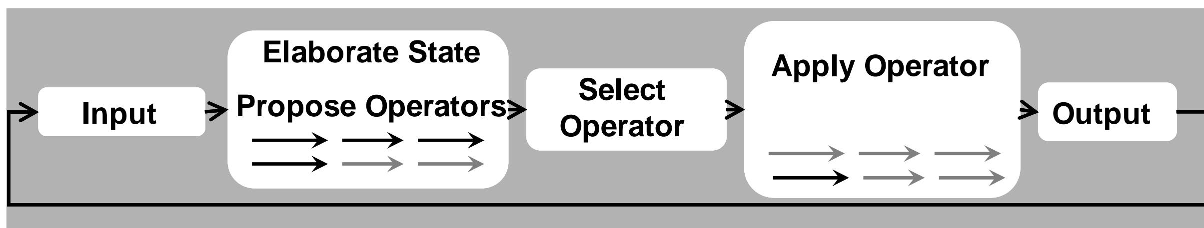

Abbildung 9: Soar Entscheidungszyklus

Am Anfang des Zyklus wird der Input durch andere Programme verändert. Hier werden die Objekte, die von außen in das Soarprogramm kommen sollen, gespeichert.

Danach geht Soar in den Elaborate State. Hier werden alle „non-operator" Regeln gefeuert, deren Bedingungen zutreffen. Dabei können Regeln auch andere Regeln auslösen. Dies war in unserem Klötzchen-Fall die „select"-Regel. So können einige Regeln, die sich auf temporäre WM-Objekte beziehen erst feuern, wenn diese ihre Aktionen bereits ausgeführt haben. Im Elaborate State werden auch alle zurzeit möglichen Operatoren vorgeschlagen.

Als nächstes wird in Soar genau ein Operator ausgewählt. In diesem Select Operator State werden alle Operatoren nach Priorität sortiert betrachtet und der Operator mit der höchsten Priorität ausgewählt.

Im Apply Operator State werden alle Operator-Regeln gefeuert. Es werden also Regeln ausgelöst, die als Bedingung den ausgewählten Operator haben. Hier werden auch die langwierigen Veränderungen erzeugt.

Als Letztes wird geht der Soarzyklus in den Output-State über. Im Output State stellt Soar die WM-Objekte aus dem ^output-link anderen Programmen zur Verfügung. Der erzeugte Output kann durch andere Programme gelesen werden und kann Aktionen in der wirklichen Welt vornehmen.

14 Unified Theories of Cognition

Der Mensch verfügt über zahlreiche kognitive Fähigkeiten: Gedächtnis, Sprache, Wahrnehmung, Problemlösen, geistiger Wille, Aufmerksamkeit, … Das Ziel der Kognitionspsychologie ist es, die Eigenarten dieser Fähigkeiten zu erforschen und soweit wie möglich, in formalen Modellen zu beschreiben. Diese Modelle können dann als kognitive Architektur auf einem Computer realisiert werden.

Unified Theories of Cognition ist ein Modell um die Details aller Mechanismen des menschlichen Denkens zu erklären. SOAR steht noch nicht als Architektur, um alle kognitiven Prozesse zu simulieren. Es wurden jedoch Projekte realisiert, um Systeme miteinander zu vernetzen und auszubauen.

15 Beispielanwendungen von SOAR

	Wahrnehmung	Lernen	Problemlösen	Sprache	geistiger Wille
NTD-SOAR			X	X	
Instructo-SOAR		X	X	X	
IMPROV	X	X	X		
TacAir-SOAR, RWA-SOAR MOUTBOT	X		X	X	

Abbildung 10 : kognitive Eigenschaften von Beispielprogrammen

Im Folgenden wollen wir uns Programme anschauen, die einige dieser kognitiven Fähigkeiten besitzen. Eine komplette Zusammenfassung der Programme und ihrer Eigenschaften ist in Abbildung 10 zu sehen.

NTD-Soar ist eine Berechnungstheorie des NASA Test Directors. Er verwendete Materialien in seiner Umgebung und kommunizierte mit Angestellten im Space Shuttle Rakettenstartteam. NTD-Soar ist eine Kombination aus NL-SOAR, einem Soarprogramm, das menschliche Sprache analysiert, und NOVA, einem Model für die Bildverarbeitung. Zusätzlich wurde Wissen zur Entscheidungsfindung und Problemlösung in Soar kodiert, dass notwendig ist, um ein Space Shuttel zu testen und auf den Start vorzubereiten. [Rubinoff, Lehman 1994]

Instrukto-Soar ist eine Berechnungstheorie, wie Menschen durch interaktive Instruktionen lernen können. Instrukto-Soar ist eine Kombination aus NL-SOAR und kodiertem Wissen, wie man aus Instruktionen während eines Problemlösungsprozesses lernen kann. Instrukto-

SOAR ist in der Lage neue Verfahren durch natürliche Sprache zu lernen. [Pearson, Huffman 1995]

