Autonomous Concept Formation: an Architecture-based Analysis

by

Manuela Viezzer

A thesis submitted to the Faculty of Science
of the University of Birmingham
for the Degree of
Doctor of Philosophy

School of Computer Science
Faculty of Science
University of Birmingham
July 2006

Abstract

Our work addresses the problem of autonomous concept formation from a design point of view, providing an initial answer to the question: What are the design features of an architecture supporting the acquisition of different types of concepts by an autonomous agent?

Autonomous agents, that is systems capable of interacting independently with their environment in the pursuit of their own goals, will provide the framework in which we study the problem of autonomous concept formation. Humans and most animals may in this sense also be regarded as autonomous agents, but our concern will be with artificial autonomous agents. A detailed survey and discussion of the many issues surrounding the notion of 'artificial agency' is beyond the scope of this thesis and a good overview can be found in [Wooldridge and Jennings, 1995]. Instead we will focus on how artificial agents could be endowed with representational and modelling capabilities.

The ability to form concepts is an important and recognised cognitive ability, thought to play an essential role in related abilities such as categorisation, language understanding, object identification and recognition, reasoning, all of which can be seen as different aspects of intelligence. Concepts and categories are studied within cognitive science, where scientists are concerned with human conceptual abilities and mental representations of categories, but they have been addressed also in the rather different domain of machine learning and classificatory data analysis, where the focus is on the development of algorithms for clustering problems and induction problems [Mechelen et al., 1993]. The two fields are well distinct and only recently have started to interact, but even though the importance of concepts has been recognised, the nature of concepts is controversial, in the sense that there is no commonly agreed theory of concepts, and it is still far from obvious which representational means are most suited to capture the many cognitive functions that concepts are involved in.

Among the goals of this thesis there is the attempt to bring together different lines of argumentation that have emerged within philosophy, cognitive science and AI, in order to establish a solid foundation for further research into the representation and acquisition of concepts by autonomous agents. Thus, our results and conclusions will often be stated in terms of new insights and ideas, rather than resulting in new algorithms or formal methods.

Our focus will be on *affordance concepts* — discussed in detail in Chapter 4 — and our main contributions will be:

- An argument showing that concepts should be thought of as belonging to different kinds, where the differences among these kinds are to be captured in terms of architecture features supporting their acquisition.
- A description (and partial implementation) of a minimal architecture (the Adaptive Behaviour architecture – AB architecture for short) supporting the acquisition of affordance concepts; the AB architecture is actually a proposal for a sustaining mechanism, in the sense of [Margolis, 1999], for affordances, and makes clear the necessity of a minimal structure for the representation of affordances.

When addressing concept formation in AI, what can be called the 'system level' is often overlooked, which means that concepts and categories are rarely studied from the point of view of a system, autonomous and complete, that might need such constructs and can acquire them only by means of interactions with its environment, under the constraints of its cognitive architecture. Also within psychology, the focus is usually on structural aspects of concepts rather than on developmental issues [Smith and Medin, 1981]. Our approach – an architecture-based approach – is an attempt (i) to show that a system level perspective on concept formation is indeed possible and worth exploring, and (ii) to provide an initial, maybe simple, but concrete example of the insights that can be gained from such an approach. Since the methodology that we propose to study concept formation is a general one, and can be applied also to other types of concepts, we decided to mention broadly 'autonomous concept formation' rather than 'autonomous affordance-concepts formation' in the title of the thesis.

Synopsis of the thesis: Chapter 1 is an introduction to the problem. We discuss, from a rather philosophical point of view, the notion of 'concept' and we illustrate the relevance of concept formation as an AI problem. Chapter 2 reviews the different theories of concepts proposed in the philosophical, psychological and AI literature, in order to give a broad overview of the different positions, the types of questions raised, and the solutions that have been proposed. In Chapter 3 we discuss our methodology and we give an initial overview of the scenario we used to ground our analyses. Chapter 4 addresses the question of whether distinct kinds of concepts can be individuated and on which basis. We argue for the superiority of a processing-based distinction and account for taxonomic concepts, goal-derived concepts and affordance concepts as distinct concept kinds. Chapter 5 presents the AB-architecture, a minimal architecture for the acquisition of affordance concepts. Finally Chapter 6 discusses the contributions of our work, its limitations and the possibilities for future research.

Acknowledgements

First of all, I would like to thank my supervisor Aaron Sloman for his help and encouragement: he has been guiding me through a research path that was sometime hard to find, he has been teaching me both philosophy and engineering, and he has given me the freedom to pursue my own ideas, even when they did not convince him very much, to give me the chance to learn from my own mistakes. At the beginning I thought this was a waste of time, but at the end I must recognise it is the best way to get a real grasp on the matter: I am grateful to him for all this! I also thank all my colleagues at the School of Computer Science for providing a stimulating research environment, and the members of my Thesis Group, John Barnden and Mark Ryan, for their comments on draft parts of this work and for their encouragement. A special thanks to Catriona Kennedy, who helped me a lot at the beginning of my work and always shared with me her ideas, and to Padma Reddy, for her friendship and her help in finding my present job.

I am grateful to Kees Nieuwenhuis, Managing director of the DECIS lab, where I am now employed, for giving me the opportunity both to finish my PhD and to continue working on concept formation within the lab. I also want to thank all my colleagues here at DECIS for making the working environment pleasant, friendly and truly multidisciplinars.

Finally, my thanks go to Mathias for his love and outstanding patience with all my emotional ups and downs, and of course *un grazie anche a mamma e papà per tutto il loro affetto e per aver sempre rispettato le mie scelte, anche quando mi hanno portata lontano.*

Contents

List of Figures

List of Tables

Part I

Problem definition and literature review

Chapter 1

Concept formation as an Artificial Intelligence problem

Our work addresses the problem of autonomous concept formation from a design point of view, providing an initial answer to the question: What are the design features of an architecture supporting the acquisition of different types of concepts by an autonomous agent?

Autonomous agents, that is systems capable of interacting independently with their environment in the pursuit of their own goals, provide the framework in which we study the problem of autonomous concept formation. Humans and most animals may in this sense also be regarded as autonomous agents, but our concern will be with artificial autonomous agents. In the near future, humans and artificial agent systems are foreseen to engage in (even closer) collaboration. These organisations are named *actor-agent communities*, in which collaboration between multiple participants, including humans and artificial agents, takes place for the realization of a common mission or for the support of a shared process [Wijngaards et al., 2006].

Actor-agent communities (AACs) require analysis as a whole, including the triple: actors, agents and their 'niche' [Hoffman et al., 2002, Sloman, 2000a], where niche is characterised by requirements, constraints and opportunities. The intense collaboration between actors and agents places additional requirements on the development of artificial agents: among others, the agents need to 'understand' humans - as communication is a pre-requisite for collaboration. Exploring the design features of concept formation architectures and identifying concept kinds that can be common to both human and artificial minds are means via which shared situation awareness and shared meaning can be facilitated.

Unfortunately, despite the wide agreement on the importance of concepts as major constituents of human cognition, the nature of concepts and their ontological status are controversial (as it is often the case with fundamental theoretical constructs), no unified, comprehensive and well-established theory of concepts exists [Laurence and Margolis, 1999], and different empirical findings seem to support different, and even incompatible, views on the matter. Therefore, we start our research with a discussion of some fundamental questions regarding the very same notion of concept, above all in relation with concept formation. This will serve as an introduction to the problem, and as a basis for further analyses. We then illustrate the relevance of concept formation as an AI problem.

1.1 The tricky concept of CONCEPT

To start with, we examine the role of concepts as structures that mediate the referential properties of human language, because it is such role that concepts have most often played both in the philosophical and in the psychological tradition. When this interface role is captured in realistic terms, the dynamics of concepts as cognitive structures is lost. Realistic semantics, however, can be and has been opposed to cognitive semantics: in Section 1.1.1 we summarise some of the criticisms that have been addressed to the realistic position, and we describe the cognitive approach that has been proposed as a substitute. The main advantage of cognitive semantics is that it provides criteria to characterise 'natural' properties, in the sense found in [Lewis, 1986]. Lewis' notion of a natural property or relation is one that is maximally "fundamental" and "carves nature at the joints". Examples might include the properties of fundamental physics and spatio-temporal relations, while negations and disjunctions of natural properties, properties such as 'grue' and 'bleen',[1] or 'being ten feet from Tony Blair', etc., are not natural because they do not correspond to real universals, and do not appear in the canonical formulations of best physics or regimented common sense.

Another advantage of cognitive semantics is that it makes room for a pragmatic account of concept formation in terms of the interactions of a cognitive agent with its environment. This last issue is discussed in Section 1.1.2 and constitutes our introduction to the notion of *concept formation*, which is the main concern of this thesis.

1.1.1 Concepts at the interface between language and the world

Very often concepts are thought of as strictly connected with language and meaning: the analytical tradition in philosophy reflects this approach [Carnap, 1956, Wittgenstein, 1961, Quine, 1963b]; psychologists working on concept acquisition and categorisation study how language is used by people to infer which concepts are in their minds, how they are acquired and how they work, e.g. [Rosch, 1973, Smith and Medin, 1981]; concepts are seen as linguistic meanings within cognitive semantics [Jackendoff, 1990, Langacker, 1987] and more recently [Gärdenfors, 2000]. [Jackendoff, 1990] provides a good overview of the situation:

> There is a fundamental tension in the ordinary language term *concept*. On one hand, it is something out there in the world: "the Newtonian concept of mass" is something that is spoken of as though it exists independently of who actually knows or grasps it. Likewise, "grasping a concept" evokes comparison to grasping a physical object, except that one somehow does it with one's mind instead of one's hand. On the other hand, a concept is spoken of as an entity within one's head, a private entity, a product of the imagination that can be conveyed to others only by means of language, gesture, drawing, or some other imperfect means of communication.
>
> Precisely the same tension has been discussed by Chomsky (1986) with respect to the term *language*. He differentiates the two poles as "E-language" (externalized language, the language seen as external artifact) versus "I-language" (internalized language, the language as a body of internally encoded information). (p.9)

[1]These were introduced by Nelson Goodman in the late 1940s as examples of time-dependent, non-projectible properties: 'grue' means green until time x and blue afterwards, while 'bleen' means blue until time x and green afterwards.

Jackendoff adopts Chomsky's terminology and speaks of "E-concepts" versus "I-concepts", he then elaborates a theory of I-concepts called Conceptual Semantics that explains how linguistic expressions are related to concepts. Such a theory is needed because language is taken to be a vehicle for expressing mental contents. Jackendoff's contrast between concepts as objective things that can be grasped, and as private entities, was also made in [Frege, 1892], where the notion of 'Sinn' is used to refer to the objective notion of 'sense' or 'concept' that could be shared by many people, and is contrasted both with other notions of meaning that were more private and with the extensional notion of 'Bedeutung'. Popper also made a related distinction in [Popper, 1972] between World 2, the world of private mental entities, and World 3, the public world of objective knowledge that grows over time as people are born, live and die.[2]

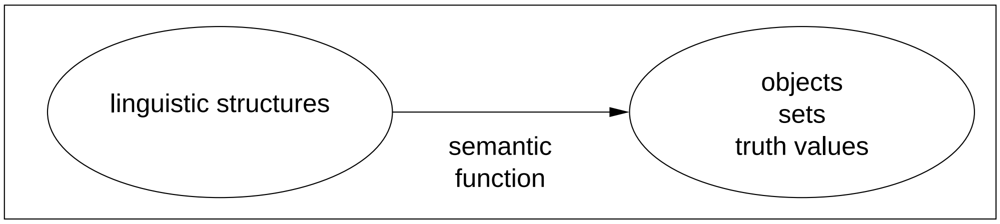
Two things emerge clearly from the above quotation: 1) concepts are identified with the meanings conveyed by language, hence they are strongly tied to language; 2) there is a tension in the ontological status of concepts between a realist conception (E-concepts) and a cognitive conception (I-concepts) and such tension parallels an analogous one concerning language itself; since the tension concerns concepts as meanings, it is a tension between a realistic semantics and a cognitive one.

A very good analysis of the tension between realistic semantics and cognitive semantics is offered in [Gärdenfors, 2000]. According to Gärdenfors, realistic semantics are either extensional or intensional (see Figure 1.1 for a schematic representation of the two semantic mappings), but in both cases the objective is to formulate *truth conditions* for the sentences in the language: "By specifying truth conditions, one intends to describe the semantic *knowledge* of competent speaker. A consequence of this approach is that the meaning of an expression is *independent* of how individual users understand it." (p. 153) Among the pioneers of this approach were [Tarski, 1944, Kripke, 1959, Montague, 1974].

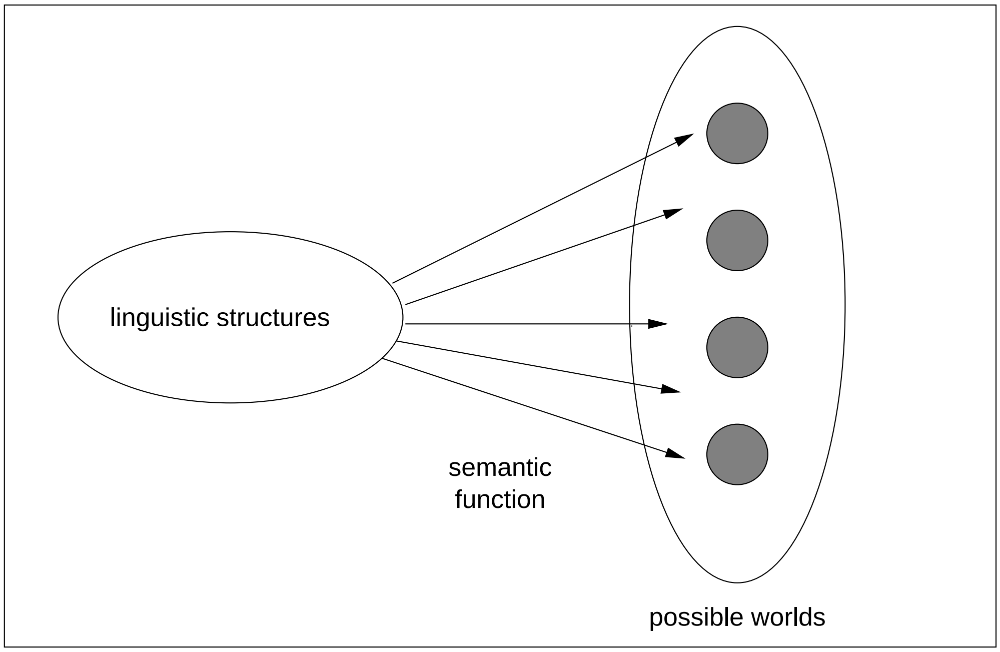
To understand the limitations of realistic semantics, it is worth considering the realistic analysis of *properties*, since the capacity to predicate, that is to assign properties to objects, is central to the conceptualising ability. In this kind of semantics, a property is either a set of objects (extensional case) or a function from possible worlds to sets of objects (intensional case), that is a set of objects indexed by a possible world. In both cases, we end up with sets of objects, so for example the property of being red will be the set of all red things, the property of running will be the set of all events of running, the property of being a ball will be the set of all balls, and so on. Even though this picture constitutes the background of mathematical logic, it has not gone without criticisms, the most severe ones being that it can hardly account for *inductive reasoning* [Goodman, 1955] and that it does not work as a theory of the *meaning* of predicates expressing properties [Putnam, 1981].[3] Another

[2]World 1 for Popper was the physical world.

[3]Putnam's example begins with the sentence 'A cat is on a mat' where "cat" and "mat" refers to cats and mats as usual. He then defines the predicate "cat*" as "x is a cat* iff some cat is on some mat and some cherry is on some tree and x is a cherry, or some cat is on some mat and no cherry is on any tree and x is a cat, or none of the previous holds and x is a cherry" and the predicate "mat*" as "x is a mat* iff some cat is on some mat and some cherry is on some tree and x is a tree, or some cat is on some mat and no cherry is on any tree and x is a mat, or none of the previous holds and x is a quark". Given these definitions, it is possible to show that 'A cat is on a mat' is true in exactly those possible worlds where 'A cat* is on a mat*' is true. According to intensional semantics, this implies that the two sentences have the same meaning. This conclusion however is counterintuitive because it clashes with the idea that the meaning of a sentence should somehow depend on the meaning of its parts. Putnam's conclusions are that truth-conditions for whole sentences underdetermine reference.



The extensional view.



The intensional view.

Figure 1.1: The components of realistic semantics. Adapted from [Gärdenfors, 2000].

problem is that, from a cognitive point of view, such an account does not provide any cue to the psychologist who tries to explain the perception of two objects as similar, as having some common property: within the realist position there is no concern for the cognitive agent, hence mental contents simply do not come into play. There are only two things in the picture: language and the world, and the meaning of language is given in terms of world states.

In order to escape the above criticisms, a different approach is possible (see Figure 1.2) within which the relation between language and the world is mediated by the mental contents of a cognitive agent. This is the framework of cognitive semantics: "The core idea is that meanings of linguistic expressions are *mental entities* — meanings are elements of the cognitive structures in the heads of the language users. Language itself is seen as part of the cognitive structure and not an entity with independent standing." (p. 154) [Gärdenfors, 2000]. Within the cognitive framework, three things are involved: language, the cognitive agent (who is also a user of the language) and the world. Language is situated inside the mind of the agent, so that it constitutes one form of mental content. Besides language, however, there are also other cognitive structures in the mind of the agent, namely concepts. Concepts are there because language primarily refers to them, and only *through them* it refers to the world.

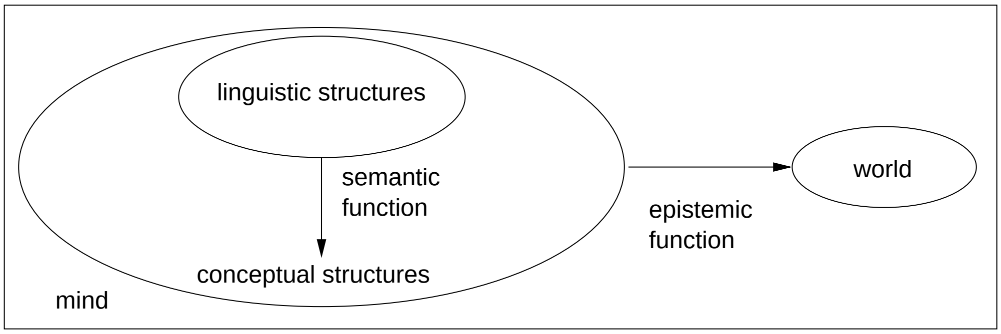


Figure 1.2: The components of cognitive semantics. Adapted from [Gärdenfors, 2000].

The main advantage of a cognitive semantics lies in the fact that it provides cognitively motivated criteria to characterise *natural properties*. As we said, the realistic treatment of properties fails to characterise their 'intended meaning', as argued in [Putnam, 1981]. Putnam's conclusions are that truth-conditions for whole sentences underdetermine the reference of predicates, because there are *infinitely many* possible different interpretations of the predicates of a language in terms of functions from objects to propositions. If we take such an "abundant" construal of properties and relations, on which we have a property for every set of objects and a relation for every set of tuples, no matter how arbitrary or miscellaneous the set, then, from the cognitive point of view, most of these properties and relations are worthless, and in this sense can be said to be unnatural. As made clear by Gärdenfors: "We, as cognitive agents, are primarily interested in the *natural* properties — those that are natural for the *purposes* of problem-solving, planning, communicating, and so forth. These are the properties that can be used in inductive reasoning, the properties we can learn and the properties that get a foothold in natural language, in the sense that we give them names."(p. 67) [Gärdenfors, 2000]. The theory of conceptual spaces offers a model for natural properties, or at least for a subset of them, since the author himself recognises that "the analysis of functional properties is an enigma for my theory" (p. 98) [Gärdenfors, 2000]. Other models are provided for example by [Jackendoff, 1990, Talmy, 1988, Langacker, 1987], but a detailed analysis and critical comparison of these models lies outside the scope of this thesis.

In the light of the above discussion, we adopt a cognitive approach to semantics and more precisely we take an internalist stance on the nature of concepts, and consider them as internal representations dependent, for their form and content, on the cognitive processes of organisms (natural or artificial) and on the specific types of interactions between organisms and their environment. In the next Section we review some central questions that remain open for any cognitive approach to semantics, and in particular what we call the 'learnability question'. It is this last question that profits, as we will argue, from the architecture-based framework that we propose.

