Bachelorarbeit

The Good, the Bad, and the Regular

Adopting the Analytical Inductive Programming System IGOR for an Autonomous Agent's Intentional Cognitive System and his Categorized Stream of Experience

Anwendung von Igor – eines Systems zur analytischen induktiven Programmierung – für das intentionale kognitive System eines autonomen Agenten und dessen kategorisierten Erfahrungsstrom

Marius Raab

Otto-Friedrich-Universität Bamberg
Fakultät Witschaftsinformatik und Angewandte Informatik
Studiengang B.Sc. Angewandte Informatik

Wintersemester 2011/ 2011

Gutachterin:
Prof. Dr. Ute Schmid

Contents

Summary

In computer science as well as in cognitive science, there's an ongoing debate how abstract, high-level (symbolic) knowledge emerges from the most basic sensory inputs. So-called hybrid systems try to bring together artificial neural networks (ANNs) with (for example) logic-based inference machines. Although many approaches exist, so far there's no satisfying answer to the famous question phrased by Stevan Harnad: "How can the semantic interpretation of a formal symbol system be made *intrinsic* to the system, rather than just parasitic on the meanings in our heads?" In particular, this applies to these system's higher learning mechanisms—which are not even able to cope with regularities more complex than simple sequences.

This thesis integrates two aspects that are often neglected when autonomous agents (AAs) are designed to tackle the *symbol grounding problem*: (a) a strictly one-store memory structure that allows for different modes of retrieval, and that is consistent with empirical findings from human memory research; and (b), a framework that regards cognition as well as emotion and motivation.

Together with a companion thesis that focuses on the early processes of symbol grounding, this is the necessary foundation for utilizing the state-of-the-art indictive rule acquisition device IGOR2. A seamless bottom-up processing of information finally feeds into this device that—under certain constraints, and provided such regularities exist in the environment—is able to detect recursive schemata.

Exemplified with the concept of *odd* and *even*, the AA proposed in this paper is designed theoretically. By using *syntactical* mapping of memory traces to ASCII characters, and *syntactical* transformation of rule output to a grammar and then to a memory fragment, IGOR2 is incorporated as a cognitive component. Additionally, a proof-of-concept implementation in JAVA demonstrates feasability.

Special regard is given to extensibility. Although motivation and emotion is implemented only rudimentary and input is kept simple, the system is designed to scale to a full-fledged cognitive-emotional-motivational architecture without structural change.

It is shown that, with some modifications, IGOR2 can be used as an even more powerful cognitive device in the future; powerful in a sense that it accounts for more strengths—and weaknesses—of human learning. Making learned rules accessible to the agent is outlined, and a possible means for analogical reasoning is shown.

This unique combination of a sound memory concept, a cognitive-emotional-motivational theory and a inductive rule acquisition device is discussed with regard to cognitive plausibility. Although some necessary components—as a classical planner—are yet to be implemented, the approach is a promising foundation for future research.

Zusammenfassung

Wie entsteht abstraktes (symbolisches) Wissen aus grundlegender sensorischer Information? Diese Frage diskutieren Informatiker und Kognitionspsychologen seit Jahrzehnten. Hybride Systeme verbinden künstliche neuronale Netzwerke mit (beispielsweise) logikbasierten Inferenzmaschinen. Trotz vieler Ansätze ist die berühmte, von Stevan Harnad gestellte Frage immer noch aktuell: „How can the semantic interpretation of a formal symbol system be made *intrinsic* to the system, rather than just parasitic on the meanings in our heads?" Dies trifft insbesondere auf höhere Lernmechanismen zu, die gerade einmal Regularitäten in Form einfacher Sequenzen verarbeiten können.

Die vorliegende Arbeit integriert zwei Aspekte, die beim Entwurf autonomer Agenten oft vernachlässigt werden: (a) ein striktes Einspeicher-Gedächtnismodell, mit verschiedenen Abrufmechanismen, und im Einklang mit empirischen Befunden; und (b) die Berücksichtigung von Kognition ebenso wie von Emotion und Motivation.

Zusammen mit einer parallel entstehenden Arbeit, die die frühen Prozesse des *symbol grounding* beleuchtet, ist dies die notwendige Voraussetzung, das induktiv arbeitende Regel-Erwerbs-System IGOR2 zu nutzen. Eine von Grund auf nahtlose Informationsverarbeitung mündet in dieses System, das – unter bestimmten Einschränkungen, und wenn solche Regularitäten in der Umwelt existieren – rekursive Schemata lernen kann.

Am Beispiel der Unterscheidung von *gerade* und *ungerade* wird der Agent theoretisch skizziert. Über die *syntaktische* Abbildung von Gedächtnisfragmenten auf ASCII-Zeichen und die *syntaktische* Übersetzung der gelernten Regel in eine Grammatik und dann in ein Gedächtnisfragment wird IGOR2 als kognitive Komponente genutzt. Zusätzlich demonstriert eine beispielhafte JAVA-Implementierung die Machbarkeit.

Besonderes Augenmerk liegt auf der Erweiterbarkeit der Architektur. Zwar sind Motivation und Emotion nur rudimentär vorhanden, und der Input einfach; aber das System kann ohne strukturelle Änderungen zu einer vollständigen kognitiv-emotional-motivationalen Architektur erweitert werden.

Mit einigen Änderungen kann IGOR2 in der Zukunft noch mächtiger werden – in dem Sinne, dass weitere Stärken und Schwächen menschlicher Informationsverarbeitung abgebildet werden. Die Nutzung gelernter Regeln durch den Agenten wird skizziert, und die Umsetzung analogen Schließens wird angesprochen.

Die einzigartige Kombination aus einem fundierten Gedächtnismodell, einer kognitiv-emotional-motivationalen Theorie und einem mächtigen induktiv arbeitendem Regel-Erwerbs-System wird im Hinblick auf kognitive Plausibilität diskutiert. Auch wenn einige Komponenten – wie ein klassischer Planer – noch integriert werden müssen, so ist der Ansatz doch vielversprechend für die weitere Forschung.

odd

adj

Definition of ODD

1 **a** : being without a corresponding mate <an *odd* shoe>

b (1) : left over after others are paired or grouped (2) : separated from a set or series

2 **a** : somewhat more than the indicated approximate quantity, extent, or degree —usually used in combination <300-*odd* pages>

b (1) : left over as a remainder <had a few *odd* dollars left after paying his bills> (2) : constituting a small amount <had some *odd* change in her pocket>

3 **a** : being any of the integers (as −3, −1, +1, and +3) that are not divisible by two without leaving a remainder

b : marked by an odd number of units

c : being a function such that $f(-x) = -f(x)$ where the sign is reversed but the absolute value remains the same if the sign of the independent variable is reversed

4 **a** : not regular, expected, or planned <worked at *odd* jobs>

b : encountered or experienced from time to time : OCCASIONAL

5 : having an out-of-the-way location : REMOTE

6 : differing markedly from the usual or ordinary or accepted : PECULIAR

1 Isn't it *Odd*?

This thesis is *odd*—in the sense mentioned as first definition on the page before[1]: "being without a corresponding mate". Here, *mate* is another thesis written in parallel, laying the foundation—the *grounding*—for the autonomous agent (AA) proposed here. Although self-contained, this work here is best understood as a second chapter of a bigger picture.

In another sense, this thesis deals with *odd* and *even*. The agent in question will learn to distinguish between these concepts. It will do so by using an inductive rule acquisition device, IGOR, and consequently, it will derive a rule that is able to judge oddness/ evenness *ad infinitum*. Such an ability is no implicitness in AI research.

Inductive learning itself is not a trivial matter. Integrating it into an AA raises many additional questions:

- Learning needs some criterion to make distinctions; in other word, the agent needs to know what is right and wrong, *good* or *bad*. Otherwise, regularity could not emerge.
- This, in turn, asks for a framework that allows for such decisions. In the best case, this is done on a psychologically valid basis.
- Additionally, learning needs some sort of memory; so how to model it? And how to connect it to the decisional framework?
- And, last but not least: What are the basic units in memory? How are they *grounded*?

An AA's model should be more than an accumulation of parts or modules. It should comprise functional realms, and integrate them seamlessly. Even when a high cognitive ability like learning of inductive rules is in focus, issues like memory and motivation should not be ignored. Ignoring the surroundings would raise the question if the system is *cognitive*; or if it would just be a powerful means for rule learning with a new interface for input/ output (I/O).

Langley et al. (2009b, p.155f) state a set of requirements that—in their opinion—should by addressed by research on cognitive systems. Two of these are of special importance for this thesis:

- "We need more research on architectures that directly support both episodic memory and reflective processes that operate on these structures it contains."

[1] Screenshot from http://www.merriam-webster.com/dictionary/odd

- "Emotions play a central role in human behavior, yet few systems offer an account of their purposes or mechanisms. We need new architectures that exhibit emotion in ways that link directly to other cognitive processes and that modulate intelligent behavior."

This paper aims at sketching the 'upper half' of such an architecture, building upon the parallel work's grounding system. While doing so, it will give attention to rule learning. This is a promising research area. The framework ICARUS by Langley et al. (2009a, p.330), for example, solely relies on deductive reasoning and lacks "abductive methods that make plausible default inferences".

While *symbol grounding* is not the main focus here, it will be kept in mind. The symbols in the categorized stream of experience entering the systems will be handled carefully, to avoid they lose ground. Hopefully, the framework developed here will contribute to this debate, too. The challenge is auspicious:

> I believe there has been clear progress on the issue of symbol grounding and that there will be much more progress in the coming decade, provided we keep an open mind, engage in interdisciplinary curiosity, and avoid false debates. (Steels, 2008, p.241)

2 Symbol Grounding

"How can the semantic interpretation of a formal symbol system be made *intrinsic* to the system, rather than just parasitic on the meanings in our heads? How can the meanings of the meaningless symbol tokens, manipulated solely on the basis of their (arbitrary) shapes, be grounded in anything but other meaningless symbols?" Since Stevan Harnad (1990, p.335) has posed this question, the *symbol grounding problem* is an ongoing issue in AI debate.

This thesis is not about symbol grounding; a thesis written in parallel will focus on the grounding of symbols, in theory and in practice. However, the issue can not be completely ignored here: Utilizing IGOR as a learning device bears the risk of using a *cognitive wheel*—a term coined by D.C. Dennett for powerful computational systems that exhibit intelligent (in a human sense) behavior while at the same time neglecting psychological and biolocigal foundations.

Such a *cognitive wheel* would not be useless. Technically speaking, it could exhibit *functional equivalence*, i.e. be comparable to a human (problem solver) in its output. Yet, *structural equivalence* is much more desireable from a psychological point of view. In other words, a system that is comparable (not only in its output but also) in its inner workings to a problem-solving mind can help to gain deeper insights about cognition in general.

This, in turn, raises philosophical questions: What *is* cognition, after all? Is it specific to biological neurons; is it inextricably linked to having a physical body? At least, the very influential work by Winograd and Flores (1989) argues that our *being-in-the-world* (a term introduced by Heidegger (1986)) is an exclusive human property.

This debate, too, is not in focus here. We will take a rather pragmatic view, building upon the other thesis mentioned above. There, an autonomous agent (AA) is designed that is equipped with a light sensor and two motor-driven wheels. The agent is able to detect changes in brightness on the floor's surface, and it is able to experience reward and punishment.

Using its light sensor input and the reward/ punishment, it shall be able to detect structure in its environment—solely by relating different possibilities of action (and experience) to changes in light reflected from the floor beneath. Equipped (only) with these modes of sensing and acting, it will build a graph as a knowledge structure, reflecting relevance and structure in the environment.

(1): The *Kreiswelt* the proposed AA is *thrown* into. Courtesy of Mark Wernsdorfer; his thesis will be the symbol grounding foundation for this thesis.

To make this a bit more conceivable: The agent will move[2] in a *Kreiswelt* (see figure 1); that is, he is *thrown* in a world of concentric circles of different number. An even (or odd, depending on the scenario) number of circles means that in the innermost spot the agent will experience reward (or punishment).

So the concept of odd and even is present in the environment; at least, a human spectator would naturally use these labels. The agent, in contrast, will only experience changes in the intensity of light reaching its sensor, and it will experience reward and punishment[3] at some points. Will this input, together with a well-grounded and -founded system, be enough to learn a high-order distinction of odd and even—at least the principle, as our agent will have no language to label this distinction?

[2]Presumably, as the proposed system will take quite a time to build meaningful representations, this will take place in a computer-simulated environment. However, it will be able to transfer the system (including learned knowledge) into a LEGO MINDSTORMS robot later on.

[3]Admittedly, reward and punishment will be handed out *deus-ex-machina*-like by a supervising, god-like instance. This, however, is only a little shortcut that will help to transfer the system to a LEGO robot later. It's not hard to imagine that such a reward/ punishment spot might be a place where the structural integrity of the robot is harmed, or where he can reload his batteries. Yet, this would be hard to realize in a real-world robot system, so we decided to just simulate these conditions.

As the other thesis is work in progress at the time of writing, no detailed account can be given. However, for our purposes, the output only is essential. We will assume that:

- We will get a *categorized* stream of experience; that is, the input we get will reflect features of the environment in a bijective relation. This means: From an input, we know what state the agent has been in.
- *Categorized* also implies that this relation reflects distinctive differences in the environment. Here, this means that the input will differentiate between the agent being over-a-line and not-over-a-line; and concerning reward, differentiate between experienced harm vs. experienced pleasure.
- The input is *segmented*; which means that it starts when the outmost circle of a number of concentric circles is crossed, and it ends when reward/ punishment is experienced.
- The single events of such a stream arrive in a *chronological order*.
- The single events can be mapped to a *8-Bit* binary representation (for this model, which will of course be extensible), and reward/ punishment is represented as *boolean* (again, for this first model). As we assume a single sensor only, that is designed to deliver a brightness value between 1 and 100, this should pose no problem.

These will be the first steps on a way that meets a central requirement of Harnad (1990, p.345): "[...T]here is really only one viable route from sense to symbols: from the ground up." With these prerequisites taken for granted, this paper will describe the second half of a journey from very basic sensory preception to a very abstract cognitive representation—not *parasitic* but *grounded*!

3 Dörner's Psi

In this section, a rough account of Dörner's Ψ theory is given. Needless to say, this can only be a rough account; yet, it will provide enough details to outline why Ψ is an apt framework for incorporating inductive rule learning.

3.1 A Basic *Blueprint*

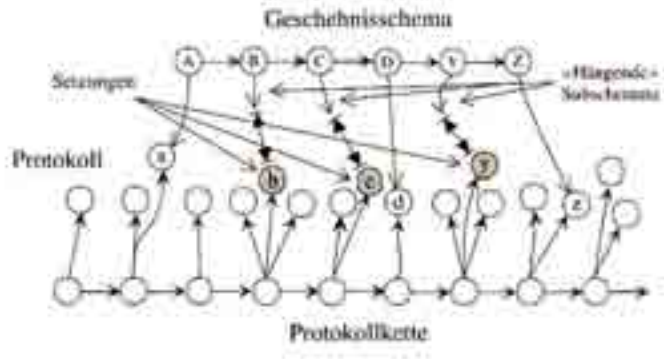
A living being is, in Dörner's view, a system that "is depending on certain aspects of its inner milieu to stay constant" (Dörner, 1996, p.329). At a basic level, such a system comprises many control variables, with desired parameters, disturbing influences and control factors—forming a regulator circuit.

A *disturbance function* exerts a causal influence on a control variable, shifting it from the desired value. A *control factor* thus is a means of regulating the system, shifting the variable towards the desired value. Obviously this factor has to be tuned with respect to actual and target values.

Temperature and energy supply are two very common parameters for living as well as for artificial systems that are usually regulated by some sort of feed back. For Dörner, *living* begins with such regulations. Living systems, however, usually rely on more complex feed-back control systems, taking the character of a disturbing influence into account. For example, overheating because of too much motor activity might require other forms of regulation (like an immediate halt) than overheating because of a fire nearby (which should result in a move-like-hell maneuver) (see Dörner, 2001).

Self-regulation is not limited to such basic parameters. In principle, every variable in the system might be subject to such feed back. It is important to note, however, that not every parameter abberation must result in countermeasures. In the first place, it is a *need* that would require some action. If any action is about to be taken—i.e. if there is a *want*—might depend on other factors. When, for instance, more than one need is present in the system, not all countermeasures can be taken at the same time.

So, with a sensoric system (for registering needs), a motor system (to perform actions), and a motivational system (for generating wants) a basic agent is almost ready. If we add some sort of ability to feel *lust* whenever a need is satisfied, the system is able to judge which actions are useful under which environmental condition. Together with a memory—storing a protocol of actions and lust- resp. aversion-experiences—the basic *blueprint* for a living system, and for an AA, is complete (see Dörner, 1996, 2001).



(2): A protocol memory chain, from Dörner (2001), p.194

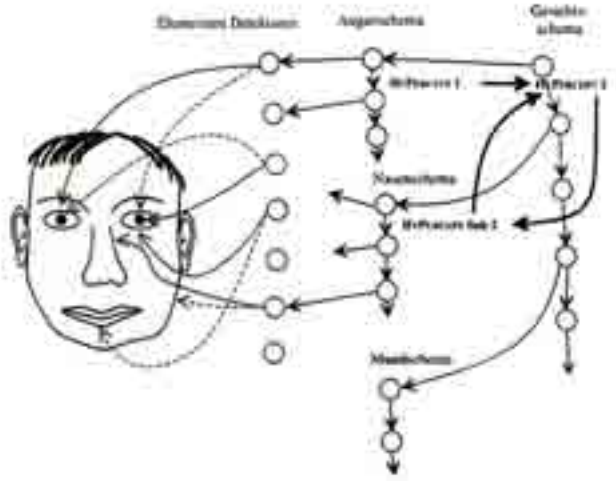
3.2 Protocol Memory

At the core, we thus are dealing with triplets: perception, motivation, reaction. For learning to be possible, a log of these triplets is needed. Dörner postulates a *protocol memory* (fig. 2). Every time something *happens*, that is, the agent's inner state changes, the event is stored. Some parts of this protocol are forgotten, *relevant* parts (which are associated with the perception of lust or aversion) are in the memory to stay.