IMPROV ist eine Berechnungstheorie, um Wissen über die Konsequenzen von Aktionen zu korrigieren. IMPROV ist eine Kombination aus SCA, einem Model für kategorisches Lernen, und kodiertem Wissen, wie man Konflikte zwischen dem internem WM und der Observation der Welt entdeckt. IMPROV kann automatisch sein Wissen über Interaktionen mit der Umgebung verbessern. So vergleicht IMPROV nach jeder Aktion, ob der Input im nächsten Zyklus dem erwarteten Zustand der Welt entspricht. Falls Unterschiede bestehen passt IMPROV seine Regeln so an, dass beim nächsten Mal die interne Weltrepräsentation sich nicht von der Außenwelt unterscheidet. So werden zum Beispiel die Aktionen von Operatoren korrigiert. IMPROV wäre so in der Lage festzustellen, dass nach dem Loslassen eines Klötzchens andere Klötzchen in der Nähe ihre Position, durch Schupsen oder Ähnliches, verändern. [Pearson, Huffman 1995]

TacAir-Soar und RWA-Soar sind Modelle von menschlichen Piloten. Mit über 8000 Regeln bei TacAir sind es die größten SOAR Modelle. Zu ihren Fähigkeiten zählen das Absolvieren von autonomen Flugmissionen, Bearbeitung von dynamischem Missionswechsel, die Einhaltung der Kommandostruktur bei Flugmanövern mit vielen Flugzeugen und die Einhaltung einer Kommunikationsverbindung in „militär Englisch". RWA-Soar wurde für Hubschrauberpiloten und TacAir-Soar für große Flugzeuggeschwader geschrieben. Diese Programme wurden durch das Militär entwickelt, weshalb sie wohl so groß und komplex wurden. [Tac-Air-Soar], [Hill et. al. 1998]

Der Soar Moutbot steht für Militär-Operationen im urbanen Terrain (MOUT). SOAR kontrolliert in diesem Programm mehrere individuelle Gegner, die Häuser verteidigen. Dabei werden diese Häuser vom Militär angegriffen und gestürmt. Soar MOUTBOT ist eine kombiniert gleich eine Vielzahl von Fähigkeiten, wie die Integration von reaktiven Aktionen, Planung, die Kommunikation mit anderen Individuen, die Koordination zwischen echten und virtuellen Personen und das räumliche Schließen. [Wray et. al. 2004]

16 Zusammenfassung

Das menschliche Denken wird schon lange von Wissenschaftlern untersucht. Soar ist als eine kognitive Architektur in der Lage menschliches Denken und Schließen zu simulieren. Bei diesem simuliertem Denkprozess treten dieselben Probleme auf, wie sie sich im Gedankengang eines Menschen stellen. Soar stellt Möglichkeiten und Lösungskonzepte vor, wie man mit solchen Problemen umgehen kann. Wissenschaftler aber auch große Militärprojekte profitieren von der kognitiven Architektur, die Soar bereitstellt, um das menschliche Denken so detailliert darstellen und simulieren zu können.

Soar ist eine kognitive Architektur, die es ermöglicht viele menschliche Denkprozesse zu simulieren. Die einzelnen Bestandteile der Soar Architektur wurden untersucht. Soar führt seine Handlungen durch einzelne Produktionsregel aus. Dabei werden die Bedingungen der Regel geprüft und ihre Aktionen ausgeführt. Das Wissen über die Welt wurde anhand von Zuständen in Soar dargestellt. Alle Wissensobjekte sind mit einander in einer großen Graphstruktur verbunden. Soar differenziert Wissensobjekte weiter. Es gibt in Soar

Wissensobjekte, die als Auswertung und Interpretation der Welt angesehen werden können, und Wissensobjekte, die durch Handlungen erzeugt wurden. Lernmechanismen bieten Programmen in Soar die Möglichkeit ihre Handlungen während ihrer Ausführung anzupassen. Soar kann nicht als eine Gesamttheorie der Kognition gesehen werden, da noch einige Aspekte des menschlichen Denkens nicht beschrieben werden. Es existieren jedoch schon viele Ansätze, die einzelne Teilaspekte des menschlichen Denkens kombinieren.

17 Referenzen

[SOAR] http://sitemaker.umich.edu/soar/home

[Lehman et. al. 2006] A gentle introduction to Soar, an architecture for human cognition: 2006 update.

http://ai.eecs.umich.edu/soar/sitemaker/docs/misc/GentleIntroduction-2006.pdf

[Pearson, Huffman 1995] Douglas J. Pearson, Scott B. Huffman, *"Combining learning from instruction with recovery from incorrect knowledge"*, 1995

http://www.sunnyhome.org/pubs/mlw95.pdf

[Tac-Air-Soar] http://www.soartech.com/projects/TacAir-Soar.pdf

[Hill et. al. 1998] R. Hill, J. Chen, J. Gratch, P. Rosenbloom, M. Tambe, "*Soar-RWA: Planning, Teamwork, and Intelligent Behavior for Synthetic Rotary Wing Aircraft*," Proceedings of the 7th Conference on Computer Generated Forces & Behavioral Representation, Orlando, FL., May 12-14, 1998.

[Wray et. al. 2004] Robert E. Wray, John E. Laird, Andrew Nuxoll, Devvan Stokes, Alex Kerfoot, *"Synthetic Adversaries for Urban Combat Training"* Presented at Innovative Applications of Artificial Intelligence Conference. San Jose, July, 2004.

[Pearson, Huffman 1995] Pearson, D.J, Huffman, S.B., *"Combining Learning from Instruction with Recovery from Incorrect Knowledge"*. *Machine Learning 95 Workshop on "Agents That Learn From Other Agents"*, 1995.

[Rubinoff, Lehman 1994] Rubinoff, R., and Lehman, J. F., "Real-time natural language generation in NL-Soar". In Proceedings of 7th Internat.GenerationWorkshop, 1994.