1.1.2 Communication, concepts without propositions and the 'learnability question'

As we discussed in the previous Section, a common approach to the study of concepts involves the investigation of language, because language is an observable manifestation of thought: differences in the linguistic constructs are taken to be indications of the existence of differences in mental contents (different kinds of mental contents, e.g. propositions *vs.*

concepts; but also different kinds of concepts), a theory of the mental contents is then proposed, and the challenge is to show that the proposed theory of mental contents accounts for the supposed linguistic differences. An example is the distinction between *proposition* and *concept*: if a proposition is taken to be the meaning of a complete sentence (something that can be either true or false), then a concept could be any unit of meaning smaller than a proposition.

The first problem arising when language and meaning are taken to be *primarily* mental contents is to explain how language can be used for communication, as it surely can. What is in need of explanation is how individual conceptual structures can become social. Even though we are aware of the importance of this problem, finding a solution to it is not among the goals of this thesis. We therefore only briefly summarise some of the research work that has been done in this direction. Cognitivism has been attacked mainly on the basis that concepts and beliefs cannot be individuated without reference to the world [Putnam, 1975, Putnam, 1988]. Among philosophers, an attempt to counter these attacks relies on the distinction between individual meanings and *social meanings*, where the latter are determined from the former together with the structure of linguistic power that exists in the community [Gärdenfors, 1993]. And a further argument in defense of cognitivism builds upon the idea that *sharing* a language in a community imposes constraints on individual cognitive representations [Gärdenfors, 2000]. On the engineering side, there have been several attempts to build systems able to acquire language from scratch: the Talking Heads experiment reported in [Steels, 1999] was a pioneering work aimed at showing how a large population of robots would be able to generate and self-organise a shared lexicon as well as the perceptually grounded categorisation of the world expressed by this lexicon, without human intervention or prior specification. Other experiments addressed the problem of the induction of grammar, either exploiting the same language game model used in the Talking Heads experiment, where linguistic structures arise in the interaction between agents, or relying on a vertical transmission of the linguistic behaviour (such as genetic transmission, learning from a teacher, or both). It seems, however, that the exploration of the origins of language is still in its initial phase; as pointed out in [Steels, 2000]: "Despite several intriguing simulations and robotic experiments, most of the work is still before us. More work is needed on experiments with dynamical environments in which the robots can perform actions as part of the language games. Many aspects of grammar, such as the spontaneous formation of new levels mapping form to meaning and including the origins of new syntactic and semantic categories, have hardly been touched upon. Most semantic domains grammaticalized in the world's natural languages have not yet been studied."

A second problem relates to the fact that conceptualisation abilities may be present also in the absence of linguistic abilities and explicit propositional attitudes. Do lions use propositions when they stalk their prey? Are propositions involved in the vigilant behaviour of zebras? Probably not, but some form of conceptualisation might well be involved. Studies on animal cognition have spanned from perception and memory (studies on vision and the processing of spatial and temporal information, but also studies of migration patterns, orientation, navigation, and the ability to be trained) to communication (comprehensive overviews are [Vauclair, 1996, Bekoff et al., 2002]), providing many data in support of the claim that sometimes the best way for animals to deal with changing social and non-social environment is to store information when it is available and extrapolate from it later, rather than respond mechanically, in a stimulus-response manner [Allen and Bekoff, 1997]. With respect to linguistic abilities, however, matters are less clear-cut: whether the behavioural plasticity of the few captive primates who have been intensively studied is or is not significantly related to the behaviour of wild members of the same species is a question still open for future research [King, 1999]. And very often the crucial linguistic component that animals are believed to lack is exactly the syntactical one, as pointed out in [Brooks, 2002]:

> What separates people from animals is syntax and technology. Many species of animals have a host of alert calls. [...] Some chimpanzees and gorillas have learned tens of nouns, a few adjectives and a few verbs, expressed as signs or symbols. They have sometimes put these symbols together in new ways, like "water bird" to refer to a duck. But they have never been able to say anything as sophisticated as "Please give me the yellow fruit that is in the bag." That requires syntax. Chimpanzees and gorillas do not have it.
>
> Irene Pepperberg has raised and trained Alex, the famous African gray parrot for over twenty years [...] Alex has been heard to say as his trainer is evidently leaving for lunch, "I'm just going for lunch and will be back in ten minutes." But this is just Alex parroting what he has heard the trainer say in these circumstances before. Alex has never said, "I see that you are going out for lunch. I expect you'll be back in ten minutes." That requires syntax. Alex does not have it. (p. 3)

Without providing a definite answer about the relations between conceptualisation and the ability to express mental contents using a language, we would like here to bring the focus on a third related issue. The 'learnability question', as described in [Gärdenfors, 2000] in relation to semantics, is the question of how the coupling between linguistic expressions and their meanings can be learned. For cognitive semantics meanings are mental entities, hence a person learns the meaning of an expression by associating it to a cognitive structure. Since both linguistic expressions (their internal representation) and other cognitive structures are mental entities, but encoded with different representational means, the learnability question seems to reduce to a problem of translation, as noted by Jackendoff: "[T]o talk about what we see, information provided by the visual system must be translated into a form compatible with the information used by the language system. So, the essential questions are: 1) What form(s) of information does the visual system derive? 2) What form of information serves as the input to speech? 3) How can the former be translated into the latter?" (p. 90) [Jackendoff, 1987]. What we would like to stress here is that, even in the case something like a translation manual were available, the 'learnability question' would not yet be settled: we would still have to explain how the mental entities that do not belong to the language system (that is, concepts) are formed. Unless we embrace innateness, our explanation needs to refer to the interactions of the cognitive agent (a person or a robot) with the world, and our hint at the lion and his possible mental contents is just to remind us that what triggers conceptualisation is more likely to be the need of providing further guidance when acting in the world (so that action will be successful), rather than the possibility to evolve a language, and to enter a communication process.

The relation between the concepts and the world is an epistemological one, and it can be accounted for either in a pragmatic way or through some kind of correspondence theory. Within a pragmatic approach a concept reflects a representational need that might have arisen to solve problems when acting in the world, while for a correspondence theory a concept actually *represents* the world. We briefly sketch Thompson's critique of correspondence theories, as described in [Thompson, 1995], to motivate our adoption of a pragmatic approach.

In his paper, Thompson argues that "the biological function of colour vision is not to detect surface reflectance, but to provide a set of perceptual categories that can apply to objects in a stable way in a variety of conditions." Computational models of vision such as the one proposed in [Marr, 1982] focus on the phenomenon of colour constancy and aim at showing how the retinal image can provide indicators of surface spectral reflectance, thus embracing a correspondence view of perception. There are however both psychophysical phenomena (e.g. surface colour metamerism) and phenomenological aspects of vision (e.g. seeing the clear daytime sky as blue, even though the sky is not a surface) that cannot be accounted for by a theory of vision as the function of detecting surface reflectance. In particular, if colour vision is "the ability to see visual qualities belonging to the phenomenal hue categories red, green, yellow, and blue" then it is possible to show that "there is no distal-based scheme that can generate the basic chromatic categories of human colour vision [...] plus the resemblance and difference relations among the hues that result from the opponent structure of these categories (red-green, yellow-blue)." After extensive analysis and discussion of evolutionary data on trichromacy, Thompson refutes the position that takes colour constancy as indicating that the biological function of colour vision is to detect objects by recovering their surface spectral reflectance, and opts for an alternative position, according to which the biological advantages of colour vision (in particular trichromacy for primates) are revealed by the disabilities experienced by colour-blind individuals:

> In scenes where the ambient light varies randomly — for example, when the background comprises dappled foliage and/or different reflective surfaces lying at varying angles to the illuminant — there are three perceptual abilities that human dichromats find particularly difficult: (i) detecting certain coloured objects; (ii) segmenting [...] the visual scene; and (iii) identifying particular objects or states.
>
> Consider how these three tasks can be perceptually and ecologically combined. In monkeys, trichromatic vision facilitates the detection and identification of orange/yellow fruits against the green forest background. To detect and identify these fruits the visual scene must be segmented into the elements that belong together, and colour and lightness are particularly useful attributes in linking these elements. The orange/yellow fruit colour also helps to identify the state of the object by indicating conditions beneath its surface — for example, that it is ripe.

Thompson's argument is built upon evidence from the field of visual perception, and therefore it entails, at least for visual perception, a pragmatic account of the epistemological relation between concepts and the world: rather than simply representing the objective patterns of the distal environment, perceptual categories are generated to guide behaviour in various ways, depending on the things that exemplify them and their significance to the animal.

1.2 Concept formation as a requirement for intelligence

In the previous Section we saw how a cognitive approach to semantics clearly establishes the ontological status of concepts as cognitive structures internal to the agent. If the agent uses a language, concepts provide the meanings for the linguistic entities, but what remains to be explained is the epistemological relation between concepts and the world. It is this relation that we want to capture by addressing the problem of *concept formation*. Thompson's work provided an argument in favour of a pragmatic solution. In this Section we provide further support to a pragmatic approach by discussing the role of autonomous concept formation in the framework of AI.

Concept formation is not a 'standard' AI topic — for example, there are no specific chapters devoted to it in well-known AI handbooks such as [Russell and Norvig, 1995] or [Nilsson, 1998] — and it is mainly regarded as a particular Machine Learning problem. AI researchers, however, have stressed the centrality of the notion of *domain knowledge* and require models of such knowledge to be dynamically refined, enriched and/or revised for intelligence to be robust. On the one hand, the growing interest for ontologies in AI is a symptom of the need to investigate concepts as the primitive building blocks of these models. On the other hand, whenever the ability to learn models is required, the related ability to acquire concepts can be seen as a necessary further requirement.

1.2.1 Modelling as a key feature for intelligence

The goal of AI can be stated in terms of *understanding* intelligent behaviour *in order to build* intelligent systems. According to this definition, AI as science, concerned with the investigation and modelling of the general principles of intelligent systems, and AI as engineering, concerned with designing and implementing new kinds of machines or systems able to do things previously done only by humans or animals, are just flip sides of the same coin, as argued in [Sloman, 1998c], [Sloman, 1999] and [Sloman, 2000b]. Our work stems from an engineering approach, and our hope is that the practice of designing and implementing a virtual agent able to autonomously form concepts about its environment may help throwing some light on the general principles that ground such an ability.

Intelligence is hard to characterise, and many tentative definitions have been proposed in the literature. In [Newell and Simon, 1990] the authors suggest the ability to solve difficult problems (e.g. difficult mathematical or logical problems) and more precisely possessing such an ability in a domain-independent way. In [Russell and Norvig, 1995] the characterising ability is the ability to act in a rational, goal-oriented manner, where an agent is said to be rational if the sequence of actions it plans to perform could eventually lead to the achievement of its goals, given its knowledge of the environment.[4] And [Genesereth and Nilsson, 1987] speak of the ability to anticipate the environment and the consequences of one's own actions.

Even though the previous definitions have different flavours, it is possible to find one common feature among them: the reference to the posession, implicit or explicit, of a model of the world by the system qualified as intelligent. In other words, a system would earn the attribute 'intelligent' if it were to act *as if* it already knew about the possible results of its

[4]An important distinction has been made (see for example [Simon, 1982]) between rationality *tout court* and bounded rationality. In order to achieve rationality, agents have to plan. But planning takes time and consumes resources, and may therefore result in a loss of performance. This suggests that an agent should only plan when the expected improvement outweighs the expected cost and no resources should be expended on making this decision. However, in order to do this, an agent would have to be omniscient. Now, real agents are very far from being omniscent, thus designers must face the so-called meta-level control problem for a resource bounded rational agent, i.e. the problem of how to approximate this ideal, without consuming too many resources in the process. It is important to notice however that the issue of knowledge representation must be raised both for ideal and for real rational agents.

action(s). Captured in these terms, intelligence imposes a requirement for a system to have at its disposal a certain amount of knowledge concerning its own goals and the environment, in order to anticipate the possible changes in the environment and the consequences of its own actions.

Part of the effort of AI has concentrated in making available to a computer-based system descriptions or pictures that correspond in some salient way to the world. This effort constitutes a relevant subfield of AI, with a name of its own: knowledge representation (KR). The underlying hypothesis is that the system could manipulate (internally) those representations in order to build new ones, and then use both to guide and control its behaviour. This hypothesis has been well-captured in [Smith, 1985]:

> Any computational embodied intelligent process will be comprised of structural ingredients that a) we as external observers naturally take to represent a propositional account of the knowledge the overall process exhibits, and b) independent of such external semantical attribution, play a formal and essential role in engendering the behavior that manifests that knowledge.

A system's knowledge base (KB) is the set of data structures (lists, arrays, records, graphs, etc.) that for example a human being can interpret as propositions expressing what the system knows or believes. From a historical point of view, the focus on knowledge representation and the perception of the KB as a critical component of any system derived from the attempt to make systems more and more flexible, robust and modular. Such an attempt also made clear that a system should "know" something about its own operations and structures, which in other words means that a rational agent should also be reflective.

According to an early survey of the KR field [Brachman and Levesque, 1985] the research can be split into two main themes:

- inventing suitable KR languages and arguing why certain languages are better than other;
- providing access to facts implicit in the knowledge base, by means of a suitable inference mechanism performing some form of reasoning.

Most of the knowledge representation enterprise is still concerned with inventing languages and inference mechanisms to express what an artificial system knows, such as those discussed in [Karp et al., 1995, Donini et al., 1996, Chaudhri et al., 1998, Fensel, 2000], while an issue that has been partly neglected concerns the investigation of the techniques that are required to analyse a particular domain to determine what concepts are important in that domain and how they are interrelated. Such techniques however play an essential role, since even assuming that a certain representational language and inference mechanism are at hand, this is still not enough for building an actual knowledge base, because the representational language does not say *what should be represented.*

1.2.2 From Knowledge Representation to Ontology Engineering

In order to build an actual knowledge base, a particular domain has to be investigated to determine what concepts are important in that domain and how they are interrelated. Such investigation leads to the notion of *ontology of a domain*.

We can think of an ontology as the theory that is in the mind of the knowledge engineer while actually designing a knowledge-base for a particular system: it specifies, from the designer point of view, what there is in the world, which are the properties and relations of things, processes and events, what are the causal connections, what are the laws, etc., for the system to use. The previous sense is sometimes narrowed, so that the term 'ontology' refers only to the most general concepts needed to specify such theory: "A number of narrow domains can be tackled by similar techniques, but domains such as shopping in a supermarket seem to require much more general representations. [...] Concepts like time, change, objects, substances, events, actions, money, measures, and so on are important because they show up in one form or another in every domain. Representing these very general concepts is sometimes called ontological engineering." (p. 217) [Russell and Norvig, 1995]. In this sense an ontology is the specification of what is thought to be common to many different domains of knowledge. The concepts represented in an ontology are therefore considered to be very general concepts, in the sense that they are not peculiar to a specific domain, but rather are present and play a significant role in many different domains.

The attempt to solve the technical problems connected with making knowledge-based software potentially shareable and reusable, gave ontologies the role of formal software specification tools, as made clear in [Gruber, 1993]: "Like conventional applications, knowledge-based systems are based on heterogeneous hardware platforms, programming languages and network protocols. However, knowledge-based systems pose special requirements for interoperability. Such systems operate on and communicate using statements in a formal knowledge representation language. They ask queries and give answers. They take 'background knowledge' as an input. And as agents in a distributed AI environment, they negotiate and exchange knowledge. For such knowledge-level communication, we need conventions at three levels: representation language format, agent communication protocol, and *specification of the content of shared knowledge.* [...] For establishing agreements about knowledge, such as shared assumptions and models of the world, ontologies can play a software specification role."

Ontologies, as tools for knowledge sharing and reuse, received much attention some fifteen years ago, with the first 'big' ontological projects appearing at the beginning of the 1990s (such as Cyc [Guha and Lenat., 1990], LOOM [MacGregor, 1991], and Ontolingua [Gruber, 1992]) and special workshops and sessions dedicated to ontological engineering issues being held in the main conferences (ECAI, IJCAI and AAAI) since the mid-nineties.

Existing ontologies vary in a continuum from lightweight taxonomies (thesaura or conceptual vocabularies) to rigorous formalizations, and a distinction can be made between ontology engineering tools and ontology applications. Ontology engineering tools vary from ontology representation languages, to tools supporting the translation from one representation language to the other, to special purpose systems (based on a specific language) that practically help creating and maintaining ontologies. Looking at the work done in the field, four main research trends can be individuated:

- A great effort is still put into the development of formal languages for ontology specification. Besides traditional ontology specification languages such as Ontolingua, CycL, or languages based on description logics, new languages have been developed that are based on Web standards. An example is OIL [Fensel, 2000], a Web-based representation and inference layer for ontologies, which combines the widely used modeling primitives from frame-based languages with the formal semantic and reasoning services provided by description logics.

- There is then a lot of work concerned with the actual building of domain-specific ontologies, such as the ontology developed to support the European project MK-BEEM (Multilingual Knowledge Based European Electronic Market place) described in [Gómez-Pérez, 2000] and [Corcho et al., 2003]. An overview of existing domain ontologies can be found in [Gómez-Pérez et al., 2003].

- A third important area of research deals with the complex and varied problem of enabling a system to discover an ontology. The area starts to be perceived as a domain field of its own, designated as ontology learning, and it has a strong focus on learning from text. Results presented and discussed in the Proceedings of the ECAI-2000 Workshop on Ontology Learning [Staab et al., 2000] range from tools for acquiring selectional preferences (subcategorization frames) at the right level of abstraction; tools for disambiguating word senses (results are discouraging for general NLP processing but things look promising in restricted domains); to tools for shallow linguistic processing over large data sets.

- Finally, the most philosophical trend in ontological research is concerned with the discovering and understanding of meta-ontological properties on which a principled methodology for building ontologies should be grounded: a good example is provided by the OntoClean project [Guarino and Welty, 2002].

1.2.3 From Ontology Engineering to Concept Formation

The effort put by the Ontology Engineering community in the definition of suitable KR languages (most of them with a logical flavour, despite the *caveats* advanced in [Minsky, 1990]) led to a notion of *model* as a set of propositions describing the relevant facts, and a set of causal laws describing the possible interactions with and the dynamics of the environment, where both propositions and laws are expressed in some KR language. In order to capture the ability to conceptualise as a requirement for the ability to model, it is convenient to think of intelligent autonomous systems in terms of *agents* interfaced with the world through a sensory-motor subsystem and provided with goals and motivations, and with a certain amount of reasoning and planning abilities. As we stressed along the previous Section, intelligent autonomous systems must have modelling abilities, because autonomy plus rational behaviour presuppose awareness of one's goals and motivations, and a certain amount of knowledge of one's environment and actions, in order to anticipate possible changes in the environment and the consequences of acting upon it. Such knowledge is expressed using a KR language whose lexicon refers to the relevant objects in the environment, their properties and relations, by means of being associated to some of the internal representations of the agent whose model of the environment is formulated using such language. We refer to those internal representations as the *conceptual structures* of that agent.

In [McNeill et al., 2003] we find an example of use of the notion of model described above. We discuss this work in a detailed way, to show the limitations of an approach to concept formation that abstracts from an agent's individual interaction with the world, and relies only on the communication among agents with different degrees of knowledge.

The problem addressed by the paper is the failure of a plan execution in a given domain because of an agent's incorrect knowledge about the domain. According to the authors' analysis, plan execution in a given domain often meets with failure because the domain does not conform to the agent's expectations of how a particular environment will evolve or change through time, as a result simply of the time passing by, or as a result of agents acting in the environment. These expectations depend on the agent's understanding of the domain, as described in the agent's ontology for that domain.

The notion of 'ontology' proposed by the authors corresponds to the notion of 'model' of a domain given above: it is the whole of the agent's knowledge of a domain; it could be unique to a particular agent, or shared (or partially shared) with other agents; it can be analysed into two components: the signature, and the theory. The signature is a description of the types of things that exist in the domain; for example, the listing of all the predicates together with their arity and the type of the arguments they take. The theory is a description of the instantiations of the signature objects; for example, the specific things that exist and the rules that describe what actions can take place in the domain.

For example, a signature may contain the predicate *money* as a unary predicate that applies to integers; and a theory may contain rules such as a *buy-rule* and some facts such as *at (station, me)*, *buy-at (ticket, station, 10)* etc.

In order to recover from failure of a plan execution in a given domain because of errors in the agent's ontology for that domain, the authors propose a dynamic refinement of the agent's ontology for the given domain. Such a refinement is to take place as failure of plan execution occurs. Replanning in accordance with the updated ontology should then follow.