So, what is an *event*? Dörner makes extensive use of the *schema* concept. Such a schema again is a protocol, it is a chain of motor actions (e.g. of eye movements) associated with a stimulus. Combinations of such basic schemata are schemata themselves. Fundamentally, everything is rooted in sensorimotor schemata.

Perception is a process of matching existing schemata with sensory input. In a *perceptual cycle*, these expectations are constantly updated. The process is recursive: Sub-parts of input are subject to this *hypothesis driven perception* (fig. 3), too. An event therefore consits of an unique combination of schemata. However, everything, the perception of inner states as well as the perception of external reality, is no passive reception. Active schema construction and matching always are in-between perceiving and acting.

Obviously, such a protocol memory is not free of context. It is the agent's inner state that determines relevance. It is previous experience that determines perception. When existing schemata are diffuse—for example, with missing parts—, or when the input is noisy, the agent won't see what's there, but what he knows that should be there (see Dörner, 2001).



(3): Hypothesis driven perception, from Dörner (2001), p.159

3.3 Time and Space

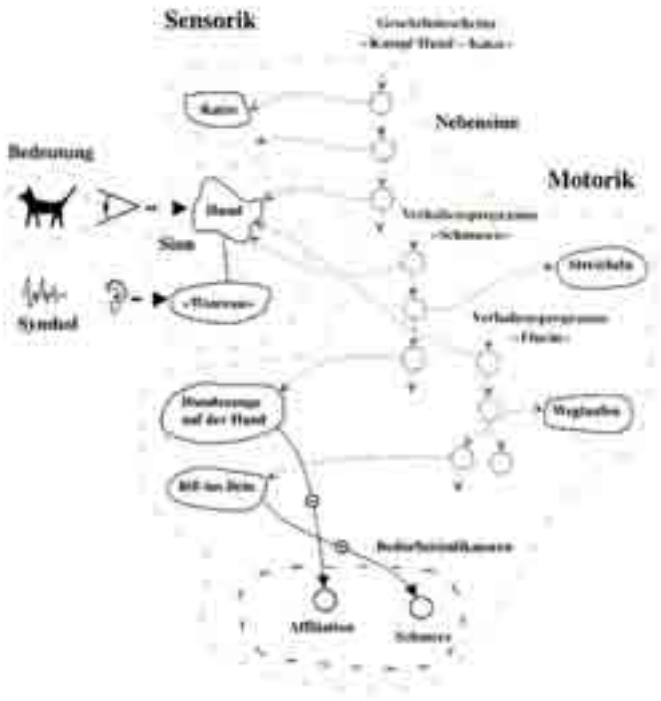
In memory, there is no absolute notion of time. Only a relative sequence of events is stored, with omissions and ideosyncratic features. Consequently, a long and dull period of time would be remembered as quite short, whereas a brief event full of new and relevant information would be stored as extensive memento.

As everything is schema-dependent, the notion of *presence* is far from being objective, too. Dörner assumes that an autonomous system constantly holds up a schema of the current state of affairs. If there's a mismatch, i.e. a notion that won't fit in this sequential schema, Dörner favours a *reflection algorithm*. In this case, memory is searched for an alternative course of events that fits better. Here again, the actual schema as well as those that might be found in memory, are a result of the agent's previous experience.

Given a schema currently active, the agent is equipped with the potential to anticipate future. The agent just has to search the nodes that are likely to follow next. When the schema has fuzzy parts, the course of events might be branching there, and the agent is faced with uncertainty.

When the agent constantly updates what is perceived (that is, objects and their position relative to one's own), and constantly maintains and updates an action schema that includes potentially upcoming events (i.e. an *expectancy horizon*), the agent is located in time and space.

Availability is not the only heuristics used here. Up to now, the agent would still be driven externally. But the agent's focus is determined by its inner state, too; by the



(4): The construction of meaning, from Dörner (2001), p.233

want most active. And search in schema space also takes place driven by wants and dangers (i.e. situations inducing wants).

When all is working like expected and a schema fits, the agent might start a *background check*, using idle time for looking at details. This would not be of immediate use, but might result in new knowledge. On the other hand, when the action schema fails, the agent should get alerted, searching memory fast—and coarse. (see Dörner, 2001)

3.4 Meaning

"Meaning [...] is inherent when [an object] invokes thoughts about things or activities, and when these thoughts in turn refer to objects and events in the outside world and to motifs and wants." (Dörner, 2001, p.226) Of course, these connections are only possible when schemata are available. Consequently, any meaning, in this notion, is founded in past events that were associated with lust or aversion (see fig. 4).

Meaning thus arises when in external reality or in the mind schemata get connected: for instance, when two objects (resp. their schemata) are present at the same time; or

when one object in an action schema is missing and bitterly needed; or when one and the same object appears in many different action schemata; or ...

With growing experience, the specific connection patterns might be regarded as a *system of values*, or a mindset. All the experiences in total give account if the agent perceives his world as predictable, if the world tends to be manageable, if many things are associated with lust, or with danger ...

All in all, Dörner (2001) offers an architecture that starts at the very basic beginnings, with small motor sequences for scanning the immediate surroundings. Being parsimonious with assumptions, Dörner constructs a self-regulating system being anchored in space and time, with the ability to construct meaning.

3.5 Motifs and Emotions

So, given this architecture: What is it that a system may *need*, and *want*; in other words, the c*ertain aspects of its inner milieu to stay constant*? Dörner postulates five basic needs—and any such need may become a *motif* when a specific goal to fulfil the need can be named.

For description, Dörner (2001) uses steam boilers as analogy. Any need can be thought of as a reservoir, constantly losing some of its filling. Sensors keep record when the reservoir is about to drain empty (indicating a need). When the tank gets filled again, i.e. a need is satisfied, this is recognized as lust.

Basic needs This includes all things with immediate relevance for physical existence. Fuel/ energy/ food as weel as liquids/ water fall into this category. Also, the avoidance of harm, the prevention of anything that might result in a danger to physical integrity, is part of the basic needs.

Sexuality As sex is not necessary for survival, but a very strong need and crucial for survival of the species as a whole, Dörner regards it as a seperate need.

Affiliation Here the need for interaction with other 'systems' is summarized. This includes helping other 'systems' in trouble, and also getting a positive feedback (*L signals*, for legitimacy) in turn. This need has an inherent ambivalence: It is fundamental for social interaction, but it might also be the foundation for aggression—when 'systems' not belonging to one's group are present. And when legitimacy is a sparse resource, rivalry (e.g. when longing for a peer's attention) might arise.

Certainty In short, this is the ability to understand and to predict the state of the environment—now and in the near future. Our short discussion of space and time showed that a schema for a given situation might be straightforward, or it might be branched and fuzzy.

An agent with a need to minimize uncertainty will be eager to understand the environment, to complete schemata for a given situation.

Competence The need to exert power, to bring about changes in the environment, might be regarded as a need for *competence*. *Self-confidence* is a similiar notion. Being an outcome of past interactions, the reservoir indicates how successful the agent has been in his dealings with the world. Lust, that is, a need fulfilled, raises competence.

Here again, this need is ambivalent: "Usually this means that *troubles* have to be sought. Lust is experienced only when wants are satisfied; for this to happen, a want has to exist in the first place. [...] Striving for lust implies striving for dislike." (Dörner, 2001, p.417)

Most of the time, a system will not be completely balanced with respect to more than one need. A human, for instance, might feel a bit hungry, a bit confused by the envrionment, lack some self esteem and, at the same time, also might be glad about company—or be glad about having sex, too. And most of the time, eating, musing about the environment and having sex won't go together well.

So, Dörner assumes a sub-system that prioritizes needs, with respect to parameters like *urgency* and *importance*. Hence, at any moment, one need is the most important and becomes a *motif*.

In his Ψ theory, Dörner (2001) gives a detailed account of emotions. For our purpose, a rough sketch will suffice. Emotions are no seperate module; they are a constellation of system parameters that modulate processes and behavior.

A very strong motif, for instance, would result in very coarse memory activation and planning. Speed is crucial, no time for details. In contrast, a very important motif should result in the opposite, leading to thorough planning. Both constellations, however, should result in a high arousal, i.e. in a pre-activation of resources.

The motif selection system, memory access and planning, and the extent to which the environment is explored are the most important parameters. When modelled in detail, they can be seen as emotions: they mirror the system's current state and automatically tune important parameters with respect to the current motif.

3.6 PSI and CLARION: a Comparison

The influential CLARION cognitive architecture by Ron Sun (2003, 2006, 2007) also claims to be an integrative model of cognition, metacognition and motivation. By sketching similiarities and differences between the motivational frameworks of Dörner and Sun, the distinctive features of Ψ will get more palpable.

CLARION is built on the observation that "motivations are the foundation of action and cognition" (Sun, 2006, p.80). The framework comprises several subsystems: the action-centered subsystem (ACS), the non-action-centered subsystem (NACS), the motivational subsystem (MS) and the metacognitive subsystem (MCS). Each of these subsystems comprises an explicit and an implicit part; a dichotomy reflecting what is often called symbolic and sub-symbolic processing.

The ACS controls actions—physical ones as well as mental actions—, the NACS maintains explicit and implicit knowledge, the MS provides motivations for perception, action and cognition, and the MCS monitors and directs operations of all subsystems (Sun, 2006).

Each compartment is dichotomic, with rule-based processing and artificial neural networks (ANNs) reflecting the basic modes. Nevertheless, bottom-up generation of explicit representations is explicitely enforced; as well as top-down assimilation, the transfer of symbolic knowledge to the ANN..

For the ACS, this translation process shall be sketched (see Sun, 2006):

- Each explicit rule (specified as $state - specification \longrightarrow action$) leads to the generation of ANN representations by observing actions and outcomes. Such connections are reinforced by Q-learning (see Mitchell (1997)).
- The *Rule-Extraction-Refinement* (RER) algorithm extracts rules from successful actions. Starting from a simple rule, subsequent experiences are used to make the rule more general or, in the case of non-successful outcomes, to make the conditions of the rule more specific. This is, in the end, a *candidate elimination* (see Mitchell, 1997).

Storage in the NACS starts with associative memory, simple mappings of inputs to outputs using ANN. Reasoning here is similiarity-baased. At the top level, explicit knowledge is coded in the form of *chunks*—dimension-value pairs. Here, reasoning is rule based. By linking rules with low-level associations, and by mixing both modes of reasoning, Sun (2006) claims to "capture essential patterns of human everyday (mundane, commonsense) reasoning" (p.89).

At the heart of the MS are *primary needs*, namely *get water*, *get food* and *avoid danger*. Sun (2006) also assumes more 'built-in' drives as to be found in Abraham Maslow's famous *need hierarchy* (see Maslow, 1943). Drawing on these AAN-represented needs, higher rule-based needs might emerge.

Sun (2003) lists six so-called low-level primary drives (like *food*, *water* and *sleep*) and eleven high-level primary drives (as, for example, *affiliation*, *autonomy*, *honor* and *curiosity*), derived from empirical observations. Sun is noting that "the coverage of motivations here may not be complete" (p.7). Additionally, when satisfying primary needs, secondary drives might be acquired. Goals are the "more explicit" (p.10) outcome, that might be activated by more than one drive at the same time, and specific in focus.

In the inner workings of CLARION, the MS serves to main purposes: determine drive strengths given an inner state and sensory input (passing these strengths to the MCS); and gathering goal actions from MCS and ACS to build a goal structure, that feeds as a goal into ACS and MCS.

Comparing[4] Sun and Dörner, we find obvious similiarities: Both authors stress the importance of goals for any AA. Both postulate a tight coupling of perception, memory, motivation and action.

For this paper, however, the differences are of greater importance:

- While CLARION's motifs are integrated, they still are modelled in a seperate module. They are at the heart of goal selection and goal building. In contrast, with the Ψ model needs are inextricably intertwined with every process. They guide perception, influence memory storage and retrieval, and influence actions not only by selescting goals, but also via modulating parameters like arousal.
- 17 primary drives in CLARION: quite a contrast to Dörner's five basic needs/ drives.
- CLARION stresses a low- and high-level-dichotomy in every module; information is able to migrate up- and downwards. Dörner, on the contrary, assumes a continuum where hypothesis driven processes condense information step by step.
- Sun seems to neclegt emotions. Yet, it's hard to imagine the full meaning of *hunger* or *honor* without considering emotions. Ψ offers a way to model emotions as a constellation of a few system variables and need states—fully integrated in processes of perception, memory and action.

[4] Comparison of models and theories, however, is done on a naive basis here, without pondering about issues of *commensurability*.

To sum it up: Ψ relies on less basic needs, motifs are even more integrated into the system, information is consequently built bottom-up, and emotions are an integral system property. So Dörner's model is preferred here, being sparse in assumptions, offering *grounding* mechanisms for information (that is, for symbol generation), and introducing emotions as necessary flavor for motifs.

Maybe these differences arise from different research philosophies. While Sun builds upon empirical observations, Dörner takes a cybernetic stance and consequently builds bottom-up. As Braitenberg (1984) points out: "[... I]t is much more difficult to start from the outside and to try to guess internal structure just from the observation of behavior. [...] A psychological consequence of this is the following: when we analyze a mechansim, we tend to overestimate its complexity." (p.20)

3.7 Principles for an Autonomous Agent

Drawing on Dörner's theory, we can sketch some specifications of our agent-to-be. For this paper, this will be done rather apodictic. For a detailed discussion of strengths and weaknesses of Dörner's framework, and a comparison with theories like *ACT-R* and *SOAR*, one might have a look at Bach (2007).

We might state, however, that Dörner's approach is rather universal, including motifs as well as memory and emotions; and that it starts from scratch, using basic sensorimotor programs as building blocks for higher cognition. Notably, an agent sensu Dörner will be inextricably connected with its environment, as all schemata stem from individual experience.

In Dörner (2001), the agent *James* is described: a small steam-engine driven being, with sensors and motors, and a motif system. James is exploring an island, looking for nutrition, and avoiding harm. It serves as example for a system built in accordance with Dörner's blueprint.

The agent proposed in this paper will be somewhat similiar to James. However, it will be no clone. The goal of this paper is to integrate an inductive learning algorithm. But an agent is more than a mere learning algorithm, and consequently we need some system around it. On the other hand, things should not be too complex, as it is supposed to run on a LEGO brick.

Taking these constraints into account, the system should look like this:

- The agent will comprise a sensoric, a motor and a motivational system
- A protocol memory will store triplets of sensor-motor-motif states

- The agent will use this memory to look for schemata matching the current state
- The agent will have five needs[5], which implies the presence of a rudimentary *lust* indicator:
    1. As a *basic need* the agent will try to avoid physical harm; and, given the possibility, try to regenerate when such harm was experienced
    2. For the sake of *certainty*, the agent will try to explore its environment in order to learn new schemata; or, if *competence* is low, instead of exploring searching for schemata in memory will suffice
    3. Trying to maintain a high level of *competence*, the agent will seek situations of danger and uncertainty, to overcome these obstacles
- An—albeit simple—selector will determine the most pressing *need*
- To include a rough notion of emotions, some variables monitoring the inner state must be present. By assessing the current *motif* and the *need reservoirs*, general *arousal* (like motor speed) and the granularity of memory access is influenced

This will be the skeleton for inductive learning. It is clear that such learning—finding regularities in the environment and making them explicit—will be corresponding to *meaning* as Dörner defines it. Furthermore, it is a meaning that is completely founded in the agent's experience. Yet, when the result is an explicit rule, it—at least to some extent—transcends the agent's ideosyncratic experience.

Most of the general conditions that are postulated here might be regarded rather pragmatic. Implementing motifs, for instance, or handling sensoric and motor data, could be done using standard computational patterns and data structures.

Some of these requirements are outside the scope of this paper. It will, for now, suffice to note that inducitve learning must not be regarded *in vacuo*. It must be embedded in a psychologically grounded framework to claim validity as a cognitive device. Machine learning is only possible when some sort of bias/ constraint allows for inferring structure. Dörner's theory serves as one of these constraints here, specifying basic cognitive-motivatinal-emotional interrelations.

The concept of *memory*, however, deserves a closer look. It is not only at the heart of the agent's inner world. It will also be crucial for an inductive learning mechanism that has to rely on these representations when looking for regularities. So, for theoretical as well as for practical reasons, this component deserves attention. In the next part of this paper, precise requirements for such a memory system will be made palpable.

[5]However, sexuality is omitted here for sake of practical robotic constraints; affiliation won't be regarded for now, as social interaction would make things completely unmanageable.

4 Building a Memory Concept

With many models of human memory 'on the market', the design issue for our AA has to address two basic questions:

- What memory model is suitable?
- Is the model in question plausible from a psychological point of view?

This section will try to shed some light on these issues.

4.1 Getting a Grip on *Memory*

Valentino Braitenberg (2009) states that "the world is represented in the brain (p.147) and identifies *cell assemblies*, neural groups, as bearers of information. These assemblies are shaped by sensory input. The brain is seen as a "stage" on which "objects, living beings and concepts [...] act [...] preferably like the originals in the outer world they represent". However, when and how these representations become more than mere activation patterns in the network, thus allowing for such an abstract play-acting, is not explained. To get more precise, we'll start—literally—at the beginng, adressing the question: How does *memory* develop in humans?

One should think that *memory* is a crucial notion when looking at children's developmental processes. When growing up is about learning and maturation—in many respects—, then a memory is not only a necessary prerequisite, it would also be likely to be affected by developement, and development be affected by these changes, too. However, when looking into the standard textbook of developmental psychology (Oerter and Dreher, 2002), things remain fuzzy. The concept by Klahr & Wallace (explained in detail later on), for example, gets mentioned as a one-liner (Oerter, 2002, p.469).

Other theories described (Oerter, 2002; Schneider and Büttner, 2002) revolve around capacities of assumed memories, around the role of prevoius knowledge, and around descriptions of empirical phenomenons—without stating what, at the bottom, memory *is* (or might be)[6].