The authors' proposal for an ontology refinement is drawn along the following lines: assume a plan implementation (PI) agent is attempting to execute a plan, for example

[buy (ticket), walk (station, embassy, amsterdam),
buy (visa), travel (amsterdam, moscow, plane)]

The general assumption is that the PI agent is actually visiting an eInstitution (agent-mediated electronic insitution), controlled by some institution agents, who facilitate the interaction between the PI agent and the other external agents that are currently visiting the same institution, in this case the interaction between the PI agent and the ticket-selling (TS) agent. The idea is that the plan that the PI agent must execute, together with the knowledge that the agent has about the domain (that is, the PI agent's ontology) are given to a plan deconstructor procedure, that produces an annotated version of the plan which contains, for each action in the plan, the rule used together with an explanation for why the preconditions of such rule are held to be true. If a failure is encountered when attempting to execute the plan, the PI agent uses the annotated plan to refine its ontology. The authors discuss examples of refinements that may occur, such as: refining the signature, for example adding extra arguments to predicates; refining the theory, for example adding preconditions to rules.

The only form of interaction that the authors take into consideration for an ontology refinement to happen is the interaction between two or more agents with a rather similar understanding of a domain. In particular, nothing is said about acquiring knowledge from the interaction with the world (rather than with another agent), and this prevents their system from dealing with failures that somehow involve the ability to recognise an object given a description of that object.

Consider, as an example, the following situation, for which we assume *is-container-for (thing1, thing2)* holds provided that the following facts also hold:

is-hollow (thing1),
volume (hollow-part (thing1)) > volume (thing2),
is-made-of (thing1, (wood or plastic or glass or metal))

Agent A and agent B are collaborating to transport some water from one room to another; A asks B for a container for the water, B brings A a colander, A complains about the fact that there are plenty of holes, hence the colander will not work in this particular case, and after some interactions B refines its ontology by learning that in the case of water or other liquids the container must not have holes. If we imagine the interactions only along the lines proposed by the authors, then, in a new situation where it is asked to partition a set of possible containers into those that can be used for water and those that cannot, B will not be able to perform the task, since it does not really know what a hole is.

Ontology refinement as a strategy to recover from plan execution failure, as described in [McNeill et al., 2003], provides a good example of the advantages of a flexible and dynamic modelling ability. To summarise the point, we can say that: if intelligence has to be robust, then it is not enough to provide the agent with a fixed model of its environment; the agent should be able to refine or even enlarge its model, to cope with unexpected situations. An important part of this ability concerns the refinement of the lexicon or the acquisition of new lexical items, in terms of which the model is formulated. The refinement or acquisition of the lexicon, however, should be accompanied by the acquisition of the concepts that constitute the meanings of such a lexicon. Sometimes it is enough to learn about new symbolic properties of the linguistic items, for example that a predicate needs a different arity, or that a rule needs extra preconditions; but in other cases what is needed is the ability to form new relations between sensory-motor data and linguistic items.

As we already stated at the beginning of this Section, intelligence is hard to characterise and is better thought of as encompassing several, relatively autonomous abilities, ranging from basic abilities such as perceptual and motor skills, to higher order ones such as modelling skills, inferencing, decision-making and planning skills, social competences, etc. In the present work we address only one phase of the modelling ability, the one which involves the acquisition of the basic representational units of a mental model, namely concepts.[5] The previous discussion of [McNeill et al., 2003] makes clear how concept formation can be achieved through communication among agents with different degrees of ontological knowledge, but also highlights that communication skills alone are sometimes insufficient. More precisely, this is the case when the referential aspect of lexical competence is involved and the agent must have knowlege of how to apply the word in the real world. Referential competence underlies performances such as naming, answering questions concerning the obtaining situation (e.g. describe what is in the room), obeying orders, following instructions, etc. Since these performances are partly based on the ability to recognise not only objects, properties and events, but also their causal relations with the motivational structure of the agent (what they can be used for, whether they might prevent a certain course of action, etc.) such performances require, on the side of the agent, the ability to categorise relevant aspects of the environment by directly acting in the environment and relating the results of one's actions to the goals and motives that initiated them. In the remainder of this work we concentrate on those aspects of the concept formation ability that require a direct interac-

[5] In section 1.3.1, pages 18-21 we provide a critique of purely reactive systems, and further support the claim that internal models are indeed needed whenever we wish our artificial agents to show higher order cognitive behaviours.

tion between the agent and its environment to learn about the features of the world that are relevant to the agent's goals.

1.3 Why concepts (and models) should be acquired

A big effort in AI has been devoted to making models and concepts available to systems from the start, by means of large knowledge-bases (see for example the Cyc project). With the emergence and establishment of the machine learning comunity, however, there has been a shift towards automatising the process of knowledge acquisition and making it autonomous. Taken to the extreme, the two approaches seem to impose a choice between, on the one hand, the attempt to build systems relying on large amounts of innate knowledge, and on the other hand the attempt to build systems able to acquire most of the knowledge they need.

In general, however, the choice between learning and innateness is not a black or white choice, and this for two reasons. First of all, it is a matter of constraints put on the system by the environment. Taking inspiration from the natural world, for example, it seems that a spider, who has no parents to learn from, should better have innate knowledge of how to make webs, because it is unlikely that it will learn this individually, by mere trial and error. The same seems true for a wildebeest: it needs to be able to run with the herd within minutes of being born, hence this ability should be innate. On the other hand, highly specific innate abilities and concepts will not suffice for an organism that needs to be able to cope with many different environments, including performing tasks that no designer could possibly foresee. In this case, what seems to be crucial is rather the possession of higher level innate knowledge, including knowledge about how to discover important features of its environment. Having such an organism in mind, an organism that could be confronted with a changing environment and with the need to perform unforeseen tasks (for example, in order to cope with anomalous situations), the advantages of learning in terms of allowing intelligence to be flexible and robust become evident.

The second reason why it is neither a matter of learning *tout court* nor innateness *tout court* has to be found in Kant's motto "Thoughts without content are empty, intuitions without concepts are blind." [Critique of Pure Reason, B 76] This means that also for a learning organism the question about innateness can and actually should be raised, both because the learning mechanism itself (or part of it) might be innate and because its actual functioning might require some further innate structures. The choice between learning and innateness becomes therefore a matter of trade-offs between having certain innate structures that enable the acquisition of new structures in a flexible and context-dependent way, and the costs and risks connected to such acquisition. In this Section we deepen these issues and provide some initial distinctions with respect to model acquisition capabilities.

1.3.1 Learning as a way to flexibility

Within computer science, model acquisition is often discussed in relation to 'reflective' systems. In [Maes, 1988] the notion of *computational reflection* is defined and a computational system is said to be *reflective* if it can reason or act upon itself. In particular, such a sytem contains a causally connected meta-system that has, as its object-system, itself, where being *causally connected* means that the system and its domain (in this case, the meta-system and the object-system) are linked in such a way that if one of the two changes, this leads to

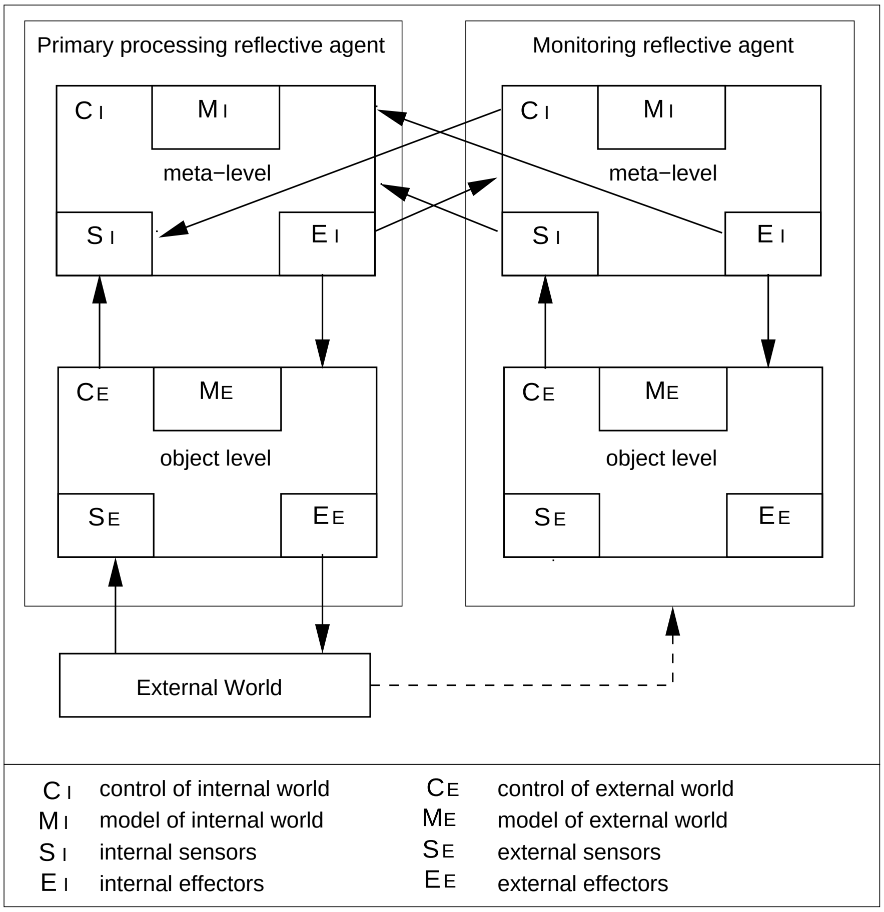


Figure 1.3: A distributed reflective architecture, as in [Kennedy and Sloman, 2002].

an effect upon the other. Most of the work on computational reflection has the purpose of giving a program access to its own implementation (consider for example 3Lisp, a Lisp interpreter written in Lisp) or providing a programming language with meta-programming facilities (consider for example reflective object-oriented languages such as CLOS or SmallTalk). Typically, the purpose is to make the internal operations of program components and the programming language features available for inspection and modification by a user, both to provide flexibility in implementation and the possibility to experiment with language extensions. In [Kennedy and Sloman, 2002] however the authors notice the relatively little work on reflective architectures for self-monitoring, where reflection would enable an autonomous agent to access its own operations for the purpose of survival and possibly self-repair in a hostile environment, without human intervention. Along these lines, Kennedy therefore investigates, in [Kennedy and Sloman, 2002] and [Kennedy, 2003], the architecture of a multi-agent system with self-monitoring abilities, where two reflective agents are mutually observing each others in order to build a model of their normal functioning. She is interested in the kinds of reflection required for fault- and intrusion-tolerance, and shows that a system with distributed reflection is able to detect problems in its own operations, as for example caused by an hostile modification of its own code. In certain cases, after detecting some form of malfunctioning, the system is able to recover without external intervention.

The distributed reflective architecture proposed by Kennedy is represented in Figure 1.3 where for each of the two agents M_E is the model of the enviroment and M_I is the model of the normal behaviour of the agent observed. Now, in Kennedy's system M_E is built-in and fixed for both agents, while M_I is acquired during a protected training phase. In order to build M_I the observing agent examines a trace of the behaviour of the agent under observation. Since agents have been implemented using Sim_Agent, a Pop11-based toolkit for agent development where agent architectures are encoded in a set of rules and then run by a rule-interpreter [Sloman and Poli, 1996, Sloman and Logan, 1999], the trace examined in order to obtain M_I is composed of patterns of rule firings.

In [Kennedy and Sloman, 2002] two main reasons are given to justify why M_I should be autonomously acquired:

- The precise content of the desirable states for self-protection is not expected to be externally specified because these states refer to internal states of the software and it is unlikely that these could be known in advance by a user, unless the software is trivial.

- We are assuming that hand-coding of "normal" patterns of software execution is unrealistic.

In other words, M_I should be acquired autonomously because usually a software design is specified in terms of what the software should be able to do, and not in terms of a detailed description of how these functionalities will be carried out in a particular working environment. Hence, the normal functioning patterns of the software in question, as coupled with a specific environment, are not supposed to be known in advance. If the system was provided with a fixed, built-in M_I, then this would result in a great lack of flexibility with regard to what the system might be able to recognise as 'normal behaviour'.

As we said, Kennedy is interested in self-monitoring, hence in her case-studies the model of the environment M_E is always fixed while the internal model M_I is acquired. However, considerations similar to those discussed above with respect to M_I can be offered with respect to the acquisition of a model of the external environment. We can say therefore that the main reasons not to provide a system with a fixed, built-in model of the external environment are:

- To allow the specification and the design of the system only in terms of functions, i.e. only in terms of what the system should be able to achieve, without going into details of how, in a specific environment, it will be achieved.

- To allow the system to be flexible, in the sense that it will be able to cope autonomously with a whole set of environmental changes, without any need of rewriting part of its code.

So, in general, agents that have *only* a fixed, built-in model of the environment will be less flexible and robust in face of environmental changes than agents who are *able to acquire* a model by interacting with the environment. On the other hand it is sometimes possible to cope with a changing environment without any need to learn models, as it is the case with purely reactive agents such as Pengi [Agre and Chapman, 1987] and certain forms of behaviour-based robotics [Matarić, 1998]. The important question therefore becomes: What are the advantages of having a model of the environment?

Reactive architectures are inspired by the idea that most human and animal behaviours consist of routine action rather than abstract reasoning (deliberating, planning, modelling, etc.). They maintain no internal models and perform no search, since their control strategy is embedded into a collection of preprogrammed condition-action pairs with minimal internal state. They have been proposed in the literature as a possible solution to the problems faced by planner-based approaches whenever uncertainty in sensing and action, sudden changes in the environment, and requirements of real-time reaction make frequent replanning inefficient, if not impossible, and usually prohibitive in terms of computational costs.

As argued in [Kirsch, 1991], purely reactive agents have proved to be superior to more traditional, representational-based ones, in some experimental settings involving simple tasks in real-world domains, but have serious problems in coping with:

- Activities which involve other agents, since these often will require making predictions of their behaviour.
- Activities which require response to events and actions beyond the creature's current sensory limits, such as taking precautions now for the future, avoiding future dangers, contingencies, idle wandering – the standard motive for internal lookahead.
- Activities which require understanding a situation from an objective perspective such as when a new recipe is followed, advice is assimilated, a strategy from one context is generalized or adapted to the current situation. All these require some measure of conceptualization.
- Activities which require some amount of problem solving, such as when we wrap a package and have to determine how many sheets of paper to use, or when we rearrange items in a refrigerator to make room for a new pot.
- Activities which are creative and hence stimulus free, such as much of language use, musical performance, mime, self-amusement.

These activities require knowledge about the world that exceeds what is usually available to egocentric perception, and must be obtained by reasoning or recall. Moreover, since they also require *global* coordination and control, they often involve the ability to anticipate the future of substantially large portions of the environment, e.g. by reasoning on a centralised model of the environment.

Behaviour-based architectures are an extension of reactive ones and fall between purely reactive and planner-based extremes [Brooks, 1986, Matarić, 1997]. In fact, even though behaviour-based systems usually contain reactive components, their computation is not limited to look-up and execution of simple functional mappings, and certain behaviours can be employed to store various forms of state and implement various types of representations (for example Toto's map behaviours [Matarić, 1997] may represent specific landmarks and their properties). In general, however, behaviour-based systems do not employ centralised representations operated on by a reasoning engine.

The tradeoffs between behaviour-based agents and representational, planner-based ones are less than those between purely reactive agents and representational-based ones, because behaviour-based systems are more flexible than purely reactive ones, they often exhibit emergent properties, and they have representational abilities. On the other hand, however, since behaviour-based systems usually employ distributed and implicit representations, we could sharpen our initial question, and ask: What are the advantages of having an *explicit* model of the environment?

In [Steels, 1995], the author explores a biologically inspired definition of intelligent autonomous agents, and when it comes to intelligence his suggestion is to consider intelligent agents to be agents that are autonomous (self-governing as well as self-steering) and able to handle representations. The ground for his suggestion is as follows:

> Representations are physical structures (for example electro-chemical states) which have correlations with aspects of the environment and thus have a predictive power for the system. These correlations are maintained by processes which are themselves quite complex and indirect, for example sensors or actuators which act as transducers of energy of one form into energy of another form. Representations support processes that in turn influence behaviour. What makes representations unique is that processes operating over representations can have their own dynamics independently of the dynamics of the world they represent.
>
> This point can be illustrated with a comparison between two control systems. Both systems have to open a valve when the temperature goes beyond a critical value. One system consists of a metal rod which expands when the temperature goes up and thereby pushes the valve open. [...] The other system consists of a temperature sensor which converts the temperature into a representation of temperature. A control process [...] decides when the valve should open and triggers a motor connected to the valve. [...]
>
> From the viewpoint of an external observer there is no difference. Differences only show up when the conditions changes. For example, when the weight of the valve increases or when the valve should remain closed under certain conditions that are different from temperature, then a new metal for the rod will have to be chosen or the system will have to be redesigned. When there is a representation, the process operating over the representation will have to change.

Steels then continues trying to solve the controversy between symbolic AI and researchers claiming that intelligence can be realised without representations, with the proposal of a misunderstanding of the extension of the notion of representation: according to Steels, symbolic AI restricts itself to *explicit* representations, and it is against this narrow notion that non-representationalists have reacted. Once implicit representations are also taken into account, the controversy is solved and the apparent paradox of a slogan such as ‘intelligence without representation’ disappears.

Steels explains that implicit or emergent representations occur when an agent has a particular behaviour which is appropriate with respect to the motivations and action patterns of other agents and the environment, but there is no model. A better definition is provided by Sloman in [Sloman, 2005], where ‘implicit’ is defined in terms of information that is encoded only in patterns of activation during its processing. Following Sloman terminology, explicit representations (and models) are therefore characterised by the fact of being enduring, and hence being accessible also when not in use to be combined with other information, compared, analysed, etc.

Now, given Steels’ initial motivations to bind the notion of intelligence to the notion of representation, we see that his successive move of extending the notion of representation in order to cover both implicit and explict representations and hence blurring the differences between the two is actually ill-placed. For in order to have representations playing the fundamental role described above, that is allowing adaptation of control under changing conditions only with changes to the processes operating over representations (and not changes of physical parts of the system or complete redesign of the control system), representations must be enduring and hence explicit. So it seems to us that the limits of behaviour-based agents will be the limits of their explicit representational abilities.

1.3.2 An example and some initial distinctions

Let's imagine an agent provided with an innate motivational structure generating some initial behaviours, for example moving around and exploring the environment. Let's also assume that in the environment different kinds of objects exist, some of which have effects on the agent when acted upon; for example, some can refill the agent with energy (let's call them Food items), others can hurt the agent (let's call them Danger items), others make the agent happy (let's call them Treasure items) and others do nothing (let's call them Neutral items). Let's also assume that our agent is usually looking for experiences of happiness and would like to avoid being hurt; however, when its energy decreases below a certain threshold, the agent feels hungry and tries to get some energy, in order to keep alive.

Given such a scenario, it is easy to show that an agent supplied with a fixed, 'well engineered' model of the environment, telling the agent what Food items, Treasure items and Danger items look like, will have no difficulty in surviving, provided there are enough Food items within reach. However, should we move our agent into a new part of the world, where for example Danger items and Food items do not look as the agent expects them to, it is also easy to imagine how our agent will be negatively affected by the environmental changes. One possible solution is to design our agent in terms of functions only, so that it will not need any model of the environment. For example, the agent could be designed in such a way as to approach any object in the environment and act upon it according to its motivation of the moment (e.g. if hungry, try to eat). Such an agent will not be able to intentionally avoid being hurt, and its experiences of happiness will be proportionate to the amount of Treasure items in the environment, but it might be able to survive for an indeterminate amount of time, also in the case of environmental changes concerning how objects look.

Another possible solution is to design our agent so that the model of the environment is acquired through interaction with the environment. For similar characteristics of the environment and the environmental changes, the main advantage of this second option is in terms of efficiency: an agent with a model of the environment can achieve its goals more efficiently, for example whenever there is a cost associated to the actions that the agent can perform. Moving has normally a cost in terms of internal energy, so it is better to approach useful objects rather then useless ones, but in order to do so it is necessary to be able to recognise useful objects (objects are useful if they can be used to satisfy the agent's needs). Also, if focusing the attention, or storing information has a cost in terms of internal energy, then it is better to focus on as few features as possible to characterise a relevant object, hence it is better to build more abstract descriptions for the interesting categories. In these examples categories are needed to make good predictions. Under certain circumstances, however, categories are needed to produce good explanations (for example if we need to provide a diagnosis for a problem we are facing, or if we want to understand a particular situation in order to avoid dangers.)