Of course: These approaches are by no means fruitless. Textbooks focusing on memory explain a lot of contemporary models (see, for example, Neath and Suprenant, 2003): The *levels of processing* approach by *Craik* and *Lockhart* (Craik and Lockhart, 1972),

[6] This, of course, does not mean that memory concepts are completely neglected in developmental psychology. Yet, they obviously remain descriptive; memory is taken for granted, on not explained bottom-up

the *multi-store model* by *Atkinson* and *Shiffrin* (Atkinson and Shiffrin, 1968) and famous approaches like Alan Baddeley's model (Baddeley, 1997), the "currently most influental view of working memory" (Neath and Suprenant, 2003, p.69). Such models are able to map many so-called *memory effects*[7] and thus have nourished psychological research. However, neuronal mechanisms of *coding* knowledge (and, consequently, the process of *encoding*) remain vague.

When, in contrast, artificial neural networks (ANN) are used, the basic kind of representation used is obvoius. Gasser and Colunga (2003), for example, tried different kinds of ANN for pattern learning using syllables; aiming to model language aquisition in infants. Results were strongly dependent on input's similiarity to the training examples. When filling a pre-defined pattern, using unfamiliar syllables, "the network correctly filled in only one or two of the four syllables at best" (p.253). Although the neural networks used showed some remarkable results in this "seemingly symbolic task" (p.254), it is obvious that no explicit rule was learned. In this case, a discrete *leap* in performance should have occurred. This example shows that ANNs might be promising, but their scope remains limited up to now.

Is it better in computer science? An exact definition of the representational structure is a *conditio sine qua non* for any computer program. The standard textbook by Russel and Norvig (2003), claiming to be "The Intelligent Agent Book" on its cover, dedicates a whole page (out of nearly 1000) to the question how "humans and other animals represent knowledge" (p.243), speaking of "some kind of nonverbal representation". Of course, all AI algorithms presented in this book rely on some sort of memory: For logic-based approaches it's a set of assertions, for statistical approaches it's a set of Bayesian probabilities, ...

Clearly, the memory is shaped according to the informatic approach in question; it is not discussed as a necessary and important concept that should be designed in the first place, with algorithms shaped accordingly. In other words, the approaches are rather pragmatic: focusing on knowledge (i.e. the *content*) rather than on memory (i.e. the *form*).

The present work does not claim to deliver an exhaustive account of human memory. This would not be possible; and, in fact, is not necessary. The point is: There is no single model to rule it all; "neither camp has a monopoly on the desire to explain important phenomena or to solve significant problems" (Gronlund et al., 2007, p.112). Every approach has its strengths and weaknesses. As this thesis is concerned with rule learning using a stream of experience, a memory representation reflecting the chronological stream (rather than assuming distinct compartments) is chosen. Consequently, the rest of this chapter will focus on this aspect, sketching and justifying such an approach. That a

[7] A definition and some examples will follow.

seperation of different kinds of long-term memory is not the only perspective possible has been voiced by Endel Tulving in a seminal article 25 years ago:

> It seems reasonable [...] to assume that [procedural, semantic and episodic memory] constitute a class-inclusion hierarchy in which procedural memory entails semantic memory as a *specialized* subcategory, and in which semantic memory, in turn, entails episodic memory as a specialized subcategory. (Tulving, 1985, p.3f)

4.2 Origins of the *Time Line*

The empirical foundations for the concept of a chronological memory—a *time line*—were laid over 50 years ago. Wilder Penfield (1955), describing electrical stimulation of cortical areas before neurosurgical practice, was able to evoke clear and vivid hallucinations of past experiences in his patients. However, patients were well aware of their present state (being awake on an operation table, to undergo surgery because of temporal lobe epilepsy) at the same time and could describe these hallucinations as if they were watching a movie. The phenomenon lasted as long as the electrical stimulation was given.

All of the recorded accounts were extremely detailed and covered quite unimportant biographic fragments at the same time. They were no mere flashbulb-memories but described whole courses of events. The accounts were given in a speed as if the scene would just happen for real. Consequently, Penfield (1955) considerd it "evident that the brain of every man contains an unchanging ganglionic record of successive experience[8] " (p.67). Therefore, he regarded the temporal cortex as being central for an "anatomical record of the stream of consciousness" (p.68).

Of course, these results contrast with contemporary research where the constructive character of human memory is pointed out (e.g., a whole chapter in Neath and Suprenant (2003) is devoted to re-constructive phenomenons, for example important when judging the reliability of eye-witnesses). After all, Penfield did not cross-check the given accounts with objective biographic information[9].

[8] The basic psychological idea, however, is older. Sigmund Freud, in his famous *Traumdeutung*, already writes: "Das Verhalten des Traumgedächtnisses ist sicherlich höchst bedeutsam für jede Theorie des Gedächtnisses überhaupt. Es lehrt, daß ‚Nichts, was wir geistig einmal besessen, ganz und gar verlorengehen kann' (Scholz, S. 34). Oder, wie Delboeuf es ausdrückt, ‚que tonte impression même la plus insignifiante, laisse une trace inaltérable, indéfmiment susceptible de reparaître au jour', ein Schluß, zu welchem so viele andere, pathologische Erscheinungen des Seelenlebens gleichfalls drängen."

[9] In fact, this wouldn't have been possible anyway: The biographic memory was evoked by stimulating a randomly chosen point on the temporal lobe's surface and thus was arbitrary, and contained

This, however, does not render Penfield's findings useless. Leaving open if the patients recalled real scenes (that are not accesible under normal circumstances) or if they just constructed memories *on-line*: We are pointed at the brain's ability to handle past occurrences in form of a time line.

4.3 An Information-Processing Approach

Klahr and Wallace (1970) started to re-formulate assumptions from developmental psychology in the language of information processing. Building upon theories by Newell and Simon, the authors try to "analyze the information processing requirements of the tasks" (p.362) when looking at children confronted with Piagetian exercises. As a *primary building block*, Klahr and Wallace assume the existence of "an ordered list[10] representing the relative values as a result of the child's experience to date" (p.366). While solving the task, the child is supposed to perform operations on this structure. Higher levels of processing, like *drives* and *motives*, rely on such basic building blocks.

The authors themselves refer to their model as "crude" (p.380), lacking for example an encoder that maps external stimuli to internal representations. However, this model was extended and refined to the *time line concept* (Klahr and Wallace, 1976). The authors, explicitly drawing on Penfield (1955) (and taking critiques into account, e.g. by (Neisser, 1976)[11]), see such a "time-line [...] of fundamental importance in our account of the structure and functioning of [long-term-memory]" (Klahr and Wallace, 1976, p.182). Each node on this chain is a "single production or a production system" (p.183).

The actual implementation is a bit more complex. As Klahr and Wallace (1976) propose five short-term memory buffers (for example, for auditory input), those would somehow have to interact with the time line. When allowing for parallel search, this would result in $5 * n$ search processes, n being indefinite. To tackle this, the time line is split in three tiers:

1. The first tier represents "production and production systems that have been added to the system's [long-term memory] repertoire as a result of consistent sequences detected in the sequential record of activity provided by the time line" (p.186).
2. On the second tier, problem-solving strategies like means-end-analysis are located.

everyday incidents. Finding post-hoc validation sources for these memories would not have been feasible.

[10] *List* is referred to as a data structure in terms of computer science.

[11] Neisser criticized that mechanisms of *growth* are not understood, and thus models fail to grasp such mechanisms like Piagetian accomodation. Yet, Brown (1979) who shares this critique also points out: "I Have a sneaking suspicion that Piaget's theory is a gigantic projective test and that it is possible to find there what one is looking for [...]" (p236)

3. The third tier comprises self-modification processes. When actions have not resulted in consistent sequences, this tier utilizes short-term buffers to fetch and re-combine productions. "[I]t is not inconceivable that [these production's] activity is largely confined to periods of sleep [...]" (p.186).

Each tier is split in several levels, representing "specific or common consistent sequences" (p.186) and thus forming a kind of *context*. Parallel search is restricted to single levels; the tiers and levels, in contrast, are searched sequentially.

The processes outlined here are assumed to rely on an "innate 'functional kernel' "(p.189). Part of this is *consistency detection*, "abstraction of the general pattern underlying the group from the previously learned individual links" (p.190). Although the results will be stored most likely in tiers *1* and *2*, the process itself is located in tier *3*.

Klahr and Wallace (1976) provide "only a crude starting point for a sufficient information processing analysis of 'abstraction' " (p.207). The proposed algorithm should work like this:

- Segment the time line by finding repeating identical tokens. Overlay the resulting sequences, mark identical parts, and repeat the process with the initial tokens of so-far not-consistent sequences.
- By rating the *commonality* (using parameters and measures to be specified for a given system) and length of the resulting *common sequences*, a decision is made

    1. to form a new node representing the common sequence
    2. to try again with shorter sequences
    3. or finally, to abort the process

This simple algorithm, further simplified here, has many shortcomings, as Klahr and Wallace (1976) note themselves. For example, there's no goal-orientation in the process. In the context of this paper, we must add another drawback: Obviously this algorithm relies on existing patterns and is not able to generalize *ad infinitum* by looking for recursive rules. However, we will keep the time line idea and share the opinion that "the notion of sequential regularity detection [is] a fundamental aspect of cognitive development" (p.223).

These ideas culminated in the BAIRN model by Wallace et al. (1987). It specifies the internal structure of *nodes* in the time line. Each node is a production system—in the sense that it has activation conditions and changing knowledge states, which in the end are "based only [on] the results of BAIRN's innate endowment of primitive perceptual and motor nodes" (p.360f). Associations between nodes are based either on temporal contiguity or functional equivalence.

Preprocessing transforms incoming sensoric data (via several short-term buffers), so all kinds of information fit into the time line. The arousal level of the system influences the granularity of node processing.

"The existence of motives is a consequence of the need for directionality in adaptive, flexible, intelligent systems" (Wallace et al., 1987, p.378), and consequently BAIRN is equipped with *motive generators*, assuming innate evaluation functions. Motifs are generated by scanning time line traces, on-line on the moment of trace generation as well as off-line. The latter process allows for "higher-level or compound emotions" (p.380), when a functional relatedness between stored primary emotions is discovered.

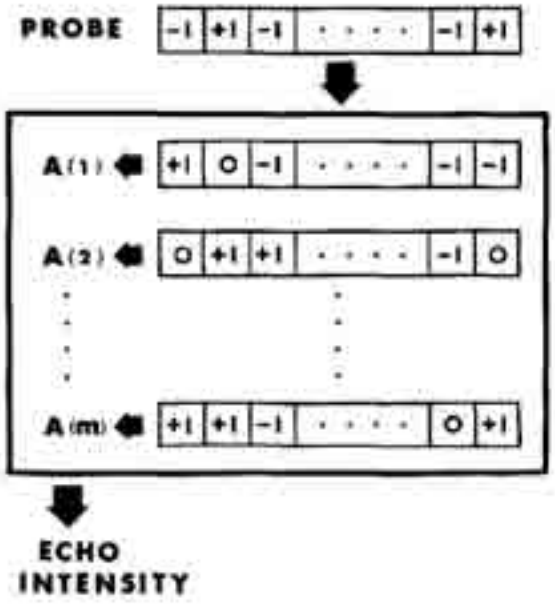
As described in their earlier works, Wallace et al. (1987) assume a regularity detection process that operates mainly during periods of rest or high automaticity. As every node is a production system, for a given state episodes (ithat is, time line parts) are sought that match in their initial conditions. Traces found are then compared regarding *functional equivalence*—i.e., the similiarity of the sequences following the respective initial states is considered. The *candidate elimination algorithm* (see Mitchell, 1997) is then used to generate a new node representing the generality/ specifity of the assumed conceptual relationship.

4.4 Hintzman and MINERVA

This *time line* idea was one of the first computational models of cognitive development. Klahr (1995) recaps strengths and limitations of this model. He also notes that systems like ACT-R, SOAR and connectionist models have followed, addressing a broader range of cognitive phenomena. However, Baddeley (1997), claiming to grasp "the historical continuity of work on the study of memory" (p.vi), doesn't even mention the approach by Klahr and Wallace (1976); neither do Neath and Suprenant (2003).

An idea that in part resembles the time line idea is at the heart of the MINERVA 2 model by Hintzman (1984). Although Hintzman doesn't quote Klahr and Wallace, his view of long-term memory sounds familiar: "[...] a vast collection of episodic memory traces, each of which is a record of an event or experience [...] MINERVA 2 represents an attempt to account for data from both episodic and generic memory tasks within a single system." (p.96)

So it is a single-system memory that stores events' constituents in their chronological order, too. And *any* event will result in a new memory trace, no matter if a very similiar event has been experienced before or not. The main difference: There is no overall time line any more; instead a huge set of seperate traces is built.



(5): Trace activation and retrieval, taken from Hintzman (1986, p.413).

Categories and abstract knowledge are built at the time of retrieval and not during encoding and storage. This implies that "no sophisticated executive routine is needed to decide when and how to tune, reorganize, or abandon memory sturctures" (Hintzman, 1986, p.423).

Memory traces are configurations of primitive properties, like "simple emotional tones and modality-specific sensory features (e.g., basic colors and odors)" (Hintzman, 1986, p.412). There are only two operations for memory access: "A retrieval cue or 'probe' can be sent from [primary memory] to all traces in [secondary memory], and [primary memory] can receive a single reply or 'echo' that emanated back from [secondary memory]." (ibid.)

In MINERVA 2, traces are implemented as vectors storing integer values between -1 and 1 (see figure 5). Learning and forgetting were modelled via a probability that an actual experience gets coded as a trace. Feature lists were coded by a random process assigning -1 and 1 with equal probability. Queries are, in short, simple matrix operations matching *all* traces with a given probe in parallel.

Consequently, a secondary/ long-term-memory might be modelled with just a single line of (in our case, JAVA) code[12]:

[12] Taken from a sample implementation by Ian Neath, provided upon personal request; see also Neath and Suprenant (2003). This 'memory' would be able to store 200 traces, with each trace being an array of 51 primitive features.

(6): An extended MINERVA model suitable for modelling decision making, taken from Dougherty et al. (1999, p.185).

```
int[][] SM = new int[200][51];
```

Without going into details: A computer model of MINERVA 2 is, according to Hintzman (1988), able to reproduce a lot of empirically observed *memory effetcs*, i.e. specific patterns of learning and retrieval under certain conditions:

frequency judgments: The ability to rate how often a certain event has occurred

recognition judgment: Has a given event already been experienced in the past

forgetting consistent with empirically observed forgetting

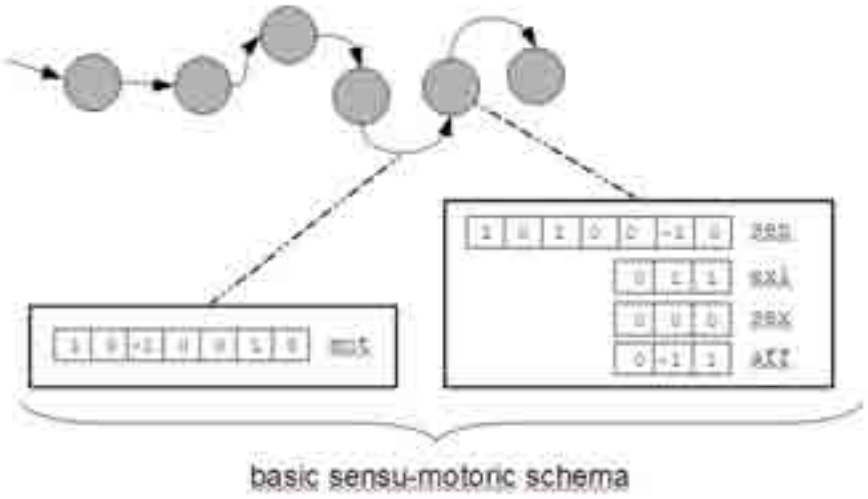
list-length effetcs, i.e. the influence of the number of words in a list-to-be-learned on memory encoding

context effects : The effect of stimuli surrounding the target stimulus (in time and space) while learning and retrieval

This is just an incomplete list of MINERVA's features given by Hintzman (1988). For our purpose it is sufficient to know that the model, albeit simple in structure, is empirically well-founded.

Another remarkable property of Hintzman's model: It lends itself to enhancments. Dougherty et al. (1999) has extended the trace vectors to hold a *hypothesis* and a *context* representation (see figure 6). While retaining the simplicity of its ancestor, the MINERVA-DM (DM for decision making) model can cope with Bayesian inference and other likelihood estimations.

The following section will describe how these concepts might be merged to provide a framework for an AA that incorporates a sound and valid memory concept—that is, in addition, compatible with Dörner's psychological theory.



(7): A basic sensu-motoric schema sensu Dörner, represented in a MINERVA data structure, arranged in a time line.

5 Tying the Ends

The time line idea lends itself to a combination with MINERVA's memory trace representation. At the same time, it will remain compatible with Ψ. Such a union is briefly sketched in this chapter.

5.1 Two Modes of Retrieval

The (quite complex) node specification—each node being a production system—in Wallace et al. (1987) might be replaced with traces sensu Hintzman.

While such a model would retain all the features empirically tested by Hintzman and Dougherty, it would also provide a framework for contiguity-based association for experiences. This would be a necessary prerequisite for regularity detection.

Such a combination would be a *dual process model* as, for example, described by Gronlund et al. (2007). Familiarity decisions would take place as a *probe* sent into the whole long-term store. It can be thought of as a quick and automatic process that results in a rather fuzzy *echo*—suitable for example to be matched against a sensory input trace, and resulting in an all-or-nothing decision about familiarity.

Recollection processes, in contrast, would utilize the time line. Such a slow and conscious[13] search would parse sections of chained events. To keep things simple, we might start with a linked list that connects the stream of incoming experiences in temporal order. Of course, this assumption might later be refined; for example, by assuming different *tiers*[14]. A recollection might start in the present, i.e. at the head of the time-line list; or it might use familiarity probes to find past events. "Despite there being no universally accepted methodology for measuring the contributions of familiarity and recollection, most researchers now agree that two processes are useful for explaining a wide range of phenomena [...]." (Gronlund et al., 2007, p.125). Such a process dichotomy is also at the heart of CLARION (Sun, 2006).

When, in addition, the MINERVA traces are extended (comparable to the decision making extension by Dougherty et al. (1999)), we will be able to map Dörner's ideas, too. We might use a trace to code motoric states; one to code sensoric states; and one to represent changes in Dörner's motivational variables. Postulated triplets of perception, motivation and action would then constitute basic units of memory.

Such a fusion of concepts is illustrated in figure 7. Please note that only existential needs, sexuality and affiliation are coded here: To build a complete AA would stretch the boundaries of this paper too far; the goal is to sketch an architecture that is Dörner-*and* IGOR-compatible. Certainty and competence, however, might (and hopefully, *will* in further research) fit seamlessly into the architecture sketched here.

5.2 Further Enhancements

When the assumption of a simple linked list would be discarded in favor of a graph that allows more complex node structures, certainty could be seen as a function of the number of edges leaving a given node; or, as an evaluation of *echoes* that respond to a given memory probe. Likewise, competence would be a function of the changes in the other motifs' states.

The dual process model, too, could be refined. In the outline just given, the recollection process would have to start at the time line's head—everytime. However, we might just assume that a *probe* does not just return a single *echo*; in addition, it might also return

[13] Here: in a mode of processing requiring most of the agent's processing resources

[14] This assumption—a stream of experience entering the system as a simple linked list (that is, a sequence, without branching)—will reduce complexity here. Yet, the proposed approach remains extendable in the future. As shown in Schmid (2003), more complex data structures (like *directed acyclic graphs* (DAGs) as an output of backward planning) might be linearized under certain conditions. *Data type inference* uses the strucure of such a plan for selecting a folding strategy.

a limited number of links to nodes that are a good match. But what to do with such links?

Dörner hypothesizes no seperate short-term memory; in his model cognitive processes just operate on the first nodes—that is, the head—of the protocol/ time line. Consequently, the links might be added to the list's head, the start of the protocol.

Similiarity search would remain fast and fuzzy. Additionally, it would also provide links to memory traces at any point in the time line. These links could be seen as *entry points*, allowing quite fast access to past event's traces. Any cognitive process thus could make use of these traces, for example in following these links to the matching traces and include them for processing; e.g., for planning.

When the number of accessible nodes at the protocol's head is limited, this possible extension would mirror the empirically found limitiations of short-term memory; and at the same time reflect phenomenons like *chunking*. When a larger part of information is present in memory and retrieved, not the whole representation will fill short-term memory/ the protocol head. Only the link, a single node, would be attached to the list head.

Furthermore, a process of forgetting could be introduced—gradually removing nodes from the time line that are not associated with noticeable changes concerning the agent's needs. However, these nodes would stay in memory and would still reply to a *probe*, contributing to similiarity retrieval.

Such a point of view would require no changes in the memory representations outlined here. The system's structure stays the same, only the retrieval processes are extended, and the time line gets more coarse. Consequently, this crude outline given here will suffice for our purposes.

6 Using IGOR

IGOR[15]—to be more precise, the current version IGOR2—is a program for constructing recursive functional programs from a few non-recursive examples. It is described in various publications, e.g. Kitzelmann (2010); Hofmann (2010); Hofmann et al. (2009); Schmid et al. (2009); Crossley et al. (2009). The following theoretical description is compiled from these publications.

6.1 IGOR—Theory and Application

Using IGOR in a cognitive framework will require to sketch IGOR's basic principles. After that, IGOR's current use in the cognitive domain is depicted.

6.1.1 Algorithmic Foundation

Inductive programming is concerned with automatically generating (recursive) programs when some I/O examples derived from this program are given. This contrasts deductive approaches where a complete and formal specification is given and examples are derived. An example:

```
last [a]     = a
last [a,b]   = b
last [a,b,c] = c
... and so on
```

Obviously, this regularity could be expressed in natural language as: *Take the last element from a given list and return it.* As a recursion, it is straightforward, too: *When a list has only one element, return it; otherwise, discard the first element and test again.* This could be written as follows:

```
last [x]    = x
last (x:xs) = last xs
```

Over the years, several strategies have been developed to automatically discover such regularities[16]:

[15] In this paper, IGOR 2.0.8.0 (available at http://www.cogsys.wiai.uni-bamberg.de/effalip/download.html), written in HASKELL, was used. The binary was compiled on an *WinXP SP3* system using THE HASKELL PLATTFORM (http://hackage.haskell.org/platform/).

[16] Here, only key aspects are given; for every approach there's a development with several extensions and implementations.

- The *analytical functional approach*: "If a function is recursively defned, then evaluating one input (that is not covered by some base case) depends on evaluating other, smaller, inputs by the same program. Hence outputs of smaller inputs go into outputs of greater inputs in a recurrent way. These recurrent relations between I/O examples are discovered and then inductively generalized to a recursive function defnition." (Kitzelmann, 2010, p.32)

  When a property for some instances is observed, and no counter-examples are found, it is inductively inferred that this property holds for all concept's instances. Often, there's no direct mapping from single examples to the (linear) recursive function, so auxilliary variables are used to substitute common subexpressions. Early approaches were restricted to primitives of the used language, for example LISP (that is, *cons*, *car* and a few more).

  A key feature of this procedure: By using induction, it avoids searching the whole hypothesis space; it is example-driven. Yet, background knowledge (in the form of additional rules or rule fragments) is hard to integrate. IGOR I falls into this category, as one of the latest systems.

- In *Inductive Logic Programming* (ILP), "intensional concept descriptions are learned from examples and counterexamples, called *positive and negative examples*" (Kitzelmann, 2010, p.49). Induced programs, examples and background knowledge are all expressed in first-order logic, as a set of clauses. Generally, "ILP is considered as a search problem in a space of (definite) logic programs" (p.52), and by excluding non-consistent programs (and with them, all their generalizations) the search tree is pruned. Different systems (e.g. FOIL, GOLEM and PROGOL) employ different strategies for combining bottom-up (generalization) and top-down (specialization) search.

- *Generate and test* approaches repeatedly generate programs—independently of the examples; these are then used to test if the resulting functions *fit*. Evolutionary algorithms explore the problem space randomized, which makes them ideal for large spaces; at the risk of finding sub-optimal solutions.

  As a subset, *genetic algorithms* use a fitness function to allow for the evolution—i.e. the stepwise building—of programs. Instead of a fitness function, given positive and negative examples can be used to evaluate generated candidates.

  Time-complexity and the possibility of non-terminating programs in the generation process are major drawbacks of these approaches.

Combining the strengths of different approaches, IGOR2 is a state-of-the-art system for inductive learning of recursive functions:

> "In particular, it is an attempt to combine the analytical recurrence detection method invented by Summers with search in program (or rule) spaces in order to overcome the strong restrictions—no usage of background knowledge and strongly restricted program schemas—of the classical analytical approach but without falling back to a generate-and-test search.
>
> The main idea behind IGOR2 is to conduct a global search in a comparatively less restricted program space including [optional] background knowledge, complex recursion schemes, and the automatic invention of auxiliary subfunctions in order to facilitate the reliable induction of non-trivial programs in different domains." (Kitzelmann, 2010, p.73)

IGOR2 is not restricted to predefined types or data structures and represents programs as *constructor systems*—leading to programs resembling modern functional language code. Induction is organized as a *uniform cost search*. At the time of writing, the system still needs a set of the first n (n being function-dependent) examples; extensions to make IGOR2 more relaxed here are in progress.

As every machine learning system, IGOR2 must rely on some sort of bias. Here, as a *restriction bias* programs must be a valid subset of HASKELL. Search space is explored with preferring a minimal number of case distinctions, and patterns must not unify pairwise (*preference bias*).

A very important feature when cognitive modelling is in focus: IGOR2 can induce several interdependent target functions at the same time. The approach thus would still be useful when an agent encounters an environment more complex than our *Kreiswelt*.

Starting point is the *least general generalisation*—that is, a hypothesis with as many unbound variables as is necessary. These variables are utilized by IGOR2 using three main analytical operators (in parallel) as induction cue:

Partition the given example rules, and for these subsets, compute *initial rules*; that is, *minimal generalizations*. In the course of the induction process, these subsets are "further partitioned into more and more, smaller and smaller subsets" (Kitzelmann, 2010, p.81).

New supbroblems are generated by using unbound variables from the right-hand side's subterms.

A sub-function then is used to replace such a given subterm with an unbound variable—so the sub-function becomes a new induction problem. Here, *background knowledge*, that is previously learned or given functional knowledge, can be utilized.

So, for each (sub-) set one rule is sought; until a program is found that is maximally general *and* closed (i.e., without unbound variables). The analytical-inductive approach guarantees termination; as well as a result that is correct with respect to all given examples (for proofs, (see Kitzelmann, 2010). Furthermore, synthesis is very fast compared to generate-and-test-approaches.

6.1.2 A Cognitive Device

Fastness, correctness and guarenteed termination—these properties make IGOR2 an ideal candidate for cognitive modelling. It offers the chance to explain how new rules are built in a system; a question not being trivial even for sophisticated state-of-the-art architectures.

In one experiment by Anderson (2007), for example, modelling the learning of linear equations, the system starts with "declarative representations of the instructions" and "general production rules" (p.21). These instructions and rules are given, not learned. In another design, ACT-R learned "to compress the initial productions [(ten)] into just three [rules]" (Anderson, 2007, p.71).

In contrast to the sub-symbolic workings, the description of ACT-R's symbolic rule-generation process remains rather vague. Anderson (2007, p.156) proposes:

Analogies A transfer from previous, similiar situations. This is quite a demanding cognitive task. (Weller and Schmid (2006) model this process by utilizing *anti-unification* and *E-Generalization*).

Deduction from principles This deduction, of course, requires such principles and raises the question where these rules come from.

Following instructions Here, either a translation from motoric actions to declarative knowledge (in case of imitation learning) or a processing of verbal instructions would be mandatory.

So it seems that inductive and abductive rule generation is still a rewarding field for research. In other words: "Since we are interested in a mechanism to induce general, typically recursive, rules and not in classification learning, we propose to investigate the potential of analytical indictive programming as such a general rule acquisition device." (Schmid et al., 2009, p.163)

In Schmid et al. (2009), IGOR2 is successfully used for learning in domains like *Tower of Hanoi* and *natural language processing*. Yet, the inputs there were not *grounded*—in the sense that the I/O examples were built *within* an agent from basic experience. Consequently, "the problem of automatically transforming traces presented by other

systems (a planner, a reasoner, a human teacher) into IGOR2" (p.167) was noted as question for future research.

This thesis (together with its companion-thesis) aims at bridging this gap. We have come a long way, starting rock-bottom with basic sensoric and motoric activations, thus enabling *grounding*.

We have seen how Ψ might be used as an emotional and motivational framework. Here we go beyond, for example, CLARION where motifs are a heterogenous concept and emotions are not regarded.

We have overcome a weakness of Ψ, using MINERVA as a memory model. The result is a *protocol*, a *time line*, that preserves the ideas by Dörner and Klahr & Wallace, while at the same time allowing for similiarity-based retrieval.

As a final step, we will show how IGOR2 will operate not *on*, but *within* such a system—going beyond ACT-R's rule generation process.

6.2 Implementation

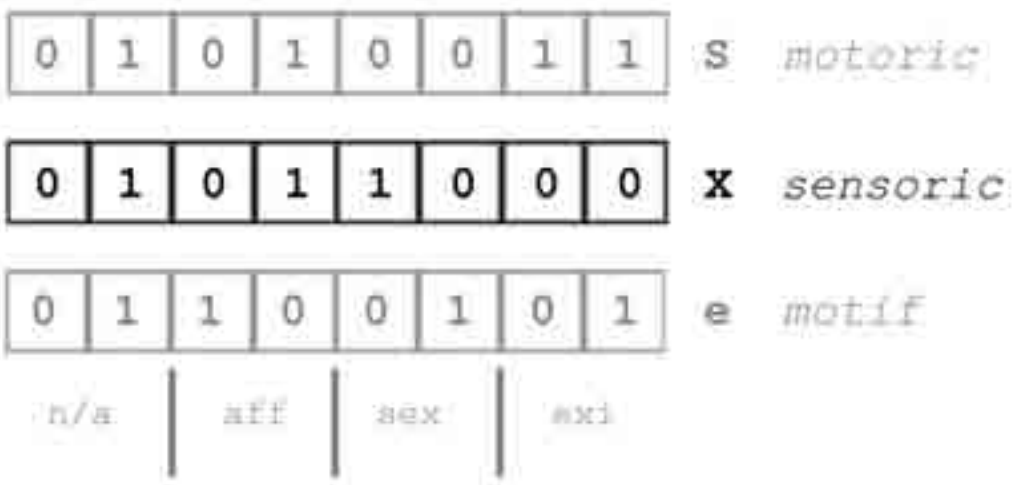
For this thesis, some simplfying assumptions are made considering the input for IGOR. Only sensory events are taken from the incoming stream snippets. Motoric as well as motivational traces are discarded. This will render the following procedure more easy. Yet, the principle would easily scale up when the information omitted here would enter in the same way.

6.2.1 Building Input for IGOR

The thesis mentioned in the second chapter, that is under progress in parallel, aims at *grounding* the sensory input of an AA. There, perception will be categorized in a way that is compatible with Dörner's ideas.

The perceptual information, coming as a sequence, will be coded as a list of ASCII characters (this coding is illustrated in figure 8). This transformation keeps the information, as it allows for a bijective mapping[17]. At the same time, the input is available for IGOR

[17] This convenient coding will, in fact, not preserve *all* information that is possible with MINERVA, as it can not map a -1/ *must not be present* coding. However, this is just a syntactical coding constraint that is accepted for this thesis, and not a constraint *in principle*.



(8): Bijective mapping of MINERVA traces onto letters for IGOR input.

now. *Reward* and *punishment*, also coming from the grounding system, will be coded as *true* and *false*[18].

We further assume that the AA has been active in the proposed *Kreiswelt*. If the grounding system works correctly, this will limit the possible input dramatically: Only streams of identical symbols (each one representing an encountered line) will enter the *time line*, each stream ending with either *true* or *false*—depending on whether the number of lines is odd or even. In other words, there's only one *context*, so all inputs can be considered together[19].

Each of these resulting categorized stream snippets will be written to a file[20] that has a generic HASKELL prologue as a header[21]:

```
module Learn where
  learn :: [Char] -> Bool
```

[18] This seperate coding of reward and punishment is a bit cumbersome. It as necessary because here only sensory traces are regarded. As soon as additional vectors for existential needs, sexuality and affiliation are used, the reward/ punishment nodes will become obsolete.

[19] Context generation, that is identification of situations belonging together as they allow for similiar acting options, is considered in the companion thesis.

[20] There is no objection against *on-line* learning in principle here. Yet, for learning we assume the agent to be in a resting/ sleeping state, with memory streams written to a file.

[21] The mapping of a sequence of MINERVA codes to ASCII, and writing this to a file, is a rather trivial task and will mainly depend on the actual representation of the incoming stream—which is defined in the companion thesis; thus, no sample code is given here.

After the prologue, the seperate streams are written:

```
learn [] = False
learn ['A'] = True
learn ['A','A'] = False
learn ['A','A','A'] = True
learn ['A', 'A', 'A', 'A'] = False
learn ['A', 'A', 'A', 'A', 'A'] = True
learn ['A', 'A', 'A', 'A', 'A', 'A'] = False
```

6.2.2 Generalization with IGOR

IGOR easily generalizes this example (in fact, even with less examples the result would have stayed the same)[22]:

```
learn [] = False
learn ['A'] = True
learn ('A' : ('A' : a0)) = learn a0
```

6.2.3 Transforming Output to a Grammar

So, inductive generalizing from the given examples was, at least in principle under simplifying assumptions, successful. However, we're only halfway done. This information would be useless if it could not be integrated into the memory structure postulated here.

Alternatively, one could think of a separate rule memory as utilized in state-of-the-art frameworks—remember CLARION employing high-level symbolic memory in addition to subsymbolic storage. This would, however, violate the framework set by Dörner and introduce a qualitative gap in our AA's system.

To transform the new knowledge gained by using IGOR back into the time line memory, we will generate a regular grammar[23]. Such a transformation can be performed by a quite simple algorithm[24]:

1. The starting state produces an (arbitrary) non-terminal; say X

[22] For a verbose IGOR2 protocol of this generalization, see appendix A.1.
[23] One might argue that grammar generation is a very strong assumption, questioning the validity of the proposed cognitive model. Such objections will be discussed later.
[24] A proof-of-concept implementation in JAVA is given in A.2

2. X produces the first *true* base case as terminal symbol (arbitrary, but one-to-one and onto)

3. X produces the induction step

   with parameters as terminal symbols (keeping the original character from IGOR as terminal symbol)

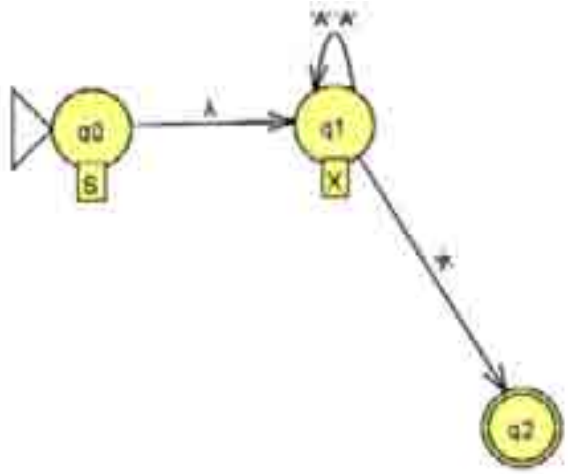
   and the recursive call as non-terminal X

In our example[25], this would result in the following grammar; as the letter A bears information we still need, terminal symbols here are given in ' ', and not as usual as a small character:

```
S -> X

X -> 'A'

X -> 'A''A'X
```

Such a grammer is very useful in two respects:

- It can be used to easily and automatically generate a finite automaton (FA); for our example grammar shown in figure 9, and described in the next part. It's not hard to imagine how such an automaton would fit in the proposed memory concept: The transitions would be re-transformed to MINERVA units, the states would become connections. The simple time line would thus be enriched with recursive structures.
- It can easily be used to *compute* if a given input sequence is produced by that grammar; in other words, if the input fits the learned scheme. However, this is not *decidable*, as for non well-formed input the process would simply hang; so it would be necessary to introduce a meta-process that terminates such processes. And furthermore, this would be a very mighty assumption—a grammar parser would be hard to *ground*. Storing the grammar would require additional forms of memory structure. Consequently, in this thesis, the grammar will be discarded after a memory representation has been generated. Here, it is just a syntactical means for *transforming* information, and not a cognitive device for *using* information.