Once a model has been acquired, and it starts being used to guide action, we say that the system has become 'specialised' in the sense that it is now able to cope efficiently with a particular (specific) environment. However, if the learning that specialises the system is such that no further learning is possible, then this same system can now be negatively affected by an environmental change.

For example, we could imagine an anomalous scenario, where we introduce in the world an object which, at first sight, looks like a Food item, but which steals energy from the agent instead of providing it. If the agent has acquired a model of its environment by acting within a scenario where all Food items are 'normal', then it will not survive in the anomalous scenario, because whenever it approaches the anomalous Food item in order to eat it, it will loose its remaining energy. The normal situation can be described as something like the following: given the preconditions `low energy level` and `at Food`, taking action `eat` results in the holding of postcondition `increase of energy level`. In the anomalous situation the same action, given the same preconditions, results in the holding of postcondition `decrease of energy level`. If the agent trusts its internal sensors (its internal reading of its own energy level) then in the abnormal situation the agent should start a deeper inspection of the new object, which looks like food, but actually does not match its notion of what a normal Food item is. This means that the agent should be able to access and modify its model of the environment. Of course it cannot be guaranteed that the agent will always be able to perform the adequate modifications: for example, in this case it will depend on whether the agent actually finds some feature that distinguishes the good from the bad Food items.

This means that the choice of allowing the system to modify its acquired model of the environment is also motivated by a requirement of flexibility and robustness, that is by a requirement that the system be able to cope autonomously with possible environmental changes.

To summarise, we propose the following distinctions and the following terminology:

- a system is said to be *open* if it has no built-in model of the environment; the main advantages are that the system is flexible and that it is designed in terms of functions only (there is no need to investigate and model the enviroment); the disadvantage is that the system is potentially inefficient, because it may enter many trial and error loops;

- a system is said to be *open and able to become specialised* if it can acquire a model of the environment and then use it to guide action; the advantage is that a system that can become specialised is potentially more efficient than a simply open one, because future action can be guided by past experience (in principle, there should be less trial and error loops);

- a system is said to be *open, able to become specialised and robust* if the acquired and internalised model of the environment can be revised; the main advantages are that the system is able to adapt itself, autonomously, to further changes in the environment, and that it is robust, i.e is able to cope autonomously with change.

1.4 Summary

In this introductory Chapter we discussed, from a rather philosophical point of view, the notion of 'concept'. In Section 1.1 we distinguished a realistic approach to semantics from a cognitive one, and we motivated our choice of framing our work within the cognitive approach. The main problem faced by a realistic account of semantics — where concepts are identified with lexical meanings and characterised in terms of world states only — is that it fails to account for properties that are natural for the purposes of an agent (problem solving, planning, communicating, etc.) As an alternative, concepts can be understood as mental contents serving the representational needs that might arise when an agent acts in its world. In Section 1.1.2 we briefly addressed the 'learnability question', that is explaining how conceptual structures that are not innate are formed. In this context, we discussed Thompson's critique of correspondence theories (built upon evidences from the field of visual perception) to motivate our adoption of a pragmatic account of the epistemological relation between concepts and the world: since semantic structures (in particular, perceptual categories) do not represent objective patterns of the distal environment, but are generated to guide behaviour, the mechanisms responsible for their acquisition need to involve the interactions of the agent with its world.

In Sections 1.2 and 1.3 we argued for the role of concept formation as a requirement for intelligence from the point of view of AI. The notion of intelligence as used in AI often makes reference to the posession, implicit or explicit, of a *model* of the world by the system qualified as intelligent, where concepts are understood as basic representational units. A lot of effort has been devoted to making models and concepts available to systems from the start, by means of large knowledge-bases. However, in Section 1.2.3 we discussed the limitations of an approach to concept formation that abstracts from an agent's individual interaction with the world, and relies only on the communication among agents with different degrees of knowledge. The main problem consists in overlooking the acquisition of knowledge from the interaction with the world (rather than with another agent), thus preventing the possibility of dealing with failures that somehow involve the ability to recognise an object given a description of that object. In Section 1.3 we therefore discussed and motivated our choice to investigate agents who are *able to acquire* a model by interacting with the environment. We also gave an initial example to guide our analysis.

Chapter 2

The context and the literature review

2.1 The philosophical debate on concepts

The aim of this second Chapter is to present the different theories of concepts proposed in the philosophical, psychological and AI literature, in order to give a broad overview of the different solutions that have been proposed. Each solution should be seen as a way to cope with the constraints put on a theory of concepts by the different aspects (cognitive, semantic, epistemic, etc.) that the theory should be able to explain, and by the properties and functions that concepts are supposed to satisfy. The review does not claim to be exhaustive. The idea was to cover the major theoretical positions, highlighting their respective advantages and disadvantages in accounting for the acquisition and use of concepts.

The philosophical debate on concepts and conceptual change tends to overlap with the philosophical debate on linguistic meanings, their ontological status and their role in inference and knowledge acquisition. Before reviewing the classical philosophical view of concepts as necessary and sufficient conditions for their application, and the criticisms raised against it (e.g. by Putnam, Quine and Sellars), we briefly present, in general terms, the constraints put on a theory of concepts by the different aspects – cognitive, semantic, epistemic – that the theory should be able to explain. This will serve both as a terminological introduction and as a checklist of issues to bear in mind when evaluating each theoretical proposal to accommodate concepts. Our list is based on [Laurence and Margolis, 1999], [Thagard, 1996] and [Davidsson, 1992].

2.1.1 Constraints on a theory of concepts

To start with, a theory of concepts should account for *concept acquisition* and explain the processes by which concepts are acquired. In this respect, we distinguish:

- The acquisition of new concepts through the interaction with the environment (with or without the help of a teacher); for example, forming the concept of edible thing after some experiences of putting objects into the mouth and trying to eat them, or forming the concept of dog after having observed and interacted with few particular dogs.

- The acquisition of new concepts either through the composition of concepts previously acquired or through the modification of concepts previously acquired because of changes in the overall belief-system; for example, forming the concept of wheeled chair by putting together the concept of wheel and the concept of chair, or modifying the concept of whale after discovering or learning that a whale is not a fish but a mammal.

Among other things, a detailed account of the concept acquisition process may throw light on issues such as: whether it is possible to distinguish successive phases in the ontogeny process, thus recognising some form of pre-conceptual representations, whether certain concepts should be acknowledged as innate, as for example Kant argued, and whether there is ground for talk of conceptual change. In general, the fact that concepts seem to change over time (such as the concept of space before and after the discovery of non-euclidean geometries, but also the fact that people tend to speak of concepts as evolving, becoming clearer, becoming larger or narrower, etc.) raises the difficult question of establishing identity criteria for concepts. A related problem concerns the distinction of cases of conceptual change from cases of mere change in belief.

Another aspect that should be covered by a theory of concepts is *categorisation* and the processes supporting it. Categorisation refers to the psychological sense of the application of a concept, that is the psychological process by which an object is judged to fall under a concept, and it should be distinguished from the related notions of semantic application and reference determination. These refer to the relation between a concept and something that is in its extension: on one hand we must explain what makes an entity an instance of a particular category; on the other hand we must explain how a particular concept determines its own extension.

Besides categorisation, there are at least two other important cognitive functions for which concepts are thought to play a role and that should therefore find an explanation: *cognitive stability* and *cognitive economy*. Cognitive stability refers to the role played by concepts in allowing the comparison of the present experience with similar past experiences; while cognitive economy refers to the role played by concepts to reduce the amount of information needed to process each individual experience of a concept instance (it is in this sense that concepts are sometimes referred to as abstract representations). From a more philosophical perspective, on the other hand, it is *epistemic justification* that calls for details: here, what is required is explaining when someone is justified in taking an item to fall under a given concept.

Last but not least, since concepts provide semantic-grounding for linguistic entities, their *semantic functions* should also be accounted for. Besides semantic application and reference determination, already mentioned above, other more specific issues concern: analytic entailment, that is, explaining the semantic phenomenon of inferences based on meaning (analytic inferences); synonymy and other semantic-based relations; and also the ability to ground some inferential processes such as inferences from perceptual to non-perceptual information.

To summarise, among the most crucial aspects that a theory of concepts should account for are: the acquisition process, categorisation, cognitive stability and economy, epistemic justification, and semantic functions such as semantic application, reference determination, analytic entailment, etc. We now bring our attention to some of the philosophical, psychological and AI proposals to capture the nature of concepts, and will try, for each of the above mentioned aspects, to evaluate the advantages and disadvantages of the proposed theories.

2.1.2 Classical and neo-classical views

One of the earliest theories of concepts in philosophy, if not the first one, is to be found in Plato's Euthyphro, and since then it has been embraced by many philosophers (among others Locke, Carnap and Rey) and linguists (as Jackendoff and Katz, for example). We refer to it as 'the classical theory of concepts'. According to such theory, concepts have a definitional structure, and they encode necessary and sufficient conditions for their own application. An example is the analysis of the concept BACHELOR as UNMARRIED, ADULT, MALE. In the empiricist tradition, a further constraint is put on the defining terms: they should be sensory concepts. Three examples, the first to be found in [Locke, 1975], the second in [Carnap, 1959], the third in [Katz, 1972], are the following:

- GOLD as YELLOWNESS, GREAT WEIGHT, DUCTILITY, FUSIBILITY, SOLUBILITY IN AQUA REGIA
- ARTHROPODS as ANIMALS, WITH SEGMENTED BODIES, WITH JOINTED LEGS
- CHAIR as OBJECT, PHYSICAL, NON-LIVING, ARTIFACT, FURNITURE, PORTABLE, SOMETHING WITH LEGS, SOMETHING WITH A BACK, SOMETHING WITH A SEAT, SEAT FOR ONE

The theory has clear advantages in terms of the explanations it provides for some of the cognitive aspects listed in the previous Section. The ontogeny model and the categorisation model offered by the classical theory are quite straightforward: since a concept is just a short hand for a collection of sensory features, the acquisition of a concept is done by assembling its features, while in order to judge whether a given instance is or is not an example of the concept into question, one must run backward the ontogeny process, and check whether the features that compose the concept are actually displayed by the instance in question. The categorisation process, as explained with the classical theory, also provides the lines to account for the determination of reference: in order to check whether an item in the environment is a proper referent for a given concept, one just checks whether the item satisfies the concept's definition (and if this is the case, then epistemic justification would follow). As far as analytic entailment is concerned, the classical theory provides an account of it in terms of conceptual containment, in agreement with the intuition that an analytic statement is so, if its truth is necessitated by the meaning of its constituent terms.[1]

In [Walton, 1973], the definitional model of concepts has been used to provide an argument in support of the Whorfian hypothesis [Whorf, 1956] that different languages somehow reflect different *conceptual schemes* (different ways of experiencing, perceiving, or thinking about the world). Generally speaking, a difference of concepts is not what makes a difference of conceptual scheme: people who lack certain concepts because their environment is different from ours or because they are physiologically unable to perceive certain features of their surroundings do not thereby have a different conceptual scheme; what we would say in such case is that the difference is in *what* is experienced, rather than in *how* it is experienced. 'What is experienced' here refers to the mere sensory input. So, for example, a colour-blind

[1] Kant defines analyticity in the following terms: "Either the predicate B belongs to the subject A, as something which is covertly contained in this concept A; or B lies outside the concept A, although it does indeed stand in connection with it. In the one case I entitle the judgement analytic, in the other synthetic." See [Kant, 1965] page 48.

person and someone who has normal, not impaired sight, would have at their disposal different sensor data and therefore different concepts. But it is not this difference that matters here. We therefore restrict the notion of difference of conceptual schemes to cases in which people have corresponding data, but ordered in different ways. An example of a statement describing a conceptual scheme is the Kantian statement that human beings intuit the objects of experience in the forms of space and time. To understand such a statement as a statement about our conceptual scheme (rather than a statement about the world) we need some understanding of what it would be to intuit the objects of experience in forms other than space and time.

In order to capture more formally the notion of two agents having different conceptual schemes, Walton exploits the notion of a property being *apparent* to an agent, and the distinction between rule-based identification and non-rule-based identification of an object.

A property of an object is said to be *apparent* for an agent if and only if the agent can observe that the object has that property or the agent knows that the object has that property, while the distinction between rule-based and non-rule-based identification is captured in the following terms: "If we are familiar enough with swallows, for example, we may recognise them without even noticing their features. We may identify a swallow [...] in very much the way we tell whether something has a salty taste, without making an inference on the basis of its features." The idea is that, even though it is the swallow's features that enable us to recognise it, we do so without explicitly using a rule for determining which birds are swallows, and/or without realising which of the disjunct preconditions of such a rule applies to a given swallow.

Given the above clarifications, Walton proposes that two agents have *different conceptual schemes* if and only if

- agent1 gives priority to non-rule-applying tests of whether something is P for at least one predicate P; P is extensionally equivalent to a (non trivial) disjunction $Q1$ *or* $\ldots$ *or* Qn; and agent1's ability to apply P is dependent on some of the properties corresponding to $Q1 \ldots Qn$ being apparent to it;

- agent2 has no predicate P' extensionally equivalent to P such that agent2 gives priority to non-rule-applying tests of whether something is P'; sometimes agent2 explicitly recognises and describes each of the properties corresponding to $Q1 \ldots Qn$, that is some of these properties being apparent to agent2 is sometimes necessary for agent2 being able to apply some predicate without using rules.

As an example, consider society A, with language L^A "which incorporates a different way of classifying birds from ours. L^A contains species names, like our 'swallow', 'sandpiper', 'finch', etc., but with different ranges of application. For instance, there is a species name, 'ϕ', in L^A which applies to some but not all of the birds we call 'swallows' and some but not all of the birds we call 'swifts'. Also, A's use relational predicates which function like our "same species" and "different species", but which are of course not true of the same pairs of birds." If we assume that we can construct complicated English predicates extensionally equivalent to A's species names, then the crucial difference would be that these English predicates contain other predicates (e.g. 'has a brown body', 'flies erratically') whereas A's species names do not. "We might say that English and L^A each treats its own species as fundamental categories and the foreign species as derivative ones. I shall argue that we can expect a corresponding difference in what categories are fundamental to our, and to

A's, thought and perception, and that this difference qualifies as a difference of conceptual scheme."

The crucial difference between us and A's is a difference in which classes of objects are "natural" (rather than "arbitrary") for us and which are "natural" for them, i.e., a difference in what we and they recognise or regard as "simple" properties. Walton has in mind a society relative notion of simple property and natural class: in a society of people who identify species members sometimes by inferring from their features with the help of a disjunctive rule, and sometimes just by looking at them, we should say that a class is natural or not depending on which method of identification is considered to have priority. If people would accept the result of non-rule-applying identifications, should the two tests conflict, the property will count as a simple property for them.

Now the following distinctions are possible: different classifications which simply reflect the way things are – as groupings of objects according to taste seem to – merely concern different sorts of facts. Different classifications which are straightforwardly arbitrary, mere linguistic devices to facilitate the exchange of information about facts, just constitute different ways of talking about the facts. In either case no difference of conceptual scheme is involved. But when people stop to consider the features which *sometimes* serve as criteria for classifications (for example, in cases such as species classifications), they can see that there is an element of convention in them, and that different systems of classification would fit the data equally well, even though in the normal course of events people lose sight of these criteria. Species classifications are thus construed as concepts by means of which facts are experienced; they become part of the mechanism of thought and perception, and if they are different they can be said to embody alternative ways of ordering the data of experience.

We have discussed this particular use of the classical theory because it provides an example of how the structure of concepts can be modelled in terms of differences between implicit and explicit use of information. In Chapters 4 and 5 we will use a similar implicit *versus* explicit distinction to account for successive steps in the acquisition process of affordance concepts.

2.1.3 Criticisms on the classical view

Problems with the notion of *analyticity*

Despites the advantages discussed above, the classical view has received also severe criticisms. The first problem, already acknowledged by Plato, is the difficulty to provide definitions even for the most early acquired concepts or those that are most used across a population. Two famous examples are Wittgenstein's attempt to define the concept of GAME [Wittgenstein, 1978] and Fodor's discussion of the verb PAINT. In [Fodor, 1981] the author tries to show that the concept of PAINTING cannot be defined even using, as a constituent, the concept of PAINT. The first definition of painting to be considered is x COVERS y WITH PAINT but if a paint factory explodes and covers all the surrounding with paint, this should not count as an instance of PAINTING. So Fodor proposes: x IS AN AGENT AND x COVERS THE SURFACE OF y WITH PAINT. But again this would imply that I am painting my shoes if I accidentally kick over a bucket of paint and thereby cover my shoes with paint. So the next proposal is: x IS AN AGENT AND x INTENTIONALLY COVERS THE SURFACE OF y WITH PAINT AND x PRIMARY INTENTION IN THIS INSTANCE IS TO COVER y with paint. But then Fodor notes that even this definition is problematic because according to it, when a painter dips her paintbrush in the paint, she would be painting the tip of her paintbrush, which we would not like to recognise as being the case. Even though the oddness of this last counterexample may be due to pragmatic factors rather than semantic ones, still Fodor's analysis makes clear the difficult in providing "good" definitions.

If the empiricist constraint is to be satisfied, then a further difficulty is given by the fact that often the defining concepts are not closer to the sensory level, than the concept to be defined. Consider for example Locke's analysis of LIE in terms of "putting certain signs together by affirmation or negation, otherwise than the Ideas they stand for". To summarise, it is difficult to provide definitions of concepts; even when a definition is proposed, it is controversial whether the defining components bring us any closer to the level of sensory concepts, than the concepts under analysis; and even when the empiricist constraint is put aside, proposed definitions are most of the time controversial, because they never seem immune to counterexamples, as it is shown by the analysis of KNOWLEDGE as JUSTIFIED TRUE BELIEF and its challenge through the Gettier problem.[2]

Further criticisms to the classical view have been directed to the accounts it provides for analyticity and constitute a more general attack of logical positivism. Logical positivism, as for example exemplified by the position of Carnap [Carnap, 1956], had put analyticity at the very center of its philosophy, to account for the distinction between (meaningless) pseudo-propositions and genuine (meaningful) ones. For logical positivists, genuine propositions are those that are verifiable. The meaning of such propositions is identified with their verification conditions, and verification depends upon analyticity because analyticity acts as a bridge between those propositions that are removed from experience and those reporting observable conditions. However, since analyticity itself is not directly observable, positivists took it to be a form of tautology fixed by the conventions of a language, hence possible of being known a priori (more precisely, by means of linguistic analysis).

Both Quine and Putnam attacked logical positivism. In [Quine, 1963a] it is argued that confirmation is inherently holistic, and is based also on considerations of simplicity and overall coherence of a theory, so that the notion of 'verification conditions' becomes inadequate. Moreover, Quine maintains that any scientific theory makes use of auxiliary hypotheses, that can be adjusted, so that no statement has a set of verification conditions that can be established a priori. In [Putnam, 1962] the author claims that no statement is immune to revision, because in the context of unexpected theoretical developments even mathematical principles might be revised. For example the analysis of STRAIGHT LINE as SHORTEST DISTANCE BETWEEN 2 POINTS should be revised in the context of non-Euclidean geometries.

A possible reply to these attacks is the recognition that analyticity does not need to carry an epistemological burden, and might be taken to mean true in virtue of meaning alone. Rey for example in [Rey, 1983] highlights the fact that, upon hearing a Gettier example, most people appreciate its force, and he argues that it is so because our intuitions about analyticity concern judgements about the constitutive conditions for satisfying a concept, where these conditions are taken to be constitutive relations among concepts.

[2]See [Gettier, 1963]. As a Gettier example, consider the following: you are watching on television the final of the rugby world cup, and the French team is winning against the English one, so you are justified in believing that you have just seen the French winning and you infer that the French team is this year's world champion. However, what actually happened is that the cameras at the rugby field were not working and the television was showing a recording of last year's match. But while the TV does so, the French team is really beating the English one. So your belief about the French team being this year's world champion is true and justified, but we would hardly admit that you know it.

Problems with the notion of *reference*

A further criticism addresses the implicit assumption of the classical theory that being competent with respect to a concept is having at one's disposal a description that picks out the concept's referent. In [Putnam, 1970] the author points out that we can be mistaken or ignorant about some crucial properties of concepts even though we are able to use those concepts.

The problem is raised with a particular concern for theoretical terms such as physical magnitude terms (e.g. temperature, electrical charge, etc. – a physical magnitude or quantity is something capable of more and less, and capable of location). The questions that are asked are: what is the meaning of these terms, when they are regarded as trans-theoretical, and how can a concept refer if it is not the case that concepts provide necessary and sufficient conditions for class membership.