(9): Automatically generated example automaton, using JFLAP

6.2.4 Creating a Memory Fragment from a Grammar

The little detour of generating a grammar will pay off now. To transform a right- or left-linear grammar into a deterministic finite automaton (only these cases are considered now; however, the principle might be applied to a context-free grammar, too, resulting in a push-down automaton (PDA), albeit such a PDA would not fit that easily into the proposed memory model without further assumptions), standard algorithms are available (see, for example, Rodger, 2006).

The algorithm from Rodger (2006, p.37) is the basic scheme for the following pseudo-code, which describes how to build a memory fragment (i.e. a graph) from the IGOR-output grammar (in this example, a right-linear one)[26]. Automatic tools for such conversions are freely available, for example JFLAP[27]:

1. Start with a right-linear grammar $G = (V, T, S, P)$; V is the set of variables used when the grammar was generated from IGOR; T is the set of terminals, likewise; S is the start variable, and P is the set of productions.
2. Produce a new FA M $(V, \Sigma, \sigma, n_0, F)$ with one more state V as there are variables in G. The initial state is v_0. Let $\Sigma = T$, and let $F = \{v_1\}$, so that v_1 is the only final state. $|\sigma| = |P|$, so there are exactly as many transistions as productions (we describe σ in step 4).

[25] The parentheses in the original IGOR output can safely be ignored here, as building the grammar works from the recursive call 'outwards'.
[26] For this algotithm, too, a proof-of-concept implementation in JAVA is given in A.2
[27] http://www.cs.duke.edu/csed/jflap/

3. Suppose the existence of a bijective mapping $\sigma : V \rightarrow (Q - F)$, where $\sigma(S) = v_0$. That is, (a) each variable corresponds to a unique nonfinal state in the FA so that for any variable A, $\sigma(A) = v_a$ is the corresponding state in the FA, (b) the grammar's start variable corresponds to the FA's initial state, and (c) no variable corresponds to the final state. These conditions are met when IGOR and adjacent grammar generation have worked out correctly.

4. For each production $p \in P$:

   (a) If p is of the form $A \rightarrow \alpha B$, where $A, B \in V$ and α is a possibly empty string of terminals, add a transition to M from $\alpha(A)$ to $\sigma(B)$ on α.

   (b) If p is of the form $A \rightarrow \beta$, where $A \in V$ and β is a possibly empty string of terminals, add a transition to M from $\sigma(A)$ to q_1 on β to σ.

5. Generate a new empty memory graph fragment *MGF*. Follow the first transition in the FA M from the starting state and begin for each $q \in Q$:

   (a) Consider the transition that leads back to the actual state q_a. For the first $v \in V$ encountered, create a new node in *MGF*, remember this node as n_r, connect the new node from the actual node (if it was not the first one at all), and set the new node as actual node n_a. The node will hold the binary representation of the FA's corresponding v as information. For each further v encountered, create a new node in *MGF*, connect the new node from the actual node, and set the new node as actual node n_a. When q_a is reached again, connect n_a to n_r. Mark this transition as processed, e.g. by deleting it from the automaton.

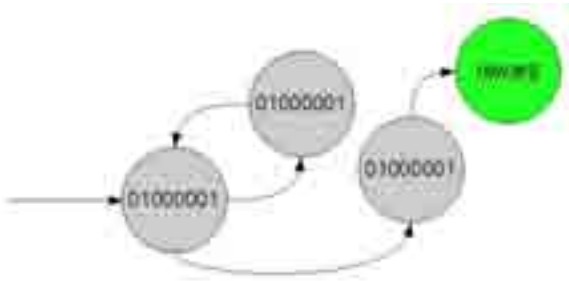
   (b) Proceed with the remaining non-recursive v, by building MGF nodes likewise. n_a is connected to a new memory node, and the new node becomes n_a. In the end, the next automaton state is reached.

   (c) Start again with (a) until the ending state is reached.

6. As a final step, create a new node representing reward and add a transition to it from the actual state.

6.2.5 Connecting the Recursive Fragment with Existing Memory

Integragting the new memory fragment into existing structures is easy: Every time line part that has contributed to rule generation with a *reward* (i.e., with a *true*) gets connected with a seperate copy of the new fragment. The last node of the fragment gets



(10): The resulting *memory graph fragment* (MGF) when the example IGOR output is transferred to a grammar and then processed.

a connection with the reward node (at the same time, replacing the MGF's reward node with the time line's), as illustrated in figure 11.

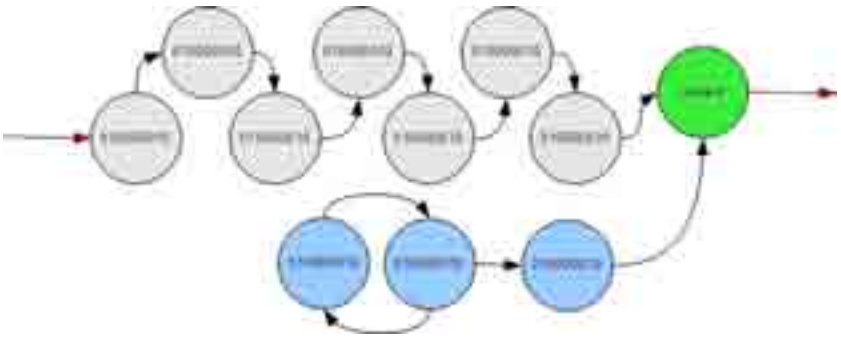
This has some implications:

- The more events with positive outcomes have contributed in learning a given rule, the more often this rule will be represented in memory.
- Only positive/ rewarding situations get associated with rule knowledge.
- Rule knowledge is not available instantly. If the agent wants to 'know' if a rule is available in a given sutuation, he would have to:

  1. send a *probe* into memory to discover similiar situations
  2. check if the outcome of the situations found (that is, the reward) is associated with a rule via back-tracing this node's connections; this would be, so to say, a more 'conscious' mode of searching

  After that, the rules found would still have to be tested if they fit the current situation—by comparing the node's vector traces with the actual perceptual encodings.
- As a last step, patterns in the given situation can be matched with found rules to evaluate possible outcomes; or they can be used to plan further actions.

The last step—matching of rule and stream-of-experience-patterns—could be rephrased as follows: *How can the AA make use of the rules it has learned?* So far this paper was concerned with presenting a seamless integration of symbol grounding, motivation/ emotion, memory and inductive learning. The last step is, without question, a very important one. Yet, a fully satisfying solution reaches (and, indeed, crosses) the boundaries of this thesis. Consequently, it is discussed in the next subsection as a *limitation*.



(11): Connecting the resulting memory fragment to a given time line part.

6.3 Limitations

The presented approach has its strengths; and it has limitations. These will be discussed here.

- Right now, IGOR must be given the first n (n depending on the complexity of the rule behind the examples) examples: no missing examples, no errors. Although there's work in progress to make IGOR more relaxed (personal message by Ute Schmid), at the moment this is a *conditio sine qua non* and restricts recursion detection.

  In a later stage, the AA should be able to find regularities in a environment with noisy input; and IGOR then should return rules with unbound variables, resulting in memory graph fragments with empty feature nodes.

- When utilizing IGOR, only positive examples (i.e. with a *true* on the right hand side) are associated with rules. Metaphorically, the AA would 'know' what to do, but the rules don't cover which sequences are futile.

  However, here the question is: *bug or feature*? From a cognitive point of view, humans construct mental models that only incorporate what is *true* in this model (Johnson-Laird, 1980, 2005); later, these models are enriched with *mental footnotes* specifying which assumptions don't hold. The presented framework would be affected by this cognitive restraint, too.

- Rules can be found only when a memory probe returns a protocol memory part 'near' such a rule, and a search along this part's edges discovers a recursive graph fragment.

  Again, the question is if this is a drawback or even a plausible feature. One does not have to reference Heidegger to postulate a *situated cognition*—that is, to assume that even high cognitive processes are framed by environmental cues. The

framework presented here would only find rules when a memory probe is successful. In the next subsection, a possible extension is sketched that in part overcomes this limitations—in a sound cognitive way.

- So far, no algorithm is presented how to remove rules that have not proven to *hold* in new situations. Such a modification of rules could be necessary when environmental regularities change; or when the examples were erroneous and have suggested a false rule.

  In this paper, this will remain an unsolved issue. A running system would have to undergo thorough empirical testing to see if such false rules pose a problem after all. Right now, as nothing is deleted in memory, new (hopefully correct) rules would just be added. As long as a planning algorithm is used that can handle conflicting rules, we might not run into problems after all. Yet, this question should be addressed empirically.

- The usage of grammars (in the transformation of IGOR output to memory graph fragments) asks for a second look. Is it sensible (from a cognitive point of view), or is it just a *cognitive wheel* that keeps the system running?

  In the framework presented here, the grammar is just performing a syntactical transformation. It is a easy-to-use means for rewriting the rule. It adds no semantic information. So the usage of a *grammar* here does not mean we have to assume a complex grammar handling device in a human's head (although we would meet with Chomsky here) to keep model and reality in touch.

  When we consider the grammatical transformation a limitation, we should also note that a replacement of this auxilliary transformation device would not affect the model's theoretical foundation. The same is true for the mapping of feature vectors onto ASCII characters. It's a mere syntactical mapping, adding no semantical sugar that would question the *grounding* of the manipulated symbols.

Finally: No algorithm is presented here to match given memory fragments with a given rule. This indeed is a major problem, and we will discuss it from two points of view: a computational, and a cognitive. Due to the importance of this question, it will be treated in a seperate subsection.

6.4 Comparing Input and Knowledge

We assume that a segmented stream of experience (say, in the coding outlined here $A\ A\ A\ A\ A\ A$, further denoted as IN) enters the system, either as result of direct

experience or as a time line part from retrieval. Alternatively, this could also be a suggestion from a planner that now has to be checked if it fits a rule in a given context.

We further assume that the memory retrieval via a probe has discovered the memory fragment MF shown in figure 11.

We would now have to match these graphs. Current approaches (like, for example, Brügger et al., 2008) test for *graph isomorphism* and even consider edge and node attributes, and allow wildcards. Although being of exponential complexity, "as the underlying query graphs are often limited in size and due to the fact that the attributes and constraints limit the potential search space for a match, the computational complexity of our algorithm is expected to be still manageable [...]" (Brügger et al., 2008, p.305). This is tempting, as search space indeed is limited by the set of memory fragments found by the probe.

However: That is no solution. Our recursive fragments are graphs, syntactically speaking. Semantically, they represent automatons, incorporating loops that can be run 0 ... n times. After all, that's the advantage of recursive concepts. So we could either:

- unfold the recursive fragment, to produce a set of possible sequences $EMF_{0...n}$ (n denoting the number of loop *pumps*). Each segment EMF_n would have to be matched against IN, if any match is found. If no match is found, the rule does not fit—or the recursion was not pumped often enough.
- detect the recursion in IN and then match IN_{rec} with MF. Inducing the recursion, however, would need an example set matching IGOR's requirements.

Obviously, the second option is not feasible, as there's only one input stream. The first option seems suitable from an algorithmic point of view (although pumping the graph and employing the algorithm by Brügger et al. (2008) is costly for on-line processing), but it would introduce further algorithms and assumptions into our system. This would require a new consideration of cognitve plausibility.

Instead of introducing new graph algorithms, there is another option. Within the bounds of this thesis, it can only be roughly sketched. We introduce two assumptions:

- Whenever IGOR finds a new rule, IGOR additionally creates a new memory node (without inherent features) and connects all situations (i.e., all positive and negative stream examples) that were used for inductive reasoning to this node.
- A search process is able, when given am MGF, to use this connection and to return a set containing all memory streams (here, ASCII sequences) that entered IGOR at that time.

With this extension[28], IGOR can do the job. The IN would just be added to the example set, and if IGOR is able to find a generalization[29] (that is, to return more than just the mere examples), we can assume that the example fits.

This has some implications:

- To stay *within* our system, the retrieved examples would be no mathematical *set*; the nodes might just be copied to the head of the time line (in Dörner's sense, the short-term memory equivalent). Thus, the matching process would—for a moment—consume all 'higher cognitive resources'.
- Rule checking requires no qualitatively new algorithms; in particular, no comparison over graph data structures.
- The AA would become fallible. If by chance the old example set together with the new example made IGOR induce another rule, the difference wouldn't be noticed.
- Another layer of connectivity is added to memory.

So this kind of extension would have consequences that seem quite in touch with human cognition. Of course, this is only a theoretical concept. After a successful implementation, empirical research would have to explore if the anticipated effects are more than a plausible, cognitve gut feeling.

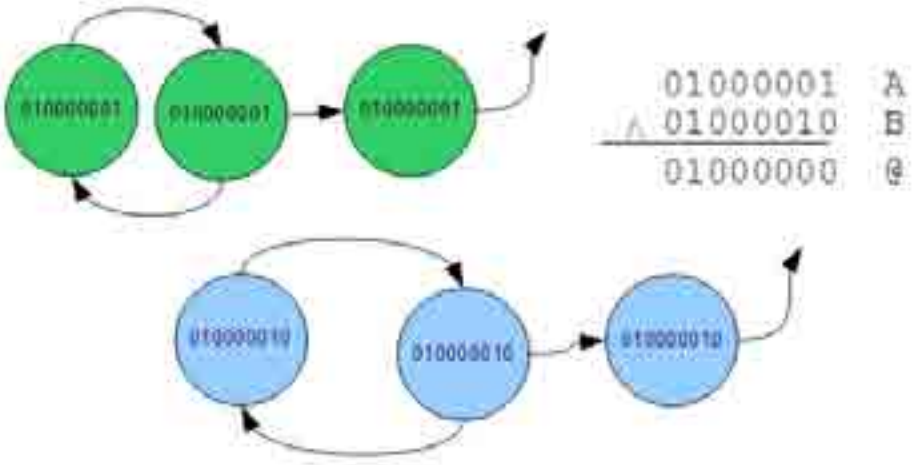
Notably, making rules accessible via IGOR is useful in another respect: This rule acquisition device can make use of *background knowledge*, i.e. other rules. By offering a means for re-building past examples, IGOR can be taken to the next step and use other knowledge (that, of course, would have to be accessible via MINERVA activation). With growing experience, rule acquisition thus could get more and more powerful.

6.5 Ready for Analogy

"Analogy is undoubtedly one of the core features of intelligence." (Weller and Schmid, 2006, p.334) And indeed, our AA is ready for this feature.

[28] In theory, this extension is not necessary. IGOR operates under the *closed world assumptiuon* (CWA), so unfolding gives all positive examples, and all negative examples are given implicitely. Although this would make things easier, it is a very strong assumption about the AA's possible knowledge in general. The fact that everything that is not true is false would be *outside* the system and stain sensible *grounding* in practice.

[29] This is not fully true. IGOR constraints prohibit unifying examples, so the system would need a little extension; to remove the unifying rule (which is already detected, and an error message is written) from the original example set. Such an extension would be useful in another respect; right now IGOR would fail if the AA would make the same experience twice, and consequently would deliver two identical streams for one and the same example set.



(12): Utilizing MINERVA feature vectors in combination with inductive reasoning for analogical reasoning.

Weller and Schmid (2006) use *E-Generalization* for modelling analogical reasoning. In short, this approach goes beyond *anti-unification* (syntactical derivation of a term containing shared structures of a given set of terms) by anti-unifying *regular tree grammars*.

For this thesis, the *regular tree grammar* approach will not be pursued any further. And although anti-unification is already inherent to IGOR, it will not be used directly here. But the sketched memory structure together with inductive reasoning allows for a very simple implementation of analogical reasoning—of transferring (structural) information from one domain to another.

Let us assume a regularity has been found for a structure of feature vectors 01000010, as depicted in figure 12. Let use further assume the same regularity would hold for a very similiar feature 01000001, and there has been no past experience with this feature. Consequently, there would be no exactly fitting memory traces, and a memory probe would retrieve *similiar* traces as result; and chances are good that a memory trace for 01000010 would be activated—and with it, the according rule.

When the AA, in such a case, would give this rule a try and use it for planning and acting, and the result would be positive, he might store the old structural rule with the new feature vector as node for the new situation. Of course, the AA will run the risk of a false transfer, but he gains the ability to transfer regularity knowledge into new situations.

As a further step, the AA might generalize the rule even further. As a 0 in a feature vector represents arbitrary input, a simple bit-wise AND over the vectors of structurally equivalent rules would generate abstract knowledge.

When a rule holds over many different situations and is becoming more and more abstract, we might end up with a rule where all feature nodes are filled with *zeroes*. This would be a rule so abstract that it holds in every situation. The AA here thus could acquire a concept of odd and even that—given enough environmental input—might generalize for *any* situation.

This is, finally, the culmination point. The AA has learned a *concept* that (in its experience) holds for every situation, that generalizes *ad infinitum*—and that, at the same time, is *grounded* in its basic sensu-motoric experiences.

7 Let's Get Cognitive!

Some limitations of IGOR as well as possible countermeasures have already been discussed in the previous section. Here, we will discuss more generally if the proposed framework is sound and novel from a cognitive as well as a computational point of view—focussing on inductive rule learning. In doing so, arguments of Schmid and Kitzelmann (2011) will not be repeated here; instead, new aspects will be introduced.