Putnam contrasts 'realist' theories of meaning with 'idealist' theories of meaning. In idealist theories of meaning, the meaning of such a sentence as 'electrons exist' is a function of certain predictions (preferably predictions about sensations) that can be derived from it, and these predictions are a function of the theory in which the sentence occurs; thus the sentence, and its components, have different meanings in different theories. The question of reference is hard for the idealist, since he views scientific theories and concepts as instruments for predicting sensations, and not as representatives of real things and magnitudes.

In realist theories of meaning, a concept may contain elements which are not true of the things which correspond to that concept. For example, a scientific characterisation of fish would include such properties as life under water and breathing through gills; yet lungfish and other anomalous species which lack these properties are classified as fish for scientific purposes. The concept FISH therefore does not provide anything like analytically necessary and sufficient conditions for membership in a natural kind. The fact that the concept FISH is not exactly correct, however, does not mean that the concept does not correspond to the natural kind Fish. The idea is that a concept is continually changing as a result of the impact of scientific discoveries, but that does not mean that it ceases to correspond to the same natural kind (which is itself also changing.) This implies that concepts in different theories may refer to the same things, and hence there can be successive scientific theories about the same things (this allows us to say for example that present electron theory is a better theory of the same particles that Bohr was referring to.) So, it is for realist theories that the problem of reference arises.

On a traditional view, any linguistic term has an intension and an extension. Knowing the meaning is having knowledge of the intension, for example in the case of 'red' it would be something like knowing how to verify sentences of the form 'x is red'. According to Putnam's theory linguistic competence and understanding are not just a matter of knowledge. To have linguistic competence in connection with a term one must, in addition to the usual linguistic knowledge and skills, be in the right sort of relationship to certain distinguished situations, normally situations in which the referent of the term is present.

Scientists with different theories of electricity could all use the term 'electricity' without there being a discernible intension (necessary and sufficient conditions for class membership) that they all share; however they have something in common, namely being connected by a certain kind of causal chain to a situation in which a description of electricity is given, and generally a causal description – that is, one which singles out electricity as the physical magnitude responsible for certain effects. Such a situation is called an *introducing event*. The account stresses causal descriptions because physical magnitudes are invariably discovered through their effects, and so the natural way to first single out a physical magnitude is as the magnitude responsible for certain effects. Once the term ‘electricity’ has been introduced into someone’s vocabulary, whether by an introducing event, or by his learning the word from someone who learned it via an introducing event, the referent in that person’s idiolect is also fixed. And once the referent is fixed, one can use the word to formulate any number of theories about that referent.

According to Putnam his theory applies to physical magnitude terms, proper names and natural kind words (and these are opposed to words like ‘bachelor’ which are linguistically associated with a necessary and sufficient condition for class membership, namely male and unmarried): what is interesting here is that a distinction is drawn among nouns, which seems to reflect two distinct cognitive abilities in forming classes of things, where one ability is world-driven and probably constitutes a primary step toward the formation of language, while the other is language-driven and constitutes one of the many mechanisms of language transformation and enrichment.

Kripke also raises some problems concerning reference when concepts are taken to be shorthands for collections of features in [Kripke, 1972]. He points out that most of the time the best candidates for crucial properties do not account for modal facts. The conditions for the application of a concept should determine what the concept applies to, given the actual circumstances and the possible ones: gold is still gold, even if it lacks its characteristic colour. The overall conclusion is that classical definitions do not mediate reference determination. The alternative suggested by Kripke and Putnam is that reference is fixed when a term is first introduced and causal-historical relations to kinds can be established.

Quine’s position [Quine, 1963b] is more radical and denies the possibility of reference tout-court. According to Quine, a person’s conceptual structure consists of a language and the (set theoretic) union of the theories of that person. A person’s theory on a given subject can be conceived as the class of all those sentences believed to be true by that person, and when one asserts or believes a sentence, s/he also posits or is ontologically committed to the entities singled out by the terms (singular or general, concrete or abstract) used in that sentence, by means of the individuation and quantification devices of the language.

The relation of language to theory is characterised by a mutual dependence of the semantical component of the language and the theory itself. Roughly speaking, language is constituted of syntax and semantics, where the semantical element assigns an interpretation to those structures generated by the syntactic element. According to Quine the meaning assigned to a given sentence in a theory is a function of the semantics, and semantics itself is a function of widely shared community belief. This implies that conceptual structures, as meaning structures, cannot be differentiated by looking at changes in reference, since reference is relative to language.

The relativity of reference is captured through an analogy between a language and a coordinate system, and can be expressed in three theses:

- questions of reference have no meaning except relative to some background language, just as questions concerning position and velocity have no sense except relative to a coordinate system;

- given the totality of sentences relative to a language, there is no sense in asking what the reference of any term really is, except relative to another background language, just as, given the position of a body relative to a coordinate system there is no sense in asking what its position really is, except relative to another coordinate system;

- there is no privileged background language, just as there is no privileged coordinate system.

In order to individuate conceptual structures, Quine proposes the following: A's conceptual structure is the same as B's conceptual structure if and only if the public circumstances of A's affirmations and denials agree with B's. And this is all that can legitimately be meant by difference and sameness of conceptual structure. More precisely, if a person A speaks a foreign language, although one may say that A's theory relative to translation manual M commits A to such and such an ontology, this is never evidence that A believes that entities of a certain sort exist. The thesis of the indeterminacy and inscrutability of reference thus amounts to saying that any particular statement to the effect that the objects of a theory T are such-and-such relative to a background language, fails to convey what the objects of the theory are.

A different way out to the problem of reference is offered by the proposal to substitute complete definitions with partial definitions (definitions that are necessary only, not sufficient). In this case however the problem is that of finding adequate completers for partial definitions: assume that animate is given as a necessary condition for dog, it follows that something can be animate without being a dog, so what makes it the case that the concept DOG applies to all and only dogs? Jackendoff's suggestion of the 3-D models in [Jackendoff, 1983] is an attempt to cope with this criticism: lexical concepts encode more than a partial definition, because reference is determined also by means of a sophisticated spatial representation. Cases of erroneous categorisation, however, cannot be distinguished from cases of veridical ones, when the spatial representation is involved: the statue taken to be a tiger, probably looked like a tiger. On the other hand, a three-legged, toothless, albino tiger is still a tiger, even though it might not look like the typical ones.

2.2 Other theories of concepts

The classical theory of concepts discussed in the previous Section inspired also the initial psychological studies on the subject, such as for example [Bruner et al., 1999]. However criticisms came also from the psychological domain when deeper investigations were carried out.

It was noticed that the definitional structure of concepts seems psychologically irrelevant, in the sense that it fails to turn up in a variety of experimental contexts. For example in [Kintsch, 1974] the effects of conceptual complexity are studied with respect to lexical concepts. Previous studies showed that reaction time for the identification of a given phoneme correlated with frequency of the word preceding the phoneme: detection was quicker after high-frequency words. Kintsch adopted the same experimental setup, but changed the manipulated variable from word frequency to definitional complexity, and found that the speed of recognition of the critical phoneme is unaffected by which of two words precedes it (where one word is definitionally more complex than the other, such as BELIEVE versus CONVINCE taken to mean 'cause to believe'). So, more complex concepts do not seem to introduce a greater processing load, implying that the definitional structure of a concept might be psychologically irrelevant. However, a different explanation could also be provided: definitions might be 'chunked', so that they function as a single processing unit, and conceptual structure might be conceived along an inferential model, rather than a containment one.

Other psychologists (see for instance [Medin, 1989]) directed the attention to the fact that often concepts are 'fuzzy', in the sense that it is not clear whether or not an example belongs to a given category. This seems to clash with the fact that the classical view on concepts incorporates a procedure for determining category membership in an unambiguous way, namely check for defining features. A further complication is connected to the so called typicality effects, i.e the fact that, for certain concepts, some instances are more typical than others. The prototype theory of concepts, discussed below, emerged to accommodate some of these psychological findings, above all those relating to the typicality effects.

2.2.1 A psychological approach: the prototype theory

The prototype theory of concepts emerged in the 1970s and was first completely outlined in [Rosch and Mervis, 1975]. According to the prototype theory, concepts are structured representations encoding the features that objects in their extensions *tend to have*. For example, the concept BIRD is taken to encode the features FLIES, SINGS, NESTS IN TREES, LAYS EGGS, ... because these features are present in typical birds, where the typicality of a particular instance is to be established through psychological tests, such as measuring of speed and accuracy of categorisation. For example, robins have been found to be typical birds, while ostriches, though birds, have been found to be less typical.

The theory presents numerous advantages. Since concepts encode only typical features (not necessary and sufficient ones), the difficulty of finding uncontroversial definitions does not affect it, and the same is true for the so-called 'ignorance problem' that is the fact that people, when characterising a concept, tend to provide a listing of irrelevant crucial features. The prototype theory also offers a good ontogeny model, for which a concept is acquired by assembling its features; this model is similar to the one offered by the classical theory, but for the fact that the acquisition process embodies a statistical procedure. It follows that the categorisation process is assimilated to similarity comparison, and it accommodates typicality effects and conceptual fuzziness. A further characterising trait is the emphasis which is put on non-demonstrative inference, so that many highly reliable though fallible inferences are preferred to a few maximally reliable ones.

Even though prototypes can accommodate many empirical data on the use of concepts, they are not immune to criticism. The first problem is given by the so-called 'prototypical primes'. This refers to the existence of typicality effects also for well-defined concepts that should lack a prototype structure [Armstrong et al., 1999]. For example, measuring of speed and accuracy of categorisation show that also the extension of EVEN NUMBER is graded, in the sense that it presents graded judgement of exemplariness (8 tends to be judged a better example of even number than 34 or 128). The immediate consequence is that typicality effects do not argue for the existence of prototypes.

Another problem is that in many cases prototypes are missing. For example, when concepts have no instances, such as THE KING OF THE UNITED STATES; when the extension is heterogeneous, such as FROG OR TABLE; or when concepts appear to be particularly complex or intricate, such as EMOTION or INTELLIGENCE. And when they are present, even though the notion of prototype might seem clear for simple, lexical concepts, it not obvious how prototypes for complex concepts should be constructed. Fuzzy sets are taken to be the standard model for composing graded categories, however they lead to counterintuitive results [Smith and Osherson, 1984]. Take for example the concept APPLE THAT IS NOT AN APPLE: it should be empty, but if we apply fuzzy set theory then we obtain that a highly representative striped apple would fall under it (in fact, assume the striped apple is judged to be an apple to the degree 0.3; it follows that the same apple is not-an-apple to the degree 0.7, hence, for the minimum rule, it would be an apple that is not an apple to the degree 0.3). Even assuming that in cases where negation is not involved, fuzzy sets would give the correct answer, a more general problem is given by the fact that fuzzy set theory somehow incorporates the assumption that the prototype of complex concepts is a function of its components' prototypes, but this seems to be false in many cases. A famous example is given by Fodor: how could the prototype for PET FISH – probably something like: small, brightly coloured, live in fish bowls, etc. – be constructed out of the prototypes of PET (furry, affectionate, tail-wagging, etc.) and FISH (gray, undomesticated, medium-sized, live in the ocean, etc.)? A different model for prototypes, based on a generalization of standard quantum mechanics, is proposed in [Gabora and Aerts, 2002], which can account for the above example, but which is not immune from the next criticism.

A further problem is given by the fact that prototypes do not determine concept extensions correctly, because they underdetermine atypical instances of concepts. Some authors (e.g. [Smith et al., 1984] and [Armstrong et al., 1999]) have therefore proposed to integrate the notion of prototype and the idea of concepts having a definitional structure, as in the classical model. Concepts are thought of as structured representations encoding a complete definition to which a prototype is associated. The constitutive properties of the instances of a concept are mentioned in the definition, while the evidential properties constitute the prototype. The prototype is the core of the identification procedures used for quick categorisation judgements, while the complete definition is accessed and used only when cognitive resources are not limited or when the fast categorisation produced an error. However, the idea to use the classical core (the complete definition) to decide whether a particular categorisation act was or was not erroneous inherits difficulties that are associated to any verificationist semantics: if the identity of a concept, let's take SMALLPOX, is given by the procedures under which one decides whether the concept is instantiated, then it would turn up that SMALLPOX in Middle Ages and SMALLPOX now refer to two different diseases.

2.2.2 Conceptual atomism

In order to avoid the problems connected with the determination of reference (problems of ignorance and error and stability problems), Fodor advanced the theory of conceptual atomism. In [Fodor, 1990b] and [Fodor, 1990a] it is suggested that lexical concepts are primitives, and hence have no structure. For example, the content of the concept BIRD is not given by its relations to such concepts as ANIMAL, WINGS etc., instead, BIRD expresses the property *bird* because there is a causal law connecting the property of being a bird with the concept BIRD. Such a causal relation is accounted for in terms of the Asymmetric Dependence Theory, which has similarities with the causal-historical theories of Kripke and Putnam, that we discussed in Section 2.1.3. Consider two lawful relations such as *cow*/COW (expressing the fact that the concept COW is causally related to cows) and *horse*/COW (holding because a horse has been taken to be a cow). The idea is that *cow*/COW is more fundamental because it holds anyway, while *horse*/COW is asymmetrically dependent upon the previous one holding. That is why the concept COW expresses the property *cow* and not the disjunctive property *cow or horse* even though it may sometimes happen to be used in connection with horses.

Conceptual atomism is interesting because it avoids problems of stability and ignorance and error. As long as one concept stands in the same mind-world relation, variations on surrounding beliefs have no effect on its content. And as long as one concept is appropriately connected with a property (that is, via a causal relation) what is believed about its referents is not relevant.

Other criticisms has been raised against conceptual atomism, for example the fact that it cannot provide any explanatory account of psychological phenomena, or of analytic data, but in [Laurence and Margolis, 1999] the authors show that, when combined with other representational resources, the theory can avoid most of the criticisms. A serious problem, however, remains: the fact that conceptual atomism seems inevitably to lead to a radical nativist position. Even though complex concepts can be explained as the result of assembling primitives ones, all primitive concepts having no structure at all must be taken to be innate; and this has to be the case also for concepts such as GALAXY or AIRPLANE, which is highly counterintuitive.

A way out from radical nativism is the notion of 'sustaining mechanisms' introduced in [Margolis, 1999]. Margolis adheres to conceptual atomism and believes that "what makes a concept the very concept that it is, is not how it is related to certain other concepts but how it is related to the world". In order to avoid radical nativism, concept acquisition within conceptual atomism has to be addressed, and it is necessary to explain how mind-world dependencies come to obtain. Margolis proposes the notion of *sustaining mechanism* as a mechanism supporting a mind-world dependency relation, the idea being that, even though the beliefs and inferential dispositions associated to one particular concept do not individuate the concept itself, still they are relevant to account for the acquisition of the concept:

> [I]n some cases the sustaining mechanism is going to be non-cognitive. This may be true in lower-level perceptual processing, where psychophysical laws are generally expected to have a biological source. But what about the concept TABLE or the concept WALKING or, worse, the concept PROTON? It is hard to see how biology alone could account for the lawful dependence of PROTON-tokenings on protons, that there is a psychophysical law, as it were, connecting PROTON with protons. [...] The idea, in brief, is to allow that much of what one believes, including the scientific theories one accepts, may be implicated in the mechanism that links, for example, one's PROTON-thoughts with protons. The beliefs a person has endow her with specific inferential dispositions. If someone believes current physical theories and has a sufficient understanding of them, she will have the disposition to infer that a proton is present in certain carefully designed experimental situations where the evidence, in light of what she knows, points to the presence of a proton.

Margolis distinguishes among three kinds of sustaining mechanisms:

- theory-based sustaining mechanisms are exemplified by the situation described in the previous quotation; they cover the case of a person already having at her disposal a scientific theory of a kind which causes her to be in a particular state of mind where certain mind-world relations hold (e.g. protons causing to token the concept PROTON);
- deference-based sustaining mechanisms are inspired by Putnam's proposal that reference depends upon a division of linguistic work (see [Putnam, 1975]) and cover the case of a person having little knowledge about a kind but who is able to rely on and exploit other people's more detailed knowledge;

- syndrome-based sustaining mechanisms cover the case where "someone, while ignorant of the nature of a kind, nonetheless knows enough contingent information about the kind to reliably discriminate members from non-members without relying upon anyone else's assistance".

Syndrome-based sustaining mechanisms are particularly suited to account for the acquisition of concepts of natural kinds. Take for example the concept DOG: when the person tokening the concept is not a dog-expert, the knowledge that comes into question for categorisation purposes is a collection of salient properties that are readily open to inspection and reliably indicate that something is a dog — Margolis calls it *the dog-syndrome*. Even though the dog-syndrome is reliable, there are cases where something may look tremendously like a dog also without being one (a particularly well manufactured dog-toy, for example): this amounts to saying that natural kinds are subject to a robust appearance/reality distinction. One way of accounting for such a distinction is to assume that the kind-syndrome is not constitutive of the kind; rather, category membership is determined to a large extent by some hidden properties that are responsible for the appearance of the syndrome symptoms. If such an assumption is made, then there is a disposition to look past the appearance of an exemplar in categorisation judgements: were we to find put more information about a particular exemplar — that it lacked the essential property of which the dog-syndrome is a reliable effect — we would step back and cease to apply the concept DOG. To summarise, syndrome-base sustaining mechanisms require three components:

- knowledge of the syndrome for the kind;
- belief that membership within the kind is determined by possession of an essential property;
- belief that the essential property is a reliable cause of the kind syndrome.

2.3 Concept formation in AI

In his chapter on machine learning (ML) in [Winston, 1993] the author characterises learning as the task of "coupling new information to previously acquired knowledge; and digging useful regularities out of data."

In [Mitchell, 1997] ML is described as dealing with the task of constructing computer programs that automatically improve with experience. "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at task T as measured by P improves with experience E."

Along similar terms, Langley defines ML as "the improvement of performance in some environment through the acquisition of knowledge resulting from experience in that environment" in [Langley, 1996].

These three definitions suggest that a learning system is a system able to manipulate *knowledge*, and that a learning problem is well-defined only when the *task*, the *performance measure* and the *training experience* are made explicit.

More precisely, manipulation of knowledge suggests some type of internal data structure and makes clear the fact that learning cannot be described in isolation, in the sense that a learner is always linked to some knowledge base from which it can draw, and in which it can store its acquired knowledge. This suggests that a proper learning behaviour must involve

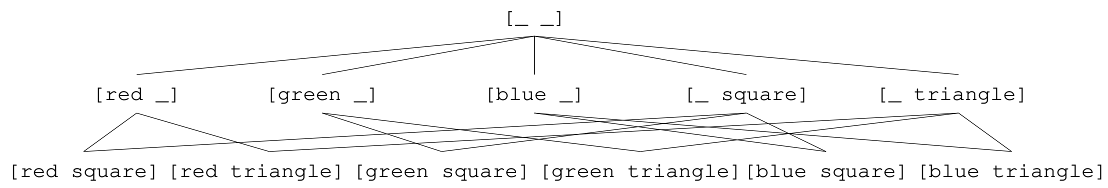


Figure 2.1: An example of hypotheses space where concepts are sets of attribute values.

an induction step, where some form of going beyond the training experiences to anticipate some features of novel test cases drawn from the same environment is present.

Performance and improvement of performance suggest that learning cannot occur in the absence of some performance task and some quantitative measure of behaviour on such task, so that it is possible to check whether, in time, some desirable change in the measure of performance has occurred. Of course many aspects of behaviour can be taken into account and measured, from accuracy to efficiency or even understanding, and sometimes the requirement of 'improvement' can be loosened more simply to 'change' since learning in certain situations may lead to decrements in performance.

Last but not least, experience suggests some form of mental processing and opens the door to all the problems connected with representational issues.

2.3.1 Learning as search

One useful perspective on ML is that it involves searching a very large space of possible hypotheses to determine either one that best fits the observed data and any prior knowledge held by the learner, or one that best performs on novel data.

In a simple classification task, the learning system is presented with a set of training examples that can be either positive or negative instances of a certain concept. Its task is to hypothesize or estimate the target concept. Depending on the representational formalism used to represents concepts, a space of possible, hypothetical concepts is defined. As an example, figure 2.1 shows the space of possible concepts if we choose to represent a concept as a set of attribute values, in a domain where only two attributes are present — *colour* and *shape* — varying respectively over the values `red`, `blue` or `green` and `square` or `triangle`.

Different kinds of hypothesis spaces (and representations) are appropriate for learning different kinds of target concepts. For each of these spaces, the corresponding learning algorithms can take advantage of a different underlying structure to organise the search. For example, in the space depicted in Figure 2.1 a learning algorithm can exploit the natural general-to-specific ordering of the possible concepts to search for a good hypotheses.

Of course, not all learning by machines or animals is concerned with finding hypotheses: in some cases, there is no prior means of formulating any hypothesis, so that learning involves acquiring new forms of representation or new languages, or even developing new skills. However, most of the learning researched within ML can be expressed in generalisations, so in our brief review we will restrict ourselves to this type of learning. More precisely, in Section 2.3.5 we will present and discuss different choices of representational formalisms for concepts that have been proposed and used in the symbolic approach to concept formation: conjunctive concepts, instance-based concepts (as an example of competitive concepts), and disjunctive concepts.

Besides the symbolic methods, that represent what is learned as (usually propositional) expressions over the given attributes, connectionist methods are also available (for a recent overview, see [Mira and Prieto, 2001]), where learning consists of adjusting weights in a (usually given) network. Focusing on two rather paradigmatic examples of each of these approaches, namely decision-tree algorithms and back-propagation, [Quinlan, 1994] compares the two approaches and characterises some classes of problems that might be expected to be well- or ill-suited to either approach. Quinlan's conjecture is that decision-tree methods are unsuitable for parallel tasks, and that gradient-descent training of networks on sequential tasks is likely to be slow.

Decision-tree methods and back-propagation, though quite different, are both instances of inductive learning. Explanation-based learning (EBL) is similar insofar as, once something has been explained, it is often possible to abstract away from some of the details and induce a more general rule that can be used for explaining or predicting future situations [Ellman, 1989]. EBL systems, after analysing a single training example, are able to produce a valid generalisation along with a deductive justification of the generalisation in terms of the system knowledge. The inputs to EBL systems are: a training example, a domain theory, a set of criteria specifying the form in which the generalised concept definition may be expressed, and a goal concept, that is a high level description of the concept to be learned. Given the above, EBL creates a generalisation of the training example which can adequately describe the goal concept and also satisfies the operationality criteria. The method first constructs an explanation by analysing the training example showing how the example satisfies the definition of the concept. This explanation is then used to formulate the definition of the general concept. An important feature of EBL methods is that they are able to justify the generalised concept. In fact, the lack of justification is a recognised problem of similarity-based learning systems.

Related to EBL, though not focusing on the explanatory step, is case-based learning. Case-based learning (CBL) algorithms are case-based reasoning systems employed in learning tasks [Aha, 1991]. Case-based reasoning refers to a style of designing a system so that reasoning in a given situation is guided by analogy to prior similar cases. CBL algorithms input a sequence of training cases and output a concept description, which can be used to generate predictions of feature values for future cases. Usually, they feature a pre-processing step which prepares the input for further processing (e.g. by normalising the range of numeric-valued features); a similarity function which assesses the similarities of a given case with the previously stored cases in the concept description; a prediction function which generates the prediction for the value of the given case features; and a memory updating step which updates the stored case-base, such as by modifying or abstracting previously stored cases, forgetting cases presumed to be noisy, or updating a feature's relevance weight setting.

In principle, all of these techniques could be relevant to the purposes of this thesis, however for practical reasons, given our logico-philosophical background, we decided to adopt a symbolic approach. Since we were not interested, at least at the beginning, in having an explanatory step, we didn't consider EBL. And given the initial simplicity of our implemented system (see Chapter 5 and Appendices A and B) we opted for a simpler learning algorithm than those featured within the CBL approach.

2.3.2 A framework for analysing a learning system

The main components of a learning system include: the environment (performance task and performance measure), the formalisms used to represent the training experience and the acquired knowledge, and the actual learning mechanism (searching strategies, biases, etc.) Each component can vary along certain dimensions which can be kept in mind as a framework to analyse and compare different learning systems. In the following paragraphs we provide a list of such dimensions based on [Langley, 1996]. The *environment* can vary along the following aspects:

- The performance task: one step classification and prediction *vs* multi-step inference.

  Given a set of training instances, *one step classification and prediction* is the task of inducing one or more class descriptions that correctly classifies most of the training instances and make accurate predictions on novel instances; *multi-step inference* involves recognition and classification in domains that involve some form of temporal or sequential information (in such domains instances are structured and usually the structure is represented in terms of a sequence of states and possible transitions from one state to another.)

- The performance measure: accuracy of prediction *vs* quality and adequacy of classification.

  Typically, the goal of classification learners is to increase the *accuracy* of their performance; in presence of an explicit classification purpose, however, the performance of an unsupervised learning system can be evaluated also with respect to the *quality and adequacy* of the produced classification.

- The amount of supervision: supervised learning *vs* unsupervised learning.

  In *supervised learning* problems each training instance includes an attribute that specifies the class it belongs to; the goal is to induce a concept description that accurately predicts this attribute.

  In *unsupervised learning* problems, where no attribute specifies the class membership of training instances, the goal is usually defined in terms of an accurate prediction over the entire set of attributes, as unsupervised learning support prediction not simply of a single class label but of arbitrary information that is missing from observations.

- The regularity of the domain: the difficulty of a learning increases with the complexity of the target concept; with the presence of irrelevant features; and with the presence of noise (we can distinguish among class noise, attribute noise, and lack of consistency over time.)

The *representational choices* concern both how to represent the training experience and the knowledge acquired through learning:

- The simplest approach to representing the training experience takes instances to be sets of attributes with binary values; more sophisticated formalisms allows for nominal or numeric values; even more sophisticated ones represent instances in terms of sets of relational literals.

- Many possible choices are available also for representing the acquired knowledge (concepts): conjunctions of features with all-or-none matching; conjunctions of features with partial matching; linear combinations of features; more complex numeric combinations of features; sets of prototypes; disjunctions of conjunctive, threshold or competitive concepts, which can be organised in decision lists, inference networks or concept hierarchies (like decision trees or discrimination networks). We discuss some of these choices in Section 2.3.5.

The *learning component* can vary along the following dimensions:

- The learning bias: search biases (also called preference or inductive biases) *vs* representational biases (also called restriction or language biases.)

  Learning biases concern how to limit the search through the space of possible hypotheses. *Search biases* assume that all possible hypothesis can be represented but limit the search by examining some hypothesis earlier than other; *representational biases* restrict the space of possible hypotheses by limiting the hypothesis representation language.

- The search strategies: differences in strategies involve which are the available operators to transform the states of the search space; where to start the search from; how to organise the search; how to evaluate alternative states; when to terminate the searching process.

  Other differences involves the *search operators*, such as whether or not structural features of the search space are exploited during the search (for example the partial ordering imposed by generality on the space of possible intensional concept descriptions); how the search is organised (exhaustive methods *vs* heuristic methods); and whether the *evaluation* takes into account only the training data or uses also other factors to make a decision.

- The processing of training instances: non-incremental learning *vs* incremental learning.

  Non-incremental learning occurs when many instances are processed at once; *incremental learning* occurs when instances are processed one at a time. A more precise notion involves the number of already seen instances the method can reprocess during each learning step: an algorithm is said to be *incremental in degree k* if, when given online data, it reprocesses at most k previous cases after encountering each training instance; if it reprocesses all earlier instances it is said to be non-incremental.

2.3.3 Concept formation: clustering + characterisation

Unsupervised incremental learning of class descriptions is usually called *concept formation*. Since the learner cannot rely on externally defined categories, its first task consists in using internalised heuristics to organise its observations into categories; this is usually called *clustering*. A second task consists in determining a concept (that is, an intensional description) for each extensionally defined subset discovered by clustering; this is usually referred to as *characterisation*.

The task of clustering algorithms is to group together similar observations into the same category and separate dissimilar observations into distinct categories. Sometimes clusters are merged to form overlapping categories; methods that form overlapping clusters are called *clumping* techniques.

Hierarchical approaches form trees where each node represents a class of observations. Agglomerative strategies build the tree in a bottom-up manner, by computing the similarity between all pairs of observations, merging the two most similar into a single category, and repeating the process until one single category (the root of the tree, that is the most general category) is formed. Divisive methods also form a tree, but do so in a top-down manner, beginning with a single category containing all observations, and repeatedly subdividing it until singleton categories are obtained.

Optimisation (or partitioning) approaches search for an optimal partitioning. They do not form a tree-structured classification but simply partition the observations at a single level. They often require the user to specify the number of categories to be formed.

Once clustering (or clumping) has been performed, methods are needed to discover functions that classify new data into the clusters that have been formed. This second task, namely discover functions that intensionally represents the clusters, is the same task of supervised learning systems for concept discovery. For a more detailed discussion see [Fisher et al., 1992].

2.3.4 Extensional *vs* intensional representations for concepts

As we said, a concept learning system must represent:

- the experience that leads to learning;
- the knowledge produced.

This means that, when designing a learning system, a decision must be taken with respect to: i) how to represent the instances of some concept; ii) how to represent the concept itself.

As far as concepts are concerned, a distinction must be kept in mind between extensional and intensional representations. A concept could be represented simply by listing all observed instances of it. Such a representation is called *extensional*, and even though plausible it has some undesirable drawbacks: it can be very expensive in terms of memory, if the instance domain contains a very large or even infinite number of instances; even though it can summarise previous experience, it cannot be used to make predictions on novel instances, at least not without the addition of a module that computes some overall properties of the extension and then uses them to define a certain notion of similarity that will be used to judge the novel instance. But such a module actually could be seen as an intensional definition for that same concept.

A concept is represented *intensionally* if its definition involves formulating some description of it for use in distinguishing between positive and negative instances. The most important feature of an intensional representation for concepts is that it can be used to make predictions on novel instances. However, this approach introduces two problems: i) a decision must be taken not only on how to represent instances but also on how to represent concepts; this means that there will be a distinction between the instance description language and the concept description language; ii) a decision must be taken on some interface between the two languages; this means finding a way to use the intensional definition to classify new instances, to be incorporated in what we call an *interpreter or matcher*. An interpreter or matcher constitutes therefore a primitive component of every concept learning system; its function can be described as follows: given an instance description and one or more concept descriptions, decide to which concepts, if any, the instance belongs.

Another way of putting things is to say that an intensional representation for a concept has two parts: how to extract information from positive and/or negative instances (as we will see, different learning algorithms exploit positive and negative instances in different ways, and use different strategies to learn from mistakes); how to use the extracted information to decide upon novel instances.

In the next Section we review in detail the case where instances are represented as sets of attribute values (or features) and concepts are represented as logical conjunctions of attribute values. Values can be boolean, nominal or numeric, while the interpreter uses an all-or-none matching. The review is based on [Langley, 1996] and [Mitchell, 1997] .

2.3.5 Examples of AI approaches to concept formation

Simple conjunctive concepts

The task of inducing simple conjunctive concepts is as follows:

- *given* a set of positive and negative instances for the class C;
- *find* a description D, which is a logical conjunction, that covers the positive instances and fails to cover the negative ones; or *find* a description D, which is a logical conjunction that, to the extent possible, correctly classifies novel instances (each component of the conjunction will be referred to as a condition or test).

These two goals are different: the latter is a better goal in noisy domains because it allows for some training instances to be mismatched.

Instances are represented as sets of attribute values (a set of attribute values is also called a set of features); concepts (or class descriptions) are represented as logical conjunctions of attribute values.

Conjunctive concepts are naturally ordered according to a general-to-specific partial order. Most methods for inducing logical conjunctions exploit such ordering of the space of possible concepts to constrain and organize the search for useful concepts.

If concepts are represented as logical conjunctions then there is a direct relation between the relative generality of two classes of instances (concept extensions) and their relative concept descriptions (concept intensions): dropping a nominal condition from a concept description D (or extending a numeric range) produces a strictly more general concept; adding a nominal condition to a concept description D (or restricting a numeric range) produces a strictly more specific concept. This relation suggests some obvious operations for moving through the space of concept descriptions.

Given the partial ordering of possible concepts, the same set of training instances can give rise to different concept descriptions, namely more general or more specific ones, which will then make different predictions on novel instances. A novel instance I is classified under description D only if it matches all conditions in D; this means that a concept is applied through a process of all-or-none matching.

The learning bias is a representational bias, which means that not all possible subsets of the instance space can be represented by conjunctive concepts. The only dimension along which simple conjunctive concepts can change is the general-to-specific ordering.

Threshold concepts

In the case of threshold concepts instances are represented, as before, as sets of attribute values (or features), but concepts are represented as *threshold units*.

In general, threshold units provide a more flexible representational tool for capturing concepts because the matching process used to make predictions on novel instances is such that an instance is assigned to a class even when *only some* of the features in the class description are present. This means that threshold concepts rely on a *partial matching* interpreter.

The task is the same as before, that is, given a set of positive and negative instances for the class C, find a description D, which is a threshold unit that, to the extent possible, correctly classifies novel instances. Three forms of threshold description can be found in the literature: criteria tables, linear threshold units (LTU) and spherical threshold units (STU).

Criteria tables: A *criteria table* includes a set of n Boolean attributes and an integral threshold m ranging from 1 to n (criteria tables are often called *m-of-n concepts*.) As an example, consider the concept:

```
2 of [red, not-circle, hot]
```

for a domain where three attributes are present: colour (either red or not-red), shape (either circle or not-circle), temperature (either hot or not-hot).

Criteria tables allow for a notion of *prototype*, given by the conjunction of the n features in the *m-of-n concept*. Such a conjunction represents a point (or a region, if irrelevant features are present) in the instance space that serves as the central tendency of the concept.

The criteria table representational formalism can be extended to deal with nominal or numeric domains, either allowing for internal disjunctions of nominal attributes and ranges of numeric attributes, or treating these as Boolean features in turn. As an example, consider the concept:

```
3 of [(red), (circle or square), (cold or warm), (hard or soft)]
```

for a domain where four attributes are present: colour (red, blue or green), shape (circle or square), temperature (cold, warm, hot or very hot), texture (hard or soft).

Criteria tables can be seen from two different perspectives: as a generalisation of the notion of simple conjunctive concept discussed above (a simple conjunctive concept of n features can also be represented as an *n-of-n concept*); or as a special case of linear threshold units, that are discussed in the next paragraph (a criteria table is an LTU in which all weights are either 1 or -1). Criteria tables can change along the following dimensions:

- general-to-specific ordering (number of relevant features, that is the m parameter of the *m-of-n concept*);

- prototypicality (which are the features that determine the prototype or prototypical examples, that is the n parameter of the *m-of-n concept*.)

Linear threshold units: A *linear threshold unit* includes a weighted set of Boolean or numeric attributes and a real threshold. As an example consider the concept

$$\texttt{if height} \geqslant 2.2\,\texttt{girth} + 0.5\,\texttt{then THIN}$$

for a domain where the relevant numeric attributes are Height and Girth.

LTU can characterise any concept extension that can be separated by a single hyperplane drawn through the instance space; the weights specify the orientation of the hyperplane while the threshold gives its location along a perpendicular. In numeric domains LTUs do not produce clear prototypes, since their extension is unbounded.

LTU concepts can change only with respect to the general-to specific ordering, depending on the number of relevant features.

Spherical threshold units: *Spherical threshold units* can represent concepts whose extension is an hypersphere. They include a set of numeric attributes (specifying the relevant dimensions of the hypersphere), a set of numeric values (one for every dimensional attribute, specifying the location of the hypersphere's center along that dimension) and a real threshold (specifying the radius of the hypersphere.) As an example of a concept for the domain of Height and Girth, consider the following:

$$\texttt{if } \sqrt{(\texttt{height} - 5.6)^2 + (\texttt{girth} - 2.5)^2} \leqslant 1.5 \texttt{ then NORMAL}$$

STU concepts can change along the following dimensions:

- general-to specific ordering, depending on the number of relevant features;
- prototypicality.

Competitive concepts

Competitive concepts rely on a *best-match* interpreter: as with the threshold approach, the interpreter carries out a partial matching; however, rather than using a threshold to determine an acceptable match, it computes the degree of match for alternative concepts, and selects the best competitor. The learning of competitive concepts is therefore intrinsically a *multiclass learning task*, as competitive concepts cannot be represented in isolation.

There are two broad approaches to the induction of competitive concepts: *instance-based learning* and *probabilistic learning*. These two methods rely on two different formalisms to represent concepts: in instance-based learning concepts are represented in terms of *prototypes*; in probabilistic learning concepts are represented as *probabilistic summaries*. In what follows we review some aspects of instance-based learning.

As we already said, a classification task consists of two main subtasks: learning a concept description for a class; and then using a concept description to make predictions on novel instances. Moreover, if the training data are unlabeled (i.e. the class membership is not specified), a further subtask is required: the partitioning of the training data into cluster of 'similar' data, to be considered the extensions of the classes to be learnt.

Given a set of training instances, each with an associated class label, the simplest algorithm to learn concept descriptions in terms of prototypes is the INSTANCE-AVERAGING method. A prototype is simply a conjunction of attribute-value pairs, which means that in instance-based learning concepts are represented using the same representational language as instances. To determine the value p_i for prototype P along a numeric attribute i, INSTANCE-AVERAGING computes the following

$$p_i = \frac{1}{n}\sum_{j=1}^{n} x_{ij}$$

where x_{ij} is the value on attribute i of the jth of n instances of a given class.

Let's now turn to the induction step, that is the task of using a concept description to make predictions on novel instances. In instance-based learning this task relies on the application of a *consistency heuristic* to a set of reference cases, stored in the memory. When a new instance is encountered, of which some unknown attribute value must be guessed, the consistency heuristic finds the *most similar* case C, as measured by known attribute values, among the stored reference cases, and guesses that the new instance will have the same values as C for all attributes. This of course implies that, given how an instance is represented, a notion of similarity among instances must be defined.

If instances are represented as real-valued points in an n-dimensional Euclidean space, the notion of similarity is usually defined in terms of the standard Euclidean distance. 'Similar' therefore means spatially close. In such a domain, the most basic induction method is the K-NN (k-nearest neighbour) algorithm. Given a set of training instances, the estimate value v_i for a new instance $Inst$ along attribute i returned by K-NN is just the most common value v_i^* along attribute i among the k training examples nearest to $Inst$.

All the attributes of an instance are considered when attempting to retrieve similar cases from memory: this means that if the target concept depends on only a few of the many available attributes, then the instances that are truly more similar can be very distant. One possible solution to this is given by a DISTANCE-WEIGHTED K-NN algorithm, where each attribute is weighted differently when calculating the distance between two instances.

Disjunctive concepts

The formalisms examined so far to represent concepts (conjunctions of attribute values with an all-or-none interpreter; conjunctions of attribute values with a partial interpreter, or criteria tables; linear and spherical threshold units; prototypes with a best-match interpreter) all suffer from an intrinsic representational bias: they cannot represents concepts covering more than a single decision region in the instance space, that is concepts involving disjunctive descriptions. In what follows we discuss one possible approach to disjunctive concepts, which makes use of a special tree structure as a representational formalism for concepts.

A *decision tree* [Quinlan, 1986] for a concept C is a tree that is used to classify instances by sorting them down the tree from the root to some leaf node. Each node in the tree specifies a test of some attribute of the instance. Each branch descending from that node corresponds to one of the possible values for the tested attribute. To classify an instance, that is to determine whether or not that particular instance exemplifies the concept we are interested in, we must start at the root node of the tree, test the attribute specified by that node, then move down the tree branch corresponding to the value of the attribute of the instance.

In general, decision trees represent a disjunction of conjunctions of constraints on the attribute values of instances. Each path from the root to a leaf corresponds to a conjunction, and the tree itself to a disjunction of these conjunctions. Decision trees can also be re-represented as sets of if-then rules.

Appropriate problems for decision tree learning have the following characteristics:

- instances are represented by attribute-value pairs;
- the target concept may be disjunctive;
- the training data may contain errors (both errors in the classification of the training examples and errors in the attribute values that describe them;)

- the training data may contain missing attribute values.

2.4 Summary

In this Chapter we outlined and discussed some of the different theories of concepts proposed in the philosophical, psychological and AI literature. We presented the classical view on concepts, according to which concepts are thought of as necessary and sufficient conditions for their application, and reviewed the criticisms raised by its poor account of the notion of analyticity and reference. We then discussed two alternatives that have been proposed: the prototype theory of concepts and conceptual atomism. According to the prototype theory, concepts are structured mental representations that encode features that objects in their extension tend to possess. Despite the fact that the theory accomodates many empirical findings, it still faces important criticism, such as the missing prototypes problem, the compositionality problem and the reference problem. As a compromise proposal, some authors have suggested a hybrid approach conveying the idea that concepts might possess a classic atomic core, accessed through slow definition-based inference procedures, and a more superficial coat of 'rough and ready' perceptual and functional properties quickly accessible by some prototype-based identification procedures.