Naturally, the architecture presented here is designed as a whole. Possible criticism could be directed at all the built-in/ innate features. When thinking of it as a model for human thinking, one should bear in mind: "[I]t is important not to equate innateness with presence at birth or with the notion of a static genetic blueprint for maturation. Whatever innate component we invoke, it becomes part of our biological potential only through interaction with the environment." Karmiloff-Smith (1996, p.10)

Maybe (indeed, surely) successful interaction with a *Kreiswelt* would need no powerful rule acquisition device. When things get only a bit more complex for a human, say a *blocksworld* or the *Tower of Hanoi*, an AA is seriously challenged. Schmid et al. (2009) show how IGOR unfolds its cognitive potential, and this thesis showed that such a cognitive potential needs not be *parasitic* (as Harnad has put it) to a system, but can be *intrinsic*. This is promising for even more complex domains, as, for example, "domain-general sensorimotor development alone cannot explain the acquisition of language" (Karmiloff-Smith, 1996, p.11).

Karmiloff-Smith (1996) postulates a process she calls *representational redescription* (RR). Knowledge representations get, step by step, rewritten; from sequentially and independently stored procedures (*implicit* level *I*) up to linguistic knowledge (level *Explicit-3*).

In her opinion, a connectionist model can't go beyond *I-level* and misses "the specifically human capacity to enrich itself from within by exploiting knowledge it has already stored" (p.192) IGOR is a means for exploiting such knowledge and—remember the easy transformation of output to a grammar and back to a memory fragment—could be seen as a device for generating linguistic/ grammatical knowledge; as discussed in Schmid et al. (2009).

However, one might ask: *Isn't* IGOR *too powerful?* In other words, should we really use complex logic operations like *anti-unification* of a *term rewriting system* in a cognitive model?

In terms of functional equivalence, there's no problem: It's only the output that counts. But even the question of structural equivalence should not be feared. Let us recap what

IGOR does: abstracting from given examples by looking for the *least general gerneralization*; partitioning of the problem space; inventing auxiliary functions for subproblem; and using background knowledge.

Newberg et al. (2004) take, like Dörner, a functional look at the human mind, by asking: "Which abilities needs a mind, to let us think and feel and experience the world in a way we might distinguish as typically human?" (p.71, own translation). They propose *cognitive operators*—the most basic analytic functions of the human mind.

Among these operators are: the holistic operator (to recognize a whole from percieving some parts); the reductionistic operator (to split something in its parts); and the abstracting operator (to induce more general features from given instances). The authors not only give a theoretical account, but also link their ideas to results from neuroimaging.

Noteworthy, too, about Newberg et al. (2004): They also postulate a *cognitive imperative*, stating that a mind not only *can* utilize these operators, but that there's no choice; they *must* be used on any input. This way, they argue, 'irrational' features of the mind, like religious and mythical thinking, stem from rational processes.

Taking this stance, IGOR's inner workings not only *could*, but even *should* be assumed when building a cognitive model. Indeed, the guided-search-approach using unbound variables as induction cues is more plausible than a random generate-and-test-search through the problem space. The remaining question is: How might these analytical and synthetic operators be realized in a (real, not artificial) neural network?

Concerning IGOR's restrictions, the current constraints—the first n examples, no errors—raise the question if such a learning device could be feasible for modelling human cognition. Learning is not error-free. Yet, relaxing IGOR is under progress, so it would return rules with unbound variables when examples are somewhat ambiguous.

And one should keep in mind that for a human, a concept like odd/ even would be shattered if one non-fitting example (say, *three is even*) would be given. The *first-n-examples* restriction is more critical. Here again, plans are to make IGOR invent missing examples—extending its applicability, and introducing (human) fallability.

As has been noted before, it is hard to actually compare models. Criteria would have to be established, and empirical results would have to be compared. At least, some other approaches shall be noted here:

Sun (2000) proposes a hybrid (i.e., symbolic and sub-symbolic) model. He extracts rules from a neural network by using the outcome of an action as success marker, and by *expanding* and *shrinking* the set of possible conditions until an optimum is found. Thus, rules are derived and stored in a seperate rule memory.

Kamruzzaman (2007) presents the RGANN algorithm to generate rules from ANNs. In the network, redundant connections are pruned, values are discretized, and the patterns occuring most frequently are used to generate a rule.

D'Avila (2009); Avila Garcez and Zaverucha (1999), in their CILP (*Connectionist Inductive Learning and Logic Programming*) system, are able to map logic programs to an ANN (using AND/OR-trees). Background knowledge enters the system in the form of explicit rules that are given. For induction, "four simple steps should be followed" (D'Avila, 2009, p.45); these are (very) shortly mentioned.

These approaches have in common that recursive concepts could not be learned. These could not be represented at all, as only acyclic ANNs are used. Rules are restricted to sequential patterns. Sun (2000) assumes a separate rule memory; the other approaches focus on ANNs and don't treat memory as a concept that deserves a consideration independent from its low-level realization. Also, goals (not to mention motivations and emotions) are neclegted. In sum, a *really* unifed model is missing.

Of course, the framework proposed in this paper has its weak spots, too. Short-term memory, for example, is assumed to be the protocol chain's head; but it is not specified any further, and the connection to sensory input is left to the companion thesis. Although motifs and emotions are considered theoretically, the practical implementation is still to be done. And if rule re-building (including utilizing these rules a background knowledge with IGOR) works in practice remains to be shown.

Yet: In principle, the demands by Langley et al. (2009b) mentioned in the introduction are met. And it is important to note: Although *odd* and *even* are human terms unknown to the agent, it has acquired a rule that grasps just these concepts. And these concepts are *grounded*, rooted in the agent's own sensoric experience, and distinguished by the agent's intrinsic need to seek pleasure and to avoid harm.

8 A Question for the Future

This thesis has come to an end. The idea, however, is just about to start. It has been encouraging to see that inductive rule acquisition is more than a *cognitve wheel*: to see that state-of-the-art computer science fits perfectly with cognitive science; and in the end, even with Heidegger.

In any case, the project will be further pursued. At the time of writing, a diploma thesis at the chair of general psychology in Bamberg is on its way; the sketched memory-model based on Ψ and MINERVA will be implemented. And to my knowledge, the *grounding* process developed in this thesis' companion might be at the heart of a PhD-thesis to come.

The claim of this project was and is: To provide a model that offers a seamless fusion of *grounding*, *memory* and higher cognitive mechanisms—goal-driven, with motivations and emotions.

Sun, Anderson, Klahr & Wallace, ... Research on this topic is multifaceted, and further research paired with empirical analysis will contribute to a better understanding of the human mind—by modelling it. Success, however, will only be achieved when the right *questions* are asked. One might run the risk of *empiricism*; for example, of (only) trying to compare an AA's performance when solving the *Tower of Hanoi* with that of a human problem solver.

There's another side, and the integration of Heideggerian ideas, of *grounding* mechanisms, of motifs and emotions, and of regularity detections that might give false results under certain conditions (like IGOR when comparing new input and knowledge) could lead to another point of view: A system, under the *cognitive imperative*, with a complex emotional-motivational system, might behave strangely. Maybe irrational. Maybe superstitious.

Figuratively: Imagine an autonomous agent that, before it enters a resting mode, will at all cost try to cross an odd number of black lines.

Newberg et al. (2004) describe how cognition could be seen as the core of belief; of myth; of religion. It may seem strange at first, but maybe such 'irrational' concepts will be the ultimate touchstone when judging an agent's rationality. Maybe, the AA proposed here has the potential to create a form of such *meaning*—for itself, based on its own grounded symbols. Of course, like Winograd and Flores (1989) note, this agent would not be human. Yet, it could be a bit more *like* a human.

It will be a long way, though, but theologian Heinrich Bedform-Strohm got right to the point. Quote, tongue-in-cheek, at the ceremonial act for Dietrich Dörner's retirement: "I hope, one day, Psi will pray for you."

References

Anderson, J. R. (2007). *How can the human mind occur in the physical universe?* Oxford University Press.

Atkinson, R. and Shiffrin, R. (1968). Human memory: A proposed system and its control processes. In Spence, K.W. & Spence, J., editor, *The psychology of learning and motivation*, pages 89–195. Academic Press, New York.

Avila Garcez, A. S. and Zaverucha, G. (1999). The Connectionist Inductive Learning and Logic Programming System. *Applied Intelligence*, 11:59–77.

Bach, J. (2007). *Principles of Synthetic Intelligence. Building Blocks for an Architecture of Motivated Cognition.* PhD thesis, Universitaet Osnabrueck, Fachbereich Humanwissenschaften.

Baddeley, A. (1997). *Human Memory. Theory and Practice.* Psychology Press, Hove, East Sussex.

Braitenberg, V. (1984). *Vehicles: experiments in synthetic psychology.* MIT Press, Cambridge, MA.

Braitenberg, V. (2009). *Das Bild der Welt im Kopf. Eine Naturgeschichte des Geistes.* Schattauer.

Brügger, A., Bunke, H., Dickinson, P., and Riesen, K. (2008). Generalized Graph Matching for Data Mining and Information Retrieval. In Perner, P., editor, *Advances in Data Mining. Medical Applications, E-Commerce, Marketing, and Theoretical Aspects*, volume 5077 of *Lecture Notes in Computer Science*, pages 298–312. Springer Berlin / Heidelberg.

Brown, A. L. (1979). Theories of Memory and the Problems of Development: Activity, Growth and Knowledge. In Cermak, Laird S. & Craik, F. I., editor, *Levels of Processing in Human Memory*, pages 225–258. Lawrence Erlbaum, Hillsdale.

Craik, F. I. and Lockhart, R. (1972). Levels of processing: A framework for memory research. *Journal of Verbal Learning & Verbal Behavior*, 11(6):671–684.

Crossley, N., Kitzelmann, E., Hofmann, M., and Schmid, U. (2009). Combining Analytical and Evolutionary Inductive Programming. In *Proceedings of the Second Conference on Artificial General Intelligence*, pages 19–24, Amsterdam. Atlantis Press.

D'Avila, G. (2009). *Neural-Symbolic Cognitive Reasoning.* Springer, Heidelberg.

Dougherty, M. R. P., Gettys, C. F., and Ogden, E. E. (1999). MINERVA-DM: A Memory Processes Model for Judgments of Likelihood. *Psychological Review*, 106(1):180–209.

Dörner, D. (1996). Eine Systemtheorie der Motivation. In Kuhl, Julius & Heckhausen, H., editor, *Enzyklopädie der Psychologie. C IV Motivation, Volition und Handlung*, pages 329–357. Hogrefe, Göttingen.

Dörner, D. (2001). *Bauplan für eine Seele*. Rowohlt, Reinbek.

Gasser, M. and Colunga, E. (2003). Pattern learning in infants and neural networks. In Quinlan, P. T., editor, *Connectionist Models of Development. Developmental Processes in Real and Artificial Neural Networks*, pages 233–256. Psychology Press, Hove.

Gronlund, S. D., Carlson, C. A., and Tower, D. (2007). Episodic Memory. In Durso, F., editor, *Handbook of Applied Cognition*, pages 111–136. Wiley, Chichester.

Harnad, S. (1990). The Symbol grounding Problem. *Physica D*, 42:335–346.

Heidegger, M. (1986). *Sein und Zeit*. Niemeyer, Tübingen, 16. edition.

Hintzman, D. L. (1984). MINERVA 2: A simulation model of human memory. *Behavior Research Methods, Instruments & Computers*, 16(2):96–101.

Hintzman, D. L. (1986). "Schema Abstraction" in a Multiple-Trace Memory Model. *Psychological Review*, 93(4):411–428.

Hintzman, D. L. (1988). Judgments of Frequency and Recognition Memory in a Multiple-Trace Memory Model. *Psychological Review*, 95(4):528–551.

Hofmann, M. (2010). IgorII – an Analytical Inductive Functional Programming System (Tool Demo). In andd Janis Voigtländer, J. G., editor, *Proceedings of the 2010 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation*, pages 29–32.

Hofmann, M., Kitzelmann, E., and Schmid, U. (2009). A unifying framework for analysis and evaluation of inductive programming systems. In *Proceedings of the Second Conference on Artificial General Intelligence*, pages 55–60, Amsterdam. Atlantis Press.

Johnson-Laird, P. (1980). Mental Models in Cognitive Science. *Cognitive Science*, 4:71–115.

Johnson-Laird, P. (2005). *Mental Models and Thought*, pages 185–208. Cambridge University Press.

Kamruzzaman, S. (2007). RGANN: An Efficient Algorithm to Extract Rules from ANNs. *Journal of Electronics and Computer Science*, 8:19–30.

Karmiloff-Smith, A. (1996). *Beyond Modularity. A Developmental Perspective on Cognitive Science*. MIT Press, Cambridge, MA.

Kitzelmann, E. (2010). *A Combined Analytical and Search-Based Approach to the Inductive Synthesis of Functional Programs*. PhD thesis, University of Bamberg.

Klahr, D. (1995). Computational Models of Cognitive Change: the State of the Art. In Simon, Tony J. & Halford, G. S., editor, *Developing cognitive competence: new approaches to process modeling*, pages 355–375. Lawrence Erlbaum, Hillsdale, NJ.

Klahr, D. and Wallace, J. (1970). An Information Processing Analysis of Some Piagetian Experimental Tasks. *Cognitive Psychology*, 1:358–387.

Klahr, D. and Wallace, J. (1976). *Cognitive Development. An information-processing view.* Erlbaum, New York.

Langley, P., Choi, D., and Rogers, S. (2009a). Acquisition of hierarchical reactive skills in a unified cognitive architecture. *Cognitive Systems Research*, 10:316–332.

Langley, P., Laird, J. E., and Rogers, S. (2009b). Cognitive architectures: Research issues and challenges. *Cognitive Sytems Research*, 10(2):141–160.

Maslow, A. H. (1943). A Theory of Human Motivation. *Psychological Review*, 50:370–396.

Mitchell, T. M. (1997). *Machine learning.* McGraw-Hill, Boston, MA.

Neath, I. and Suprenant, A. M. (2003). *Human Memory. Second Edition.* Thomson Wadsworth, Belmont, CA.

Neisser, U. (1976). *Cognition and reality. Principles and implications of cognitive psychology.* Freeman, San Francisco.

Newberg, A., d'Aquili, E., and Rause, V. (2004). *Der gedachte Gott. Wie Glaube im Gehirn entsteht.* Piper, München.

Oerter, Rolf & Dreher, M. (2002). Entwicklung des Problemlösens. In Oerter, Rolf & Montada, L., editor, *Entwicklungspsychologie*, pages 469–494. Beltz, Weinheim.

Oerter, R. and Dreher, M. (2002). *Entwicklungspsychologie.* Beltz, Weinheim.

Penfield, W. (1955). The permanent record of the stream of consciousness. *Acta Psychologica*, 11:47–69.

Rodger, S. H. (2006). *JFLAP–an interactive formal languages and automata package.* Jones and Bartlett's, Sudbury, MA.

Russel, S. and Norvig, P. (2003). *Artificial Intelligence. A Modern Approach.* Prentice-Hall.

Schmid, U. (2003). *Inductive Synthesis of Functional Programs. Universal Planning, Folding of Finite Programs, and Schema Abstraction by Analogical Reasoning.* Springer, Berlin.

Schmid, U., Hofmann, M., and Kitzelmann, E. (2009). Analytical inductive programming as a cognitive rule acquisition devise. In *Proceedings of the Second Conference on Artificial General Intelligence*, pages 162–167, Amsterdam. Atlantis Press.

Schmid, U. and Kitzelmann, E. (2011). Inductive Rule Learning on the Knowledge Level. *Cognitive Systems Research*, to appear in special issue Complex Cognition (end of 2011).

Schneider, W. and Büttner, G. (2002). Entwicklung des Gedächtnisses bei Kindern und Jugendlichen. In Oerter, Rolf & Montada, L., editor, *Entwicklungspsychologie*, pages 495–516. Beltz, Weinheim.

Steels, L. (2008). The Symbol Grounding Problem Has Been Solved. So What's Next? In de Vega, M., editor, *Symbols and Embodiment: Debates on Meaning and Cognition*, chapter 12. Oxford University Press, Oxford.

Sun, R. (2000). Beyond Simple Rule Extraction: The Extraction of Planning Knowledge from Reinforcement Learners. In *In Proceedings of the IEEE International Joint Conference on Neural Networks*, pages 105–110.

Sun, R. (2003). A Detailed Specification of CLARION 5.0. Addendum 1: The enhanced description of the motivational subsystem. Technical report, Cognitive Science Department, Rensselaer Polytechnic Institute.

Sun, R. (2006). The CLARION cognitive architecture: Extending cognitive modeling to social simulation. In Sun, R., editor, *Cognition and Multi-Agent Interaction*. Cambridge University Press, New York.

Sun, R. (2007). The motivational and metacognitive control in CLARION. In Gray, W. D., editor, *Integrated Models of Cognitive Systems*. Oxford University Press, New York.

Tulving, E. (1985). Memory and Consciousness. *Canadian Psychology*, 26(1):1–12.

Wallace, J., Klahr, D., and Bluff, K. (1987). A Self-Modifying Production System Model of Cognitive Development. In Klahr, David; Langley, P. . N. R., editor, *Production system models of learning and development*, pages 359–435. MIT Press, Cambridhge, MA.

Weller, S. and Schmid, U. (2006). Analogy by abstraction. *Proceedings of the Seventh International Conference on Cognitive Modeling (ICCM, Trieste, 5-8 April). Edizioni Goliardiche.*, pages 334–339.

Winograd, T. and Flores, F. (1989). *Erkenntnis Maschinen Verstehen. Zur Neugestaltung von Computersystemen*. Rotbuch, Berlin.