According to the view proposed by conceptual atomism, lexical concepts are primitive and lack any structure. Conceptual atomism is interesting because it avoids problems of stability and ignorance and error by postulating that concepts can be appropriately connected with a property via a causal relation. On the other hand, at least in the original version proposed by Fodor, it remained open to the criticism of radical innatism. We therefore discussed one of the most recent proposals to counter this criticism, namely the idea to explain how mind-world dependencies come to obtain in terms of sustaining mechanisms.

Finally, in Section 2.3, we went through some of the proposals to be found in the Machine Learning literature to characterise conceptual structures. We proposed a framework for analysing a learning system and saw how the concept formation process can be divided into two steps: the clustering phase and the characterisation phase. We then gave some examples of different types of concepts that can be induced, depending on the representational formalism adopted.

Instead of an exhaustive review, the idea was to cover the major theoretical positions, highlighting their respective advantages and disadvantages in accounting for the acquisition and use of concepts.

Chapter 3

Our approach to the problem: an architecture-based analysis

3.1 Methodology

In this Chapter we discuss our methodology and we present our proposal to address concept formation in terms of an architecture-based analysis, while in Chapter 4 we will examine the question of whether different kinds of concepts can actually be captured in terms of architectural differences.

As we said, we approach the study of concept formation from a *design point of view*: this means that we aim at understanding concept formation by *investigating the space of possible designs* of a system that supports the emergence of concept formation as a cognitive phenomenon. The idea is to provide an explanation of concept formation by specifying the *architecture* of a system (in our case, the architecture of an autonomous virtual agent) that could actually work and exhibit the ability to form concepts. Even though the methodology we propose is general, we focus on a particular type of concept, namely affordance concepts. These are presented in depth in the next Chapter, while in the following sections we discuss the notion of architecture and the related notions of design-space and niche-space. We then provide an overview of the research on architectures.

3.1.1 Architectures and architecture-based analyses

An *architecture* is a structure, which can be analysed in terms of component parts, their functions within the whole structure, and their relationships. It includes those aspects of a system that are constant over time and across different application domains. In order to specify an architecture, its representational assumption should be clearly stated, together with the characteristics of its memories, and the processes that operate on them. For example, in [Sloman, 1993, Sloman and Scheutz, 2002, Sloman, 2001] Sloman discusses a specification for the architecture of a (normal, adult) human mind, the H-CogAff architecture, represented in Figure 3.1. The architecture is schematic because many components are incomplete or under-specified. It is however possible to identify 3 major layers, interacting within an integrated whole. Sloman's model integrates the *triple tower* and *triple layer* models discussed in [Nilsson, 1998], allowing for more complexity both on the perception and action side: information flowing through perceptual, control, and motor subsystems is involved in tasks of varying abstractness, for which different processing mechanisms and representations are

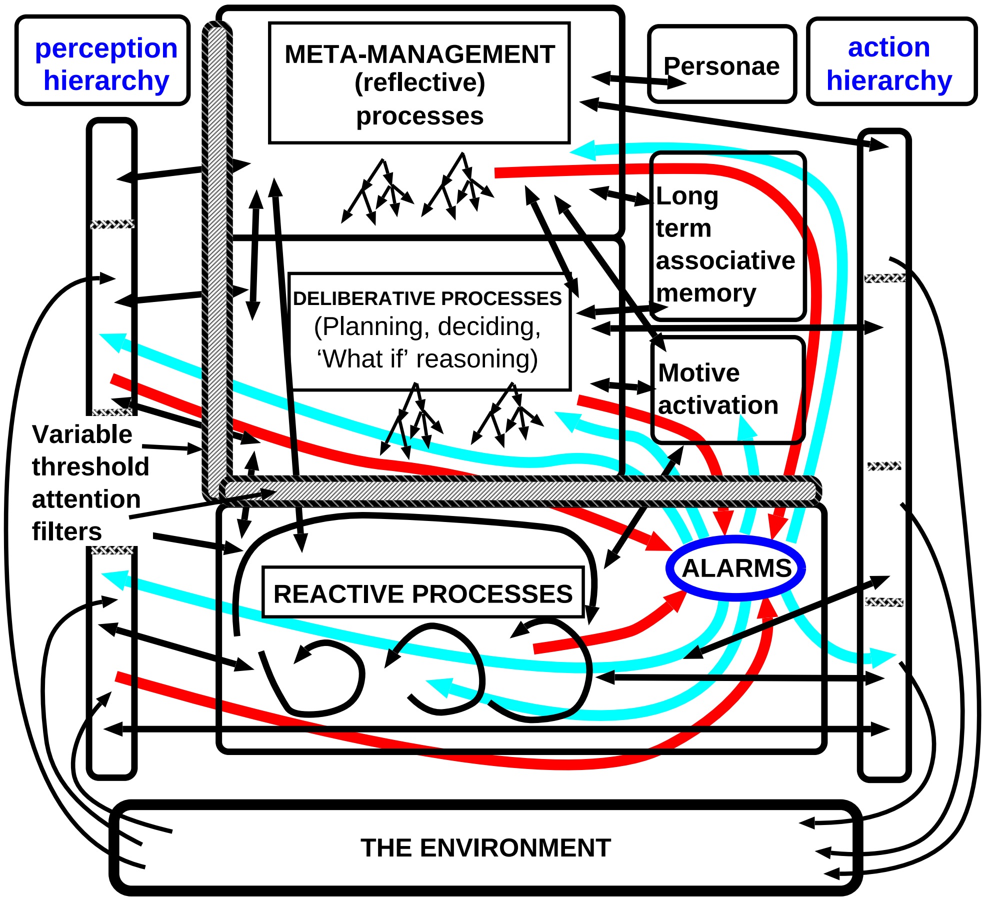


Figure 3.1: The H-CogAff architecture, as in [Sloman, 2001].

needed. The bottom layer refers to purely reactive processes, and can support both innate and learnt reflexes (an insect architecture might include only this layer). Reactive processes can have long-term memories (e.g. trainable, adaptive neural networks) and they can do reasoning (e.g. deterministic forward- chaining inferences). What makes them reactive is their inability to represent and compare hypothetical entities of varying complexity (e.g plans, explanations, predictions, hypotheses about things that are out of sight). The latter are defining features of deliberative mechanisms, involved in the second layer of the H-Cogaff architecture, and seem to require forms of representation supporting hierarchical structures and compositional semantics. The top layer monitors the overall behaviour of the agent and involves meta-management processes. The key feature of meta-management is meta-semantic capability, i.e. the ability to monitor, represent, evaluate, compare and (in some cases modulate) information-processing processes, i.e. processes whose internal states have semantic content.

The component parts of an architecture are usually identified in terms of their function, so that on the one hand we can speak of the set of functional components of one particular architecture, and on the other hand it is possible to say what kind of functionalities are supported by the architecture in question. It is this last property that is exploited in an *architecture-based analysis* of a cognitive phenomenon: the cognitive phenomenon (e.g. concept formation) is explained by providing an architecture that supports the constitutive functions of the phenomenon under investigation. An architecture-based analysis is therefore opposed to other kinds of analysis such as: 'correlation based' analyses, where we try to find out under what conditions a system produces which behaviours or features; 'semantics based' analyses, where we try to find out what people think about the functionality under investigation, and how they define it; 'descriptive' analyses, that is just describing properties of the system that don't explain how it works; or also analyses from the 'intentional stance' point of view, which assumes that the system is rational and has beliefs, desires, intentions, etc. which justify the actions, but that, again, says nothing about how the system actually works.

Examples of architecture-based analyses that take as a starting point the CogAff architecture discussed above can be found in [Beaudoin, 1994, Kennedy, 2003, Hawes, 2003]. Beaudoin elucidates goal processing in autonomous agents from a design-stance, Kennedy presents some proof-of-concept implementations of distributed reflection based on a multi-agent system, and Hawes explores an architecture design with the functionality to support an anytime planner in a dynamic, complex world. For a wider discussion of the research on architectures, a review of the state of the art is provided in Section 3.1.3.

In general, it is difficult, if not impossible, to *prove* that some bits of an architecture are *needed* to enable a certain cognitive function. Therefore, our strategy will be as follows: first of all we show that a particular architecture actually works (more precisely, we show that a particular architecture supports the autonomous learning of affordance concepts – which is not obvious at first sight), then we analyse such an architecture to get a deeper understanding of why it works, and finally we argue for the relevance of certain particular features of the proposed architecture. The novelty of this approach is discussed in [Sloman and Scheutz, 2002] where a methodological framework is proposed, within which different architectures can be compared and contrasted.

The notions of 'design space' and 'niche space' and their relationships are discussed in the next Section; here it is important to stress that, from the methodological point of view, building a system that supports the required functionality can be part of the explanatory activity for that same functionality, but it cannot exhaust it, unless it is accompanied by a detailed analysis of the architecture that the system implements. However building systems can be a pre-cursor to finding the correct design-based explanation of a particular functionality, because by building systems it is possible to learn what sorts of things are and are not capable of producing the required behaviours. This is the case even when a particular system is not yet a good model of the phenomenon we are trying to explain, because the analysis of a particular system always requires understanding a region of design space, i.e. a variety of designs, how they differ, what the implications are, etc. Often, in fact, it is only by building a system and finding out its inadequacies by experimenting with it that we find out what the required behaviours are, because it is very difficult to specify all requirements in advance except for trivial problems. E.g. people may think that an intelligent tutoring system needs to be able to do X, Y, Z, build something that actually does X, Y, Z and then find out that is not enough for a useful system, because it also needs to do other things.

Following the terminology of [Bates et al., 1991], we can therefore say that our approach in designing architectures is 'broad and shallow': the more distinct functions there are in an architecture, the *broader* it is; on the other hand, we say that a *shallow* architecture is one that specifies a system in terms of scaled-down, simplified versions of the components needed. Shallowness is a practical, unavoidable necessity in the early stages of the exploration of a design-space, above all when aiming at complete systems. However, we assume that a shallow architecture can be deepened without changing its fundamental properties, or at least can guide the development of a deeper architecture, more realistic and more complex. Often the process of deepening reveals errors that mean that the high level architecture has to be changed. But starting shallow may be a good way (if not the only one) of discovering the problems.

In the long term, a deeper validation of an architecture-based analysis can be based on the methodology of 'Research programmes' [Lakatos, 1995]. Typically, a research programme has a hard core, a protective belt of auxiliary hypotheses, and a heuristic. For example, in the Newtonian programme, the hard core is given by the laws of motion and the universal law of gravitation; auxiliary hypotheses concerning the refraction of light or the existence of a hitherto unknown planet are called for to deal with anomalies in the motion of planets; and the problem solving machinery (the heuristic) is given by classical mathematical physics. Lakatos then proposes the capacity of "predicting new facts" as a major criterion to distinguish a scientific or progressive research programme from a pseudoscientific or degenerating one. Borrowing Lakatos' terminology, we say that the aspects of architectural design that do not change in a series of simulations based on different implementations can be assimilated to the "core assumptions" characterising and identifying a theory. On the other hand, the implementational details that may vary across different simulations can be assimilated to the "peripheral assumptions", changing in response to new empirical data. The nature of the implementational changes (for example, whether they involved principled refinements or ad hoc hand-coding) therefore determines the value of any proposed architecture. Similar ideas have been advanced in [Cooper, 2002, Petters, 2004].

3.1.2 Design-space and niche-spaces

An architecture-based analysis requires the adoption of a *design-based approach* that aims at finding out the *architecture design* of a system supporting the particular functionality under investigation. A detailed explanation of the ideas grounding such an approach can be found in [Sloman, 1995, Sloman, 1998b, Sloman, 2000a, Sloman et al., 2004]. In what follows we summarise the most relevant issues and clarify them in relation to our own work.

In [Sloman, 1995] the author suggests a list of tasks for AI in terms of understanding niche-space, design-space and their relationships. More precisely, he proposes: the study of niche-space and the study of design-space, keeping an eye on the different dimensions and levels of abstraction that may occur; the study of the possible mappings bewteen the two spaces; and the study of the possible trajectories in both spaces.

The *design space* refers to the space of all possible designs, while a *niche* is a set of constraints and requirements for a set of designs (plural because many different designs can meet the same requirements and constraints, according to different criteria of optimality). The concept is inspired from biology, and it should not be confused simply with a physical location, because a particular physical location may instantiate several niches at the same time: for example, a mammal and an insect sharing the same physical location can be in very different niches, and the same is true of a squirrel and a black-bird living on the same branch of the same tree in the same forest. Sloman proposes the following ingredients, in order to specify a niche: the ontology of the environment, as sensed and acted on by the agent; the dynamics possible within that ontology (which events and processes can occur); the means of sensing the environment available to the agent; the types of actions required of the agent; and a collection of possible tasks and constraints, where the set of actual tasks may be dynamically changing.

Since requirements can vary, we can speak of a space of possible niches. And since requirements can be met, or partially met, we can say that a niche-space is related to a *subset of the design-space* that is, the space of possible system architectures satisfying the niche requirements. However, as pointed out in [Sloman, 2002], the relations between designs and niches typically cannot be captured by a simple numerical fitness function, or by a total ordering. This is so for many reasons: different designs can be coupled with different niches and vice-versa, thus requiring a trade-offs analysis of the advantages and disadvantages of each particular coupling; often, complex architectures are the result of the co-evolution of different designs and different niches that interact and change along time; moreover, there are different sorts of trajectories along design and niche spaces, such as individual development through learning, but also species development through evolution (genetically controlled) or even discontinuous development by means of hardware intervention.

An architecture-based analysis involves exploring the subset of the design space relevant to a particular 'niche'. A design is a description of how an actual system works in order to meet the demands of its niche (e.g. performing the individuated tasks, with the individuated sensing and acting capabilities, etc.) The description should be given from the standpoint of an engineer that is trying to figure out how the system should work, in order to satisfy the demands of the niche.

The notions of need and function, as outlined in [Sloman et al., 2004], are close to the notions of niche and design. According to the authors, the general notion of X having a need does not presuppose a notion of goal or purpose, but merely refers to necessary conditions for the truth of some statement about X, $P(X)$. In non trivial cases, for example, something like "X grows, reproduces, avoids damage" etc. So, somehow, needs express requirements, they are relative to whatever they are necessary for and they may also be relative to a context. The notion of function is related to that of need, in particular for those entities that we can usefully describe as having needs related to survival and reproduction and having components and states that can be said to have functions insofar as they serve those needs. The authors' proposed definition is: "Parts of a system, or states or processes in a system, can be said to have *function* in that system if their existence helps to serve the needs of the system, under some conditions. In those conditions the parts with functions are *sufficient*, or *part of a sufficient condition* for the need to be met." As we said, designs are meant to meet requirements: in this sense we can therefore say that in a design, functional elements can be individuated.

In a preliminary analysis the four interrelated notions seem to have disjoint denotations: on one hand, needs and niches expressing requirements, on the other hand designs defining functions, with no overlap. At a closer look, however, the categorisation in terms of niche/needs and design/functions is more a matter of perspective than anything else. As an example, let's consider the squirrel and the blackbird mentioned before and let's try to describe their niches. We could start by saying something like: they are both living organisms with two general tasks to accomplish, namely surviving and producing offspring. Looking more closely at the task of surviving, we could say that it is both a matter of maintaining a certain level of energy within the body, so that the body vital functions can be performed, and also a matter of preventing certain physical interactions that may destroy the body or parts of it. Let's consider the energy maintenance problem, and switch to the design point of view, that is: how can we proceed in order to solve it practically. We could for example distinguish among three strategies for mantaining a certain level of energy: one strategy would be having some sort of energy storage, something like a battery (and if, from the niche point of view, the task of surviving is limited in time – something like survivng for n years, rather than surviving indefinitely – then the battery could be large enough not to have to be changed or recharged; incidently, this kind of mechanism has been discovered also by nature: the egg providing enough food to the embryo until this last one has reached a certain maturity stage *is* some kind of battery.) Another possibility would be having someone taking care of the energy recharging, like baby animals waiting for their parents to feed them. Still another possibility would be to have the system actively seeking to recharge: from the niche point of view, this design solution could engender some new subtasks, such as looking for food, preparing it for consumption, consuming part of it, storing part of it for later consumption, etc. Refining the design in order to address these new subtasks may involve taking into consideration the fact that squirrels and blackbirds have quite different bodies that enable them with very different movements, for example the squirrel being able to run fast along a tree-branch and the black-bird being able to fly.

Let's now step back and consider the case where there is someone taking care for the energy recharging of our system: this is a design-stance description in the sense that it is a way of solving the surviving problem, but it can also be considered a niche-stance description, in the sense that it establishes a more refined task for which we can imagine different ways of getting around to achieve it. For example, if the niche is such that plenty of food is always at hand of the care-taker, we could imagine the following mechanism: when the energy is low, the system being taken care of produces some form of signal that will produce, in the care-taker, a response in terms of feeding. Again, this is a design-stance description with respect to the 'being recharged by someone' task, but it can play the role of a niche-stance description, that is, a description of what needs to be achieved, with respect to a more fine-grained detailing of how the mechanism actually works. A reactive response to the signal by the care-taker (the response is always there, no matter how the signal is produced) or the care-taker being able to decide whether or not to respond to the signal, depending on whether or not there is something more urgent to be done for its own survival are two examples. A further possibility is the care-taker being able to base its decision also upon an evaluation of the state of the system it is taking care of, so that we can speak of the care-taker as actually having some form of knowledge of the system it is taking care of.

To summarise our position, a niche-stance description for a particular system is a description in terms of requirements for that system. Usually, this is the starting point of an engineering process. Such a description asks for a design, that is the individuation of a set of functionalities that will meet the requirements. However, if the design is not too much detailed, it could still leave room for many different ways of realising each function. In this sense, any intermediate design, is at the same time a design with respect to the preceeding higher level specification and a new set of requirements with respect to the next, more detailed design. At the bottom level we find, of course, an actual implemementation.

3.1.3 An overview of the research on architectures

Even though research in AI has been dominated by concerns of algorithms efficiency, adequacy of representational languages, and development of more or less *ad hoc* solutions to specialised problems, the importance of a system-level perspective, capable of providing some form of understanding of how various components should be put together and interrelated to obtain a complete working system was already highlighted in [Newell, 1973]. More recently, the renewed interest for system-level analyses and architectures is witnessed by the organisation of the ‘Cognitive Architectures’ symposium, held at Stanford on March 2003 (the symposium materials are available online at `http://www.isle.org/symposia/cogarch/`).

The notion of a ‘complete system’ is not uncontroversial, but some definitions have been attempted. According to [Franklin, 1995], a complete system is an agent structurally coupled to its environment and autonomously pursuing its own agenda. In this sense, complete systems are close to what Davis calls ‘problem space agents’ in [Davis, 2001], that is computational or robotic systems that aim to display generalised capabilities. In particular, Davis insists on three characteristics that complete agents must present, namely *holistic designs*, to mean the fact that cognition should be related to and incorporate also non-deliberative processing, *meta-cognition*, that is the ability to monitor one’s own internal control states in order to modify or change them to further their aims or achieve them in a more effective manner; and *self-actuation*, that is the ability to modify one’s own behaviours, capabilities and environment in pursuit of given or self-developed roles.

In the research traditions initiated by [Minsky, 1986] and [Sloman, 1993] the attempt is to be comprehensive and to provide complete architectures for human-like systems. In the previous Sections we already provided an overview of Sloman’s approach, which is very general and comprises the attempt to understand the space of possible different architectures, the design options and trade-offs relevant to different sets of requirements, and the trajectories for learning and evolution through those requirement spaces (niches). An example of this kind of research can be found in [Scheutz, 2000, Scheutz and Sloman, 2001], where the authors report on the exploration of small regions of design space for relatively simple ‘organisms’.

A different, but surely complementary, approach is the one of Michael Arbib, who has a similar interest in complete systems and has been focussing on the mechanisms underlying the coordination of perception and action. According to Arbib, the human mind constructs reality through a network of *schemas*. A schema is both a mental representation of the world and a process that determines action in the world. Arbib’s theory of schemas is based on Pierce’s notion of a ”habit” (a set of operational rules that, by exhibiting both stability and adaptability, lends itselft to an evolutionary process) and Piaget’s notion of a ”scheme” (the generalizable characteristics of an action that allow the application of the same action to a different context). Both assume that schemas are compounded as they are built to yield successive levels of a cognitive hierarchy. Categories are not innate, they are constructed through the individual’s experience. What is innate is the process that underlies the construction of categories. The theory of schemas is consistent with a model of the brain as an evolving self-configuring system of interconnected units. At `http://www-hbp.usc.edu` a good overview of Arbib’s research can be found.

With a different flavour, another trend in the research on architectures has focused on the development of comprehensive frameworks to study, evaluate and compare the different architecture proposals. One possibility is to provide a detailed analysis of the capabilities that should be supported, and then compare different architecture properties enabling such capabilities. Such approach has been adopted, for example in [Langley and Laird, 2002]. The authors individuate nine functional capabilities that should be supported in order to achieve human-level intelligence: recognition and categorisation; decision making; perception and situation assessment; prediction and monitoring; problem solving and planning; reasoning and belief-maintenance; action; interaction and communication; remembering, reflection and learning. The completeness and complexity of an architecture can be evaluated in terms of how many of these capabilities it supports. The authors then propose to characterise and compare different architectures showing the same functionalities, in terms of the internal properties that give rise to these same functionalities. Their proposal is to consider how knowledge is represented and organised; how it is used in problem-solving behaviours; and how it is acquired and possibly revised through learning mechanisms.

A still different approach is followed in [Davis, 2001]. Here, the idea is to find a way of organising architectures into kinds on the base of a taxonomy of their motivational drives. The author proposes three categories of agents: regulatory agents, domain-specific agents, and problem space agents.

Regulatory agents have no planning abilities but they are able to cope with a limited range of stimuli in their environment. Examples are provided by behaviour based agents (see for example the insect-like robots decribed in [Brooks, 1991] and the cellular automata described in [Davis, 1999]).

Domain specific agents constitute a more sophisticated category of agents that are capable of learning new behaviours in pursuit of their drives. Even at the design level, however, they remain situated within a specific domain and typically focus on a sub-set of the possible tasks within that domain. Examples are agents based on Kaelbling's reactive planning architecture [Kaelbling, 1989] and Ferguson's Touring machines [Ferguson, 1995]. Agents in this category extend the capabilities of regulatory agents through the ability to manipulate representations (typically related to plans) or select between behaviour sets, in order to achieve goals that persist over time.

Problem space agents are those intended to be capable of extending their knowledge and behaviours, in order to traverse the space of problems within a domain. Typically those agents supports many types of processing at various levels, and attempt to model general problem-solving abilities that can be of use in more than one domain or category of task. Using the terminology introduced in the previous Sections, we can say that problem space agents are capable of effective functioning in a number of closely related niches. Hence, they need to determine the kind of niche they inhabit, and be able to make predictions about the effect of actions in their current problem space. In the next paragraphs we discuss Soar as an example of architecture for generic problem solvers.

The Soar architecture [Laird et al., 1987] has been proposed as a candidate for a *unified theory of cognition* [Newell, 1990], that is, a fixed set of mechanisms and structures that can be used to provide computational models of many aspects of cognition, varying from natural language understanding and generation, to vision and supervised concept acquisition. We first briefly introduce Soar and the learning mechanism it incorporates (called *chunking*), then we describe the Symbolic Concept Acquisition system (SCA) [Miller and Laird, 1996], which is a theory of symbolic concept learning that has been implemented in Soar.

The Soar architecture incorporates a view of cognition as *goal-directed problem-solving*, and all problem solving in Soar consists of search in a problem space, going repeatedly through a processing cycle which first proposes operators to apply to the curent state; then registers a preference for one or more operators to be selected; and finally applies and terminate a selected operator, to move ahead to the next state. A graphical representation of a *problem space* is depicted in Figure 3.2, where the problem space itself is represented as a triangle arising out of a circle (the current goal) to symbolise the expanding set of possible actions that could unfold over time. Soar's fundamental data structure is called a *goal context* and is defined by four slots and their values: the goal, the problem space (to organise knowledge in a goal-related way), the state and the operator. So, if a particular domain and cognitive behaviour has to be modelled using Soar, then the first step to be taken by the

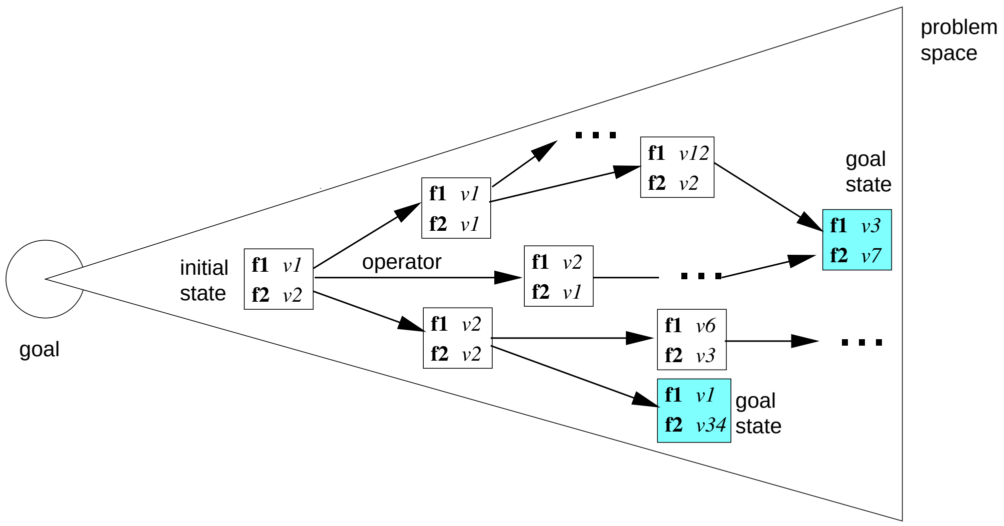


Figure 3.2: A problem space, as conceived within Soar. The problem space, represented by a triangle, arises with the attempt to solve a goal, represented by a circle at the apex of the triangle. Within the problem space, squares represents states, with their characterising features (in bold) and values (in italics). Arrows represent operators that transform states. Goal states are shaded.

designer is to translate the knowledge available to her into states and operators. The basic processing mechanism operating on goal contexts is the *decision cycle*, composed of two phases. During the elaboration phase, the production rules stored in the long term memory are accessed (these can be viewed as associations that map from the current goal context in the working memory to a new goal context). All rules that can fire do so, in parallel, resulting in changes to the features and values that define the state and suggesting new values for the goal context slots. When all the knowledge that can be elicited by the current context has been elicited, the decision phase begins. The decision procedure exploits knowledge of when to choose actions or methods (e.g. if both freeze and slowly-move-backwards are suggested then consider freeze to be better than slowly-move-backwards), interprets the domain-independent language of preferences and suggestions, and results either in a single change to the goal context or in an *impasse*. Impasses signal a lack of relevant knowledge in association with the current context. When an impasse arises, the architecture automatically begins the creation of a new sub-goal context whose goal is to resolve the impasse.

In Section 3.1.1 we briefly introduced the notions of reflection and meta-level system in the computational domain. The Soar architecture can be considered as a potentially infinite tower of implementation meta-levels [Rosenbloom et al., 1988], made available to a problem-solving agent. The 'base' level contains the initial description of the problem in terms of a state space and a set of operators for changing from one state to another. The first meta-level implements the operators using a parallel matching, parallel firing rule-based system, using preference mechanisms for selecting the operators. If an impasse occurs during the search performed at the base level, a new level of search is created with its corresponding meta-level, and so on and so forth. Whenever resolved, impasses enable Soar to learn through a mechanism called *chunking* that consists in the addition to the long term memory of a new production rule summarising the information processing that achieved the sub-goal.

The SCA model of concept learning [Miller and Laird, 1996], based on the chunking mechanism, was proposed as an alternative to gradient, probabilistic representations, such as neural nets and probabilistic declarative structures, in order to show the potentialities of discrete, symbolic, rule-based systems. In general terms, SCA is a symbolic, rule-based system that incrementally acquires prediction rules as it is trained. When the task is to predict the category of an instance without feedback SCA performs a specific-to-general search over prediction rules: it first looks for prediction rules for all features, and then progressively ignores features and searches for less specific rules. When feedback is present, the prediction algorithm is also used for learning: in this case, the algorithm specialises the retrieved prediction rule with the last ignored feature. This specialised rule is then stored as a new prediction rule. Over multiple trials, this process results in a general-to-specific search over the feature space. The representation of the learnt concepts becomes more specific as more features are incorporated into prediction rules. A simplified representation of the SCA algorithm together with a learning example is given in Table 3.1.

SCA is one among many models of concept learning and its distinguishing characteristics derive from the constraints of the Soar architecture. An extensive comparison with similar works together with an evaluation of the SCA's performance when simulating human behaviour can be found in [Miller and Laird, 1996], Sections 3.4 and 4. With respect to the approach proposed in this thesis, two major considerations have to be noted. If we consider SCA simply as a learning algorithm, then we can see it as a different way of providing a solution to the characterisation problem, that is, the problem of determining an intensional description for a set of instances. As we said, SCA is an incremental, general-to-specific concept learning algorithm for conjunctive concepts. The algorithm that we have been using as part of our cognitive architecture is also incremental, but it is able to learn disjunctions and operates a specific-to-general search over the feature space, using a separate-and-conquer heuristic (for a more detailed explanation and description, see Chapter 5). However, even though we made some minor changes to an algorithm already provided by the literature, in

Pseudocode of SCA:
1. instance i = features and values 2. while no matching prediction rule r for i 3.   ignore feature from i 4.   remember most recently ignored feature f 5. if no feedback return r 6. else specialise r with f 7. store r
A SCA learning example:

Available prediction rules:
  rule 1. null $\rightarrow$ accept
  rule 2. null $\rightarrow$ reject
  rule 3. colour red $\rightarrow$ accept
New instance: (size S, shape spherical, colour red)
To be accepted.
Abstraction order: size, shape, colour
New prediction rule:
  rule 4. shape spherical, colour red $\rightarrow$ accept

Table 3.1: A pseudocode representation of the SCA algorithm and a learning example.

order to tailor it to our needs, our work is not meant to provide any significant advance with respect to the accuracy, efficiency or even psychological plausibility of the characterisation algorithm.

In order to see the novelty of our approach, we must consider SCA in its relations with the Soar architecture. As we said, Soar is an architecture for general problem solving, so SCA can be viewed as a Soar-based solution to the category learning problem, in the sense that SCA's design is constrained by the set of symbolic mechanisms and memory structures made available by Soar. However, since SCA is not integrated into the architecture of a complete agent, the problem of individuating the relevant categories and then determining what must be stored in memory (and in what form) in order to be able to learn them (using this or that algorithm) is simply out of the scope, with respect to SCA. In our work we have addressed these two last questions, and other issues related to them as we will make clear in the remainder of this thesis.

Another well-known cognitive architecture inspired by the work of Newell is the ACT (Adaptive Control of Thought) architecture, developed by John R. Anderson. Anderson's first ideas on how human memory works date back to the early 1970s and were extended into the ACT* model of human cognition in 1983. In his personal biography (see `http://act-r.psy.cmu.edu/people/ja/ja_bio.html`), Anderson himself describes the evolution of ACT* into ACT-R in the following terms:

> The publication of the 1983 book left Anderson in somewhat of a quandary. It seemed to him that it had taken the ACT theory about as far as it could go. It was called ACT* to denote his belief that the ACT theory had reached its ultimate form. As it was a bit early to retire, Anderson announced that his research plan was "to eventually gather enough evidence to permanently break the theory and to develop a better one." However, he has proven unable to replace the theory although the next 10 years saw him engage in two major attempts to achieve just that.
>
> One attempt was to build intelligent computer-based tutors around the ACT theory. The basic idea was to build into the computer a model of how ACT would solve a cognitive task like generating proofs in geometry. The tutor used ACT's theory of skill acquisition to get the student to emulate the model. As Anderson remembers the proposal in 1983, it seemed preposterous that ACT could be right about anything so complex. It seemed certain that the enterprise would come crashing down and from the ruins a better theory would arise. However, this effort to develop cognitive tutors has been remarkably successful. [...]
>
> The second effort to "break the ACT theory" began a little later when it was clear that intelligent tutoring was not going to do the trick. It began with a sabbatical in 1987 to Flinders University in Australia when Anderson began to pursue the issue of how cognition might be adapted to the statistical structure of the environment. He developed what he called "rational analysis" [...] The fundamental idea was that to understand human cognition we did not need develop a theory of its mechanisms but only had to understand the statistical structure of the problems it faced. [...]
>
> Anderson remembers the time around 1990 as involving a lot of seemingly disconnected research directions. [...] Finally, Anderson was rewarded for enduring this intellectual ambiguity. The 1990s saw the development of the ACT-R

theory which was described in “Rules of the Mind” published in 1993. It incorporated the lessons of his tutoring work in a new theory of procedural learning. The rational analysis work played a major role in defining a better version of the subsymbolic activation processes. Anderson realized that while these subsymbolic processes were tuned to the statistical structure of the environment, one needed an overall computational structure like ACT to understand how they interacted.

ACT-R (where R stands for Rational analysis) constitutes therefore Anderson’s most developed attempt to specify how the brain itself is organised in a way that enables individual processing modules to produce cognition. On the exterior, ACT-R looks like a programming language; however, its constructs reflect assumptions about human cognition that are based on numerous facts derived from psychology experiments. One important feature of ACT-R that distinguishes it from other theories in the field is that it allows researchers to collect quantitative measures that can be directly compared with the quantitative measures obtained from human participants. When researchers create models in ACT-R, they add their own assumptions about a particular task to those already incorporated in the ACT-R’s view of cognition. These assumptions can then be tested by comparing the results of the model with the results of people doing the same tasks, e.g. using traditional measures of cognitive psychology such as time to perform the task, accuracy in the task, and even neurological data such as those obtained from FMRI.

As described in [Anderson et al., 2004], ACT-R’s main components are:

- modules,
- buffers and
- the pattern matcher.

There are two types of modules: perceptual-motor modules and memory modules. Perceptual-motor modules take care of the interface with the real world or with a simulation of the real world. The most well-developed perceptual-motor modules in ACT-R are the visual and the manual modules. Memory modules constitute the knowledge bases in ACT-R and are of two kinds: declarative memory modules, consisting of configurations of small numbers of elements called *chunks*, and procedural memory modules, made of productions. Productions represent knowledge about how we do things: for instance, knowledge about how to type the letter “Q” on a keyboard or about how to perform addition. From a computational point of view, productions are condition-action data structures with variables, but they can be seen also as a formal specification of the flow of information from cortex to basal ganglia and back again. Buffers are the constructs through which ACT-R accesses its modules (except for the procedural memory modules). For each module, a dedicated buffer serves as the interface with that module. The contents of the buffers at a given moment in time represent the state of ACT-R at that moment. Finally, the pattern matcher searches for a production that matches the current state of the buffers. Only one such production can be executed at a given moment. That production, when executed, can modify the buffers and thus change the state of the system. Thus, in ACT-R cognition unfolds as a succession of production firings.

ACT-R is a hybrid cognitive architecture. As described above, its symbolic structure is a production system; besides this, the subsymbolic structure is represented by a set of massively parallel processes that can be summarized by a number of mathematical equations.

The subsymbolic equations control many of the symbolic processes. For instance, if several productions match the state of the buffers, a subsymbolic utility equation estimates the relative cost and benefit associated with each production and decides to select for execution the production with the highest utility. Similarly, whether (or how fast) a fact can be retrieved from declarative memory depends on subsymbolic retrieval equations, which take into account the context and the history of usage of that fact. Subsymbolic mechanisms are also responsible for most learning processes in ACT-R, allowing the system to adapt to the statistical structure of the environment. They make use of 'spreading activation', first insofar as spreading activation is part of the learning that changes the weights in the neural nets, and secondly insofar as it causes information to be prioritised for retrieval from memory.

ACT-R and Soar are based on quite different foundational assumptions: for ACT-R it is the fact that implementation should be in terms of neural-like computation and cognition should be adapted to the structure of the environment; for Soar it is the fact that humans apply their knowledge in some rational manner to achieve their goals and the knowledge they manipulate is symbolic in nature. This results in different theories of control (in the sense of conflict resolution): ACT-R treats control as an automatic process, whereas Soar treats it as a potentially deliberate, knowledge-based process. A close comparison, based on the versions available at the end of 1996 is provided by [Johnson, 1997] and concludes that Soar can model extremely flexible control, but has problems accounting for probabilistic operator selection and the independent effects of history and distance to goal on the likelihood of selecting an operator. In contrast, ACT-R control is well-supported by empirical data, but has difficulty modelling task-switching, multiple interleaved tasks, and dynamic abandoning of sub-goals. For a more recent comparison between Soar, ACT-R and other architectures see also [Langley and Laird, 2002].

3.2 A scenario to study concept formation: the case of affordance concepts

An architecture-based analysis of a cognitive ability aims at unveiling the architecture requirements of a system that manifests the ability in question. Since our concern is with concept formation, we explore the design space of an agent architecture that supports the autonomous and unsupervised acquisition of different kinds of concepts.

Part of our work has been devoted to the development of a prototype architecture supporting concept formation, and our initial step was the design and implementation of a prototype system, in which an agent is able to learn autonomously a set of *affordance concepts*. The notion of 'affordance' has been inspired by [Gibson, 1979] and [Sloman, 1989] and is further discussed in Chapter 4. In what follows we give an overview of our system (the environment and the agent inhabiting it), and we briefly discuss the kind of learning that is involved.

3.2.1 The scenario in terms of requirements

In Section 3.1.2 we outlined the distinction between niche-space and design-space, and we considered a description from the niche point of view (in terms of constraints and requirements) to be the starting point of an engineering process. In the case of our system, the requirements to be met are

- the ability to survive in an environment where different causal classes exist and there is *a priori* knowledge that these classes exist, but not what they are (so that the agent has to find out what causes what);

- and the ability to survive in an environment where the causal classes can vary across time.

Our aim is to explore a design subspace where a modelling functionality is present, because we will be addressing the problem of surviving when causal classes vary across time by means of the ability to form different conceptualisations of the environment at different times. This in turns means that the ability to form a partial model of the environment (involving the conceptual structures needed to categorise some of the objects in the environment) is our new, more refined requirement.

As we already noticed, usually a niche is not determined by a physical location, however, if among the requirements set by a niche there is a requirement for modelling capabilities, then the environmental characteristics play a major role in determining the possible contents of a model; as a consequence, the complexity of the environment figures as one dimension of variation of the design-space. The other dimension of variation is given by the specific solution that is provided to account for adaptation, that is the ability to cope with environmental changes. The ability to adapt to environmental changes depends on that part of the agent's architecture which controls whether and how actions are selected on the basis of the processing of external sensor information. In Section 1.3.2 we discussed the possible evolution of an open architecture, i.e. an architecture with no built-in model of the environment. Such an architecture constitutes our starting point. We then show how this initial architecture can be extended with only meta-knowledge of the environment, i.e. the ability to build a model by interacting with the environment but with no innate knowledge of the ultimate contents of the model. It is this bootstrapping meta-knowledge that enables both the acquisition of concepts corresponding to the environment causal classes, and the use of these concepts to guide action. The complexity of the environment depends both on the set-theoretical complexity of the causal classes to be learned (e.g. whether there are disjunctive concepts, whether there are overlapping concepts, etc.) and on the type of environmental changes.

To start with, the virtual scenario that we have been using to explore the architectural characteristics of an agent that can learn autonomously a set of concepts consists of an agent autonomously moving around and acting in an environment where different kinds of objects exist. The situation is depicted in Figure 3.3. Our attention has focused on affordance concepts, hence on the mechanisms that enable the agent to gather some knowledge about its environment to form new internal representations capturing aspects of the environment that are of interest for it, because functionally relevant.

The objects in the environment have properties (among others, a location and a size), they can be sensed by the agent, and some may change the agent's internal state when acted upon. These classes constitute part of the world's ontology, and can be considered as 'causal classes', i.e. classes for which causal laws exist in the world. In particular, the physical interactions among agents and objects conforms to such laws. For example if an object belongs to the class Food then it will provide energy whenever acted upon correctly, that is, whenever eaten. Depending on the causal class they belong to, the objects that can modify the agent's internal state can therefore be useful or dangerous. Which in turns means that because of the world laws, some of these causal classes are *relevant* for the agent, since it is only by acting upon certain objects and not others that the agent can achieve its goals, or