A Appendix

A.1 Verbose IGOR2 Output for Odd/ Even Example

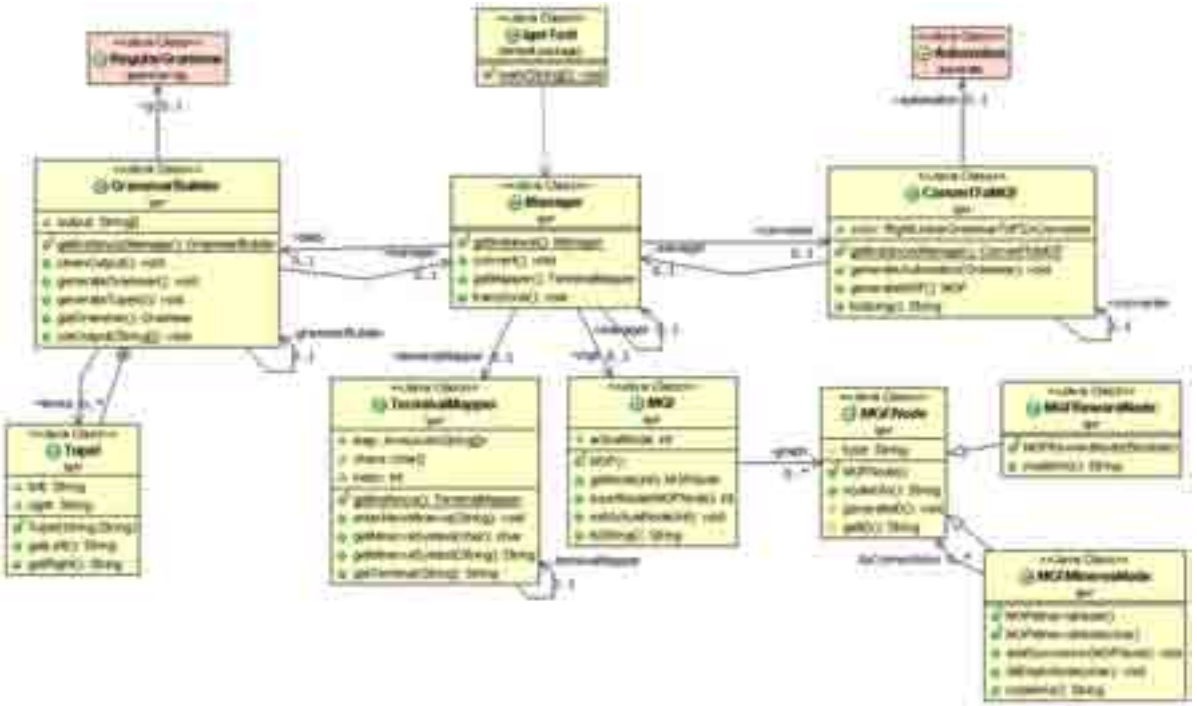
```
*********************************************
*** Initialising Igor ***
*********************************************

Targets              'learn'
Background           <none>
Simplified           True
Greedy rule-splitting False
Enhanced             False
Use paramorphisms    False
Compare rec args     AWise
DumpLog              False
Debug                False
Maximal tiers        0
Maximal loops        -1


*********************************************
*** STARTING SYNTHESIS for TARGET 'learn' ***
*********************************************


- - - - - - - - - - - - - - - - - - - - - - - -
- - Entering Rule-Advancement-Loop for the 1. time - -
- - - - - - - - - - - - - - - - - - - - - - - -

#Hypos: 1

Advancing
  learn a0 = a1 ^[0,1,2,3,4,5,6]
of Hypo (1
        ,1
        ,1
        ,1
        ,([1.0,1.0],1.0))
(*learn,{a0 = a1})

- -
- - Computing matchings
- -

Try for:
  learn a0 = a1 ^[0,1,2,3,4,5,6]

No direct call to 'learn' possible! Need to compute full matchings
(C vs T)
Insufficient matchings! At least one argument had no I/Os at all!

- -
- - Partitioning
- -

Rule: learn a0 = a1 ^[0,1,2,3,4,5,6]

- -
- - Introducing Subfunction
- -

No Ctor at Root, return []

- -
- - Summary
- -


Advancing
  learn a0 = a1 ^[0,1,2,3,4,5,6]
of Hypo (1
        ,1
        ,1
        ,1
        ,([1.0,1.0],1.0))
(*learn,{a0 = a1})
resulted in 1 new hypotheses.

OP1a: FOLD
  Advancements:
[]
```

```
  Hypotheses  :
[]
OP1b: NAIVE_MAP
  Advancements:
[]
  Hypotheses  :
[]
OP2: MTCH
  Advancements:
[]
  Hypotheses  :
[]
OP3: PART
  Advancements:
[({learn ('A' GHC.Types.: a0) = a1 ^[1,2,3,4,5,6]
  ;learn [] = GHC.Bool.False ^[0]}
 ,[])]
  Hypotheses  :
[Right Hypo (2
            ,1
            ,2
            ,1
            ,([1.0,1.0],1.0))
(*learn
,{('A' GHC.Types.: a0) = a1})
(learn,{[] = GHC.Bool.False})]
OP4: CALL
  Advancements:
[]
  Hypotheses  :
[]

- - - - - - - - - - - - - - - - - - - - - - - - -
- - Entering Rule-Advancement-Loop for the 2. time - -
- - - - - - - - - - - - - - - - - - - - - - - - -

#Hypos: 1

Advancing
  learn ('A' GHC.Types.: a0) = a1 ^[1,2,3,4,5,6]
of Hypo (2
        ,1
        ,2
        ,1
        ,([1.0,1.0],1.0))
(*learn
,{('A' GHC.Types.: a0) = a1})
(learn,{[] = GHC.Bool.False})

- -
- - Computing matchings
- -

Try for:
  learn ('A' GHC.Types.: a0) = a1 ^[1,2,3,4,5,6]

No direct call to 'learn' possible! Need to compute full matchings
(C vs T)
Insufficient matchings! At least one argument had no I/Os at all!

- -
- - Partitioning
- -

Rule: learn ('A' GHC.Types.: a0) = a1 ^[1,2,3,4,5,6]

- -
- - Introducing Subfunction
- -

No Ctor at Root, return []

- -
- - Summary
- -


Advancing
  learn ('A' GHC.Types.: a0) = a1 ^[1,2,3,4,5,6]
of Hypo (2
        ,1
        ,2
        ,1
        ,([1.0,1.0],1.0))
(*learn
,{('A' GHC.Types.: a0) = a1})
(learn,{[] = GHC.Bool.False})
resulted in 1 new hypotheses.

OP1a: FOLD
  Advancements:
[]
```

```
  Hypotheses  :
[]
OP1b: NAIVE_MAP
  Advancements:
[]
  Hypotheses  :
[]
OP2: MTCH
  Advancements:
[]
  Hypotheses  :
[]
OP3: PART
  Advancements:
[({learn ('A' GHC.Types.: ('A' GHC.Types.: a0)) = a1 ^[2,3,4,5,6]
  ;learn ['A'] = GHC.Bool.True ^[1]}
 ,[])]
  Hypotheses  :
[Right Hypo (3
            ,1
            ,3
            ,1
            ,([0.6666667,1.0],0.8333334))
(*learn
,{('A' GHC.Types.: ('A' GHC.Types.: a0)) = a1})
(learn
,{[] = GHC.Bool.False
 ;['A'] = GHC.Bool.True})]
OP4: CALL
  Advancements:
[]
  Hypotheses  :
[]

- - - - - - - - - - - - - - - - - - - - - - - - -
- - Entering Rule-Advancement-Loop for the 3. time - -
- - - - - - - - - - - - - - - - - - - - - - - - -

#Hypos: 1

Advancing
  learn ('A' GHC.Types.: ('A' GHC.Types.: a0)) = a1 ^[2,3,4,5,6]
of Hypo (3
        ,1
        ,3
        ,1
        ,([0.6666667,1.0],0.8333334))
(*learn
,{('A' GHC.Types.: ('A' GHC.Types.: a0)) = a1})
(learn
,{[] = GHC.Bool.False
 ;['A'] = GHC.Bool.True})

- -
- - Computing matchings
- -

Try for:
  learn ('A' GHC.Types.: ('A' GHC.Types.: a0)) = a1 ^[2,3,4,5,6]

Direct call to 'learn' possible!

- -
- - Partitioning
- -

Rule: learn ('A' GHC.Types.: ('A' GHC.Types.: a0))
   = a1 ^[2,3,4,5,6]

- -
- - Introducing Subfunction
- -

No Ctor at Root, return []

- -
- - Summary
- -


Advancing
  learn ('A' GHC.Types.: ('A' GHC.Types.: a0)) = a1 ^[2,3,4,5,6]
of Hypo (3
        ,1
        ,3
        ,1
        ,([0.6666667,1.0],0.8333334))
(*learn
,{('A' GHC.Types.: ('A' GHC.Types.: a0)) = a1})
(learn
,{[] = GHC.Bool.False
 ;['A'] = GHC.Bool.True})
```

```
resulted in 2 new hypotheses.

OP1a: FOLD
  Advancements:
[]
  Hypotheses  :
[]
OP1b: NAIVE_MAP
  Advancements:
[]
  Hypotheses  :
[]
OP2: MTCH
  Advancements:
[({learn ('A' GHC.Types.: ('A' GHC.Types.: a0))
      = learn a0 ^[2,3,4,5,6]}
 ,[(learn,learn,LT)])]
  Hypotheses  :
[Right Hypo (3
            ,0
            ,3
            ,0
            ,([0.33333334,1.0],0.6666667))
(learn
,{('A' GHC.Types.: ('A' GHC.Types.: a0)) = learn a0
 ;[] = GHC.Bool.False
 ;['A'] = GHC.Bool.True})]
OP3: PART
  Advancements:
[({learn ('A' GHC.Types.: ('A' GHC.Types.: ('A' GHC.Types.: a0)))
      = a1 ^[3,4,5,6]
  ;learn ['A', 'A'] = GHC.Bool.False ^[2]}
 ,[])]
  Hypotheses  :
[Right Hypo (4
            ,1
            ,4
            ,1
            ,([0.5,1.0],0.75))
(*learn
,{('A' GHC.Types.: ('A' GHC.Types.: ('A' GHC.Types.: a0))) = a1})
(learn
,{[] = GHC.Bool.False
 ;['A'] = GHC.Bool.True
 ;['A', 'A'] = GHC.Bool.False})]
OP4: CALL
  Advancements:
[]
  Hypotheses  :
[]

- - - - - - - - - - - - - - - - - - - - - - - - -
- - Entering Rule-Advancement-Loop for the 4. time - -
- - - - - - - - - - - - - - - - - - - - - - - - -

#Hypos: 2

FINISHED HYPOS, finish Tier:
Maximum tiers reached!
Stopped after entering loop the 4. time in tier 0:

      - - - - START SYNTHESIS WITH - - - -

Targets              'learn'
Background           <none>
Simplified           True
Greedy rule-splitting False
Enhanced             False
Use paramorphisms    False
Compare rec args     AWise
DumpLog              False
Debug                False
Maximal tiers        0
Maximal loops        -1

      - - - - - - - FINISHED - - - - - - -

            learn        in 3     loops
            CPU: 0.0001s


                HYPOTHESIS 1 of 1

learn [] = False
learn ['A'] = True
learn ('A' : ('A' : a0)) = learn a0
```

A.2 Java Implementation of IGOR2 Output Post-Processing



JAVA class diagram für proof-of-concept implementation of IGOR output post-processing. Red boxes indicate classes used from JFLAP.

Here, a sample implementation of the IGOR output processing routines (given in the main text) is shown. It is standard JAVA code that is integrated (as is outlined in class diagram above) in the aforementioned JFLAP automaton framework.

The implementation worked correctly for rule examples containing different MINERVA representations, with sequences (in- and outside of the recursion fragment) of different lengths. The sample here shows such a recursion.

However, it only works for rightlinear output. Empirical research with different kinds of environment would have to show which types of recursion *do* occur, and the routines would have to be extended. Yet, the purpose was to show that the procedure outlined in the main text works correctly.

Console output for sample IGOR function

```
>>>> Starting transformation of IGOR output to grammar
>>>> Pseudocode implementation (page 39)

Original IGOR-Output:
        learn [] = False
        learn ['L'] = True
        learn ('L' : ('L' : a0)) = learn a0

Cleaned IGOR-Output
        ['L']=True
        ('L'::('L'::a0))=a0

Term List contains 2 tupels, now generating grammar ...

Regular grammar, rightregular

        Terminals:
                         a (=original MINERVA symbol was 'L')

        Non-Terminals:
                         S
                         X

        Productions:
                        S -> X
                        X -> a
                        X -> aaX
```

```
>>>> Starting conversion of grammar to MGF
>>>> Pseudocode implementation (page 40ff)

First step: Using JFLAP to build automaton

JFLAP automaton built: automata.Automaton@be990fe0
--> q_0 at (134,33) label: S
        [q_0 at (134,33) label: S] -> [q_1 at (220,483) label: X]: ""
q_1 at (220,483) label: X
        [q_1 at (220,483) label: X] -> [q_2 at (107,372) label: null]: "a"
        [q_1 at (220,483) label: X] -> [q_1 at (220,483) label: X]: "aa"
q_2 at (107,372) label: null **FINAL**

Found start node q0 S
Proceed to successor node q1 X
Building MGF, empty graph
        Found recursion, building memory nodes from automaton's transitions
        >> [q_1 at (220,483) label: X] -> [q_1 at (220,483) label: X]: "aa"
        Actual node now is 0
Added node L to MGF
        Setting node 0 as rNode
        Actual node now is 1
Added node L to MGF
Closing recursion, connecting nodes 1 and 0
        Actual node now is 0
Added node L to MGF
>> Automaton processed, remaining transitions in automaton: 0

>>>>>>>> Memory Graph Fragment created, MGF Info:

Number of Nodes: 4 (8-digit-number is a hash code to identify nodes with equal content)
        Content: L--1102920 , connected to:     L--14737862
        Content: L--14737862 , connected to:    L--1102920      L--444956
        Content: L--444956 , connected to:      Reward Node true--29303146
        Content: Reward Node true--29303146
```

The JAVA classes implemented for this thesis:

```
import igor.Manager;

/**
 * @author Marius Raab
 * Main class to start testing procedure
 *
 */
public class IgorTest {

  public static void main(String[] args) {

    Manager manager = Manager.getInstance();
    manager.transform();
    manager.convert();

  }

}
```

```
/**
 * Main class to handle the whole procedure.
 * Here, data is coordinated and passed between the
 * different converters.
 */
package igor;

public class Manager {

  static Manager manager;

  /**
   * Constructor for Singleton-pattern
   * @return reference to newly created resp. existing instance.
   */
  public static Manager getInstance() {
    if (manager == null) {
      manager = new Manager();
      manager.init();
    }
  return manager;
}

MGF mgf;

GrammarBuilder data;

TerminalMapper terminalMapper;

ConvertToMGF   converter;

/**
 * Utilizing the converter
 */
public void convert() {
  System.out
      .println("\n>>>> Starting conversion of grammar to MGF\n"
          + ">>>> Pseudocode implementation (page 40ff)\n");
  this.converter
      .generateAutomaton(this.data.getGrammar());
  mgf = this.converter.generateMGF();

  System.out.println("\n>> Memory Graph Fragment created");
  System.out.println(mgf.toString());

}

/**
 * Accessing the TerminalMapper to ensure
 * correct mapping of Minerva -- Ascii
 * @return instance of TerminalMapper
 */
public TerminalMapper getMapper() {
  return this.terminalMapper;
}

/**
 * Calling different steps of grammar transformation
 */
public void transform() {
  System.out
      .println(">>>> Starting transformation "
          + of IGOR output to grammar\n"
          + ">>>> Pseudocode implementation (page 39)\n");
  this.setExampleOutput();
  this.data.cleanOutput();
```

```
    this.data.generateTupels();
    this.data.generateGrammar();
  }

  private void init() {
    this.data = GrammarBuilder.getInstance(this);
    this.terminalMapper = TerminalMapper.getInstance();
    this.converter = ConvertToMGF.getInstance(this);
  }

  // Here for convenience the IGOR output is hard-coded
  // Reading it dynamically from a file would be no problem
  // save cumbersome work for syntactically correct parsing
  private void setExampleOutput() {
    String[] example = new String[3];
    example[0] = "learn [] = False";
    example[1] = "learn ['L'] = True";
    example[2] = "learn ('L' : ('L' : a0)) = learn a0";
    this.data.setOutput(example);
  }

}
```

```
/**
 *         This class builds and maintains a surjective
 *         mapping of MINERVA traces (i.e., ASCII chars) to
 *         lowercase- characters that are recognized by
 *         JFLAP {@linktourl
 *         http://www.cs.duke.edu/csed/jflap/}
 */
package igor;

import java.util.ArrayList;

public class TerminalMapper {
  ArrayList<String[]>          map;
  char[]                       chars;
  int                          index;
  // reference to self, to guarantee single instance
  private static TerminalMapper terminalMapper;

  /**
   * Singleton Pattern, to guarantee there's only one
   * TerminalMapper
   *
   * @return new resp. already created instance
   */
  public static TerminalMapper getInstance() {
    if (terminalMapper == null) {
      terminalMapper = new TerminalMapper();
      terminalMapper.init();
    }
    return terminalMapper;
  }

  /**
   * @param minerva an ASCII character that
   * is translated into a terminal symbol.
   * If the character is already mapped, nothing
   * needs to be done; otherwise, a new terminal
   * symbol is taken from the pool and mapped
   */
  public void enterNewMinerva(String minerva) {
    if (!this.containsMinervaSymbol(minerva)) {
      String[] temp = new String[2];
      temp[0] = minerva;
      temp[1] = String.valueOf(this.chars[this.index]);
      this.map.add(temp);
      this.index++;
    }
  }

  /**
   * translates a terminal symbol back
   * to its mapped MINERVA char
   * @param terminal the terminal symbol as char
   * @return the MINERVA char
   */
  public char getMinervaSymbol(char terminal) {
    String term = String.valueOf(terminal);
    for (int i = 0; i < this.map.size(); i++) {
      if (this.map.get(i)[1].equals(term)) {
        return this.map.get(i)[0].charAt(0);
      }
    }
    return ' ';
  }

  /**
```

```
 * translates a terminal symbol back
 * to its mapped MINERVA char
 * @param terminal the terminal symbol as String
 * @return the MINERVA char
 */
public String getMinervaSymbol(String terminal) {
  for (int i = 0; i < this.map.size(); i++) {
    if (this.map.get(i)[1].equals(terminal)) {
      return this.map.get(i)[0];
    }
  }
  return null;
}

/**
 * Returns a reminal symbol for a given
 * MINERVA chars
 * @param minerva char as a String
 * @return the terminal symbol as a String
 */
public String getTerminal(String minerva) {
  for (int i = 0; i < this.map.size(); i++) {
    if (this.map.get(i)[0].equals(minerva)) {
      return this.map.get(i)[1];
    }
  }
  return null;
}

// Tests if a given String is already mapped to a
// terminal symbol
private boolean containsMinervaSymbol(String toTest) {
  for (int i = 0; i < this.map.size(); i++) {
    if (this.map.get(i)[0].equals(toTest)) {
      return true;
    }
  }
  return false;
}

// initializes the Singleton class
private void init() {
  this.map = new ArrayList<String[]>();
  this.chars = new char[20];
  for (int i = 0; i < 20; i++) {
    this.chars[i] = ((char) (97 + i));
    }
    this.index = 0;
  }
}
```

```
/**
 * Class to build a grammar
 */
package igor;

import grammar.Grammar;
import grammar.GrammarChecker;
import grammar.Production;
import grammar.reg.RegularGrammar;

import java.util.ArrayList;

public class GrammarBuilder {

  /**
   * A very basic (inner) class to maintain a LHS-RHS-Tupel
   */
  private class Tupel {
    String left;
    String right;

    public Tupel(String l, String r) {
      this.left = l;
      this.right = r;
    }

    public String getLeft() {
      return this.left;
    }

    public String getRight() {
      return this.right;
    }
  }

  String[]                      output;
  ArrayList<Tupel>              terms;
  RegularGrammar                g;
```

```
Manager                         manager;

// Singleton instance
private static GrammarBuilder grammarBuilder;

/**
 * Singleton Pattern, to guarantee there's only one
 * GrammarBuilder
 *
 * @return new resp. already created instance
 */
public static GrammarBuilder getInstance(Manager manager) {
  if (grammarBuilder == null) {
    grammarBuilder = new GrammarBuilder();
    grammarBuilder.init(manager);
  }
  return grammarBuilder;
}

/**
       * Removes unnecessary white space
       * and replaces : with :: to ensure
       * parsing for MINERVA chars works correctly
       */
public void cleanOutput() {
  if (this.output != null) {
    int count = 0;
    System.out.println("Original IGOR-Output:");
    for (int b = 0; b < this.output.length; b++) {
      System.out.println("\t" + this.output[b]);
      if (!this.output[b].contains("False")) {
        count++;
      }
    }
    System.out.println("\nCleaned IGOR-Output");
    int a = 0;
    String[] temp = new String[count];
    for (int b = 0; b < this.output.length; b++) {
      if (!this.output[b].contains("False")) {
        // To discard function name
        String[] t = this.output[b].split(" ");
        temp[a] = this.output[b]
            .substring(t[0].length() + 1);
        temp[a] = temp[a].replaceAll(t[0], "");
        temp[a] = temp[a].replaceAll(" ", "");
        temp[a] = temp[a].replaceAll(":", "::");
        System.out.println("\t" + temp[a]);
        a++;
      }
    }
    this.output = temp;
  }
}

/**
       * Here the actual grammar generation takes place
       * as described in the main text
       */
public void generateGrammar() {
  // defaults as outlined in the pseudo code
  String startSymbol = "S";
  String recursionVariable = "X";

  // ////////////////////////////////////////////
  // step 1 of pseudocode on page 39
  Production step1 = new Production(startSymbol,
      recursionVariable);
  this.g.addProduction(step1);
  this.g.setStartVariable(startSymbol);

  // ////////////////////////////////////////////////
  // step 2 of pseudocode, the MINERVA symbols of the
  // first rule's parameters are translated into
  // terminal symbols and become the RHS of the second
  // production
  for (int i = 0; i < this.terms.size(); i++) {
    if (this.terms.get(i).right.toLowerCase().contains(
        "true")) {
      // To be on the safe side,
      // in case there's more than one MINERVA symbol
      ArrayList<String> temp = new ArrayList<String>();
      char[] ch = this.terms.get(i).getLeft()
          .toCharArray();
      for (int j = 0; j < ch.length; j++) {
        // parsing for ' '
        if ((ch[j] == 39) && (j + 2 < ch.length)
            && (ch[j + 2] == 39)) {
          temp.add(String.valueOf(ch[j + 1]));
          this.manager.getMapper().enterNewMinerva(
              String.valueOf(ch[j + 1]));
        }
      }
```

```
    String rhs = "";
    for (int k = 0; k < temp.size(); k++) {
      rhs += this.manager.getMapper().getTerminal(
          temp.get(k));
    }
    try {
      this.g.addProduction(new Production(
          recursionVariable, rhs));
    } catch (Exception e) {
      System.out.println("\nNO REGULAR GRAMMAR!");
      System.exit(0);
    }
  }
}

// ///////////////////////////////////////////////////
// step 3 of pseudocode on page 39
for (int i = 0; i < this.terms.size(); i++) {
  if (!this.terms.get(i).right.toLowerCase().contains(
      "true")) {
    String rec = this.terms.get(i).getRight();
    String lef = this.terms.get(i).getLeft();
    lef = lef.replace(rec, "~~~~~");
    ArrayList<String> temp = new ArrayList<String>();
    char[] ch = lef.toCharArray();
    for (int j = 0; j < ch.length; j++) {
      // parsing for ' '
      if ((ch[j] == 39) && (j + 2 < ch.length)
          && (ch[j + 2] == 39)) {
        temp.add(String.valueOf(ch[j + 1]));
        this.manager.getMapper().enterNewMinerva(
            String.valueOf(ch[j + 1]));
      } else if ((ch[j] == 126) && (j + 3 < ch.length)
          && (ch[j + 1] == 126) && (ch[j + 2] == 126)
          && (ch[j + 3] == 126)) {
        temp.add("~~~~~~~~~~");
      }
    }
    String rhs = "";
    for (int k = 0; k < temp.size(); k++) {
      if (temp.get(k).equals("~~~~~~~~~~")) {
        rhs += recursionVariable;
      } else {
        rhs += this.manager.getMapper().getTerminal(
            temp.get(k));
      }
      }
      try {
        this.g.addProduction(new Production(
            recursionVariable, rhs));
      } catch (Exception e) {
        System.out.println("\nNO REGULAR GRAMMAR!");
        System.exit(0);
      }
    }
  }
  this.printGrammarInfo();
}

/**
       * SPlitting terms in tupels
       */
public void generateTupels() {
  for (int i = 0; i < this.output.length; i++) {
    String[] temp = this.output[i].split("=");
    this.terms.add(new Tupel(temp[0], temp[1]));
  }
  System.out.println("\nTerm List contains "
      + this.terms.size()
      + " tupels, now generating grammar ...");
}

/**
 * @return current grammar
 */
public Grammar getGrammar() {
  return this.g;
}

/**
 * @param s IGOR-output to be processed, as String[]
 */
public void setOutput(String[] s) {
  this.output = new String[s.length];
  for (int i = 0; i < s.length; i++) {
    this.output[i] = s[i];
  }
}

// To init the class
private void init(Manager manager) {
  this.terms = new ArrayList<Tupel>();
```

```
    this.g = new RegularGrammar();
    this.manager = manager;

  }

  // Just for convenience, to print info to console
  private void printGrammarInfo() {
    if (GrammarChecker.isRegularGrammar(this.g)) {
      System.out.print("\nRegular grammar, ");
      if (GrammarChecker.isLeftLinearGrammar(this.g)) {
        System.out.print("leftregular");
      }
      if (GrammarChecker.isRightLinearGrammar(this.g)) {
        System.out.print("rightregular");
      }
      System.out.print("\n\n\tTerminals:\n");
      String[] ter = this.g.getTerminals();
      for (int i = 0; i < ter.length; i++) {
        System.out.print("\t\t\t " + ter[i]);
        System.out.print(" (=original MINERVA symbol was '"
            + this.manager.getMapper().getMinervaSymbol(
                ter[i]) + "')\n");
      }
      System.out.print("\n\tNon-Terminals:\n");
      String[] var = this.g.getVariables();
      for (int i = 0; i < var.length; i++) {
        System.out.print("\t\t\t " + var[i] + "\n");
      }
    } else {
      System.out.println("\nNO REGULAR GRAMMAR!");
      System.exit(0);
    }
    System.out.println("\n\tProductions:");
    Production[] prods = this.g.getProductions();
    for (int i = 0; i < prods.length; i++) {
      System.out.println("\t\t\t" + prods[i].getLHS()
          + " -> " + prods[i].getRHS());
    }
  }
}
```

```
/**
 * Class to handle the conversion
 * from a grammar to a Memory Graph Fragment
 */
package igor;

import grammar.Grammar;
import grammar.reg.RightLinearGrammarToFSAConverter;

import java.util.ArrayList;

// Here JFLAP data stuctures are used to get an automaton
import automata.Automaton;
import automata.State;
import automata.Transition;

public class ConvertToMGF {

  // JFLAP
  RightLinearGrammarToFSAConverter conv;

  // JFLAP
  Automaton                        automaton;

  private Manager                  manager;

  static ConvertToMGF              converter;



  /**
   * Singleton conctructor
   * @param manager to enable communication with Manager
   * @return new or already existing instance
   */
  public static ConvertToMGF getInstance(Manager manager) {
    if (converter == null) {
      converter = new ConvertToMGF();

      converter.init(manager);
    }
    return converter;
  }

  /**
   * Takes a grammer and as a first step
   * generates an automaton using JFLAP
   * @param g
```

```
 */
public void generateAutomaton(Grammar g) {
  System.out
      .println("First step: Using JFLAP to build automaton\n");
  this.automaton = this.conv.convertToAutomaton(g);
  System.out.println("JFLAP automaton built: "
      + this.toString());
}

/**
 * Once an automaton has been built,
 * the MGF is cenerated according to the pseudocode
 * given in the main text
 * @return the resultung MGF
 */
public MGF generateMGF() {
  MGF mgf = new MGF();
  State actualState;

  // pointers as used in the pseudocode
  int actualNode = 0;
  int rNode = 0;

  // Getting initial state
  State start = this.automaton.getInitialState();
  System.out.println("Found start node "
      + start.getName() + " " + start.getLabel());
  // Getting all States, and transforming them to a
  //{@see ArrayList()},
  // so processed states can easily be discarded
  State[] allStatesArr = this.automaton.getStates();
  ArrayList<State> allStates = new ArrayList<State>();
  for (int i = 0; i < allStatesArr.length; i++) {
    allStates.add(allStatesArr[i]);
  }
  // First step in pseudocode:
  // Find first state after start
  // and discard initial state
  State q1 = allStatesArr[start.getID() + 1];
  this.automaton.removeTransition(this.automaton
      .getTransitionsFromStateToState(start, q1)[0]);
  allStates.remove(start);
  System.out.println("Proceed to successor node "
      + q1.getName() + " " + q1.getLabel());
  actualState = q1;
  Transition[] t = this.automaton
      .getTransitionsFromState(q1);
  System.out.println("Building MGF, empty graph");

  // Do processing until the automaton is fully parsed
  while (this.automaton.getTransitions().length > 0) {
    // Start with detecting recursion by identifying
    // transition that loops back
    for (int i = 0; i < t.length; i++) {
      if (t[i].getToState().getName().equals(
          actualState.getName())) {
        System.out
            .println("\tFound recursion, building memory nodes"
                + " from automaton's transitions");
        System.out.println("\t>> " + t[i].toString());
        char[] nodesToBeBuilt = t[1].getDescription()
            .toCharArray();
        // Builing MGF nodes
        // from automaton's transitions
        for (int m = 0; m < nodesToBeBuilt.length; m++) {
          actualNode = mgf.insertNode(new MGFMinervaNode(
              this.manager.getMapper().getMinervaSymbol(
                  nodesToBeBuilt[m])));
          System.out.println("\tActual node now is "
              + actualNode);
          System.out.println("Added node "
              + this.manager.getMapper()
                  .getMinervaSymbol(nodesToBeBuilt[m])
              + " to MGF");
          if (m == 0) {
            System.out.println("\tSetting node " + rNode
                + " as rNode");
            rNode = actualNode;
          } else {
            MGFNode tempNode = mgf
                .getNode(actualNode - 1);
            if (tempNode instanceof MGFMinervaNode) {
              MGFMinervaNode minTempNode = (MGFMinervaNode) tempNode;
              minTempNode.addSuccessor(mgf
                  .getNode(actualNode));
            }
          }

        }
        // Finally, close recursion in MGF
        System.out
            .println("Closing recursion, connecting nodes "
```

```
            + actualNode + " and " + rNode);
      MGFNode tempNode = mgf.getNode(actualNode);
      if (tempNode instanceof MGFMinervaNode) {
        MGFMinervaNode minTempNode = (MGFMinervaNode) tempNode;
        minTempNode.addSuccessor(mgf.getNode(rNode));
      }
      actualNode = rNode;
      this.automaton.removeTransition(t[i]);

    }

  }
  // Once the recursion has been transferred,
  // proceed with transition to non-recursive
  // automaton state
  Transition[] trans = actualState.getAutomaton()
      .getTransitionsFromState(actualState);
  char[] nodesToBeBuilt = trans[0].getDescription()
      .toCharArray();
  for (int m = 0; m < nodesToBeBuilt.length; m++) {
    System.out.println("\tActual node now is "
        + actualNode);
    actualNode = mgf.insertNode(new MGFMinervaNode(
        this.manager.getMapper().getMinervaSymbol(
            nodesToBeBuilt[m])));
    System.out.println("Added node "
        + this.manager.getMapper().getMinervaSymbol(
            nodesToBeBuilt[m]) + " to MGF");

    MGFNode tempNode = mgf.getNode(actualNode - 1);
    if (tempNode instanceof MGFMinervaNode) {
      MGFMinervaNode minTempNode = (MGFMinervaNode) tempNode;
      minTempNode.addSuccessor(mgf.getNode(actualNode));

    }

  }
  // The processed transitions and nodes are removed
  // and the process starts again, as stated in the pseudocode
  actualState = trans[0].getToState();
  this.automaton.removeTransition(trans[0]);
}
// At last, when we've reached the final state,
// add the reward node to the MGF
actualNode = mgf.insertNode(new MGFRewardNode(true));
MGFNode tempNode = mgf.getNode(actualNode - 1);
    if (tempNode instanceof MGFMinervaNode) {
      MGFMinervaNode minTempNode = (MGFMinervaNode) tempNode;
      minTempNode.addSuccessor(mgf.getNode(actualNode));
      System.out.println("Finally: Adding reward node");
    }

    System.out
        .println(">> Automaton processed, "
            + "remaining transitions in automaton: "
            + this.automaton.getTransitions().length);

    return mgf;
  }

  @Override
  public String toString() {
    return this.automaton.toString();
  }

  // Initializing this converter
  private void init(Manager manager) {
    this.conv = new RightLinearGrammarToFSAConverter();
    this.manager = manager;
  }

}
```

```
/**
 * Abstract class to represent a memory node
 * In this most basic form, the only property is
 * the ability to return a String representation
 * characterizing this node, i.e. the content and
 * hash code to discriminate different nodes
 * with equal content
 */
package igor;

public abstract class MGFNode {

  private String   id;

  protected String type;
```

```
  public MGFNode() {

  }

  /**
   * Will be overriden by classes implementing
   * this abstract class
   * @return info about this node
   */
  public String nodeInfo() {
    return "Empty Node";
  }

  /**
   * Generates a unique ID for this node,
   * i.e. a String holding node's content
   * and an unique ID
   */
  protected void generateID() {
    this.id = this.type + "--" + this.hashCode();
  }

  /**
   * @return the node's ID String
   */
  protected String getID() {
    return this.id;
  }

}
```

```
/**
 * Class to hold a node with MINERVA content
 * This is a blueprint for a memory node, in our
 * context it just holds the content as ASCII
 * character and a list of nodes it is connected to
 *
 * Implements the abstract MGFNode
 *
 * Later, here a MINERVA feature vector
 * would be stored
 */
package igor;

import java.util.ArrayList;

public class MGFMinervaNode extends MGFNode {

  private ArrayList<MGFNode> toConnections;

  private char               content;

  /**
   * Concstructor for a new node holding
   * MINERVA content
   * @param content the MINERVA char
   */
  public MGFMinervaNode(char content) {
    super();
    this.content = content;
    this.toConnections = new ArrayList<MGFNode>();
    this.type = String.valueOf(content);
    this.generateID();
  }

  /**
   * Adds a successor node
   *
   * @param m the successor node
   */
  public void addSuccessor(MGFNode m) {
    this.toConnections.add(m);
  }

  /* (non-Javadoc)
   * @see igor.MGFNode#nodeInfo()
   */
  @Override
  public String nodeInfo() {
    String output = new String();
    output = "Content: " + this.getID()
        + " , connected to:";
    for (int i = 0; i < this.toConnections.size(); i++) {
      output += "\t" + this.toConnections.get(i).getID();
    }
    return output;
  }
}
```

```
/**
 * Class to hold a node with reward content
 * This is a blueprint for a memory node, in our
 * context it just holds the reward as a boolean
 *
 * In this example, a reward node has no successors
 * Later, when this MGF is integrated in a memory
 * graph, the node would be (as outlined in the
 * thesis) part of a larger graph.
 *
 * Implements the abstract MGFNode
 *
 * Later, here a MINERVA feature vector
 * would be stored
 */
package igor;

public class MGFRewardNode extends MGFNode {

  private boolean reward;

  /**
   * Constructor for a new reward node
   *
   * @param reward the assigned reward property
   */
  public MGFRewardNode(Boolean reward) {
    super();
    this.reward = reward;
    this.type = "Reward Node " + String.valueOf(reward);
    this.generateID();
  }

  /* (non-Javadoc)
   * @see igor.MGFNode#nodeInfo()
   */
  @Override
  public String nodeInfo() {
    String output = new String();
    output = "Content: " + this.getID();
    return output;
  }
}
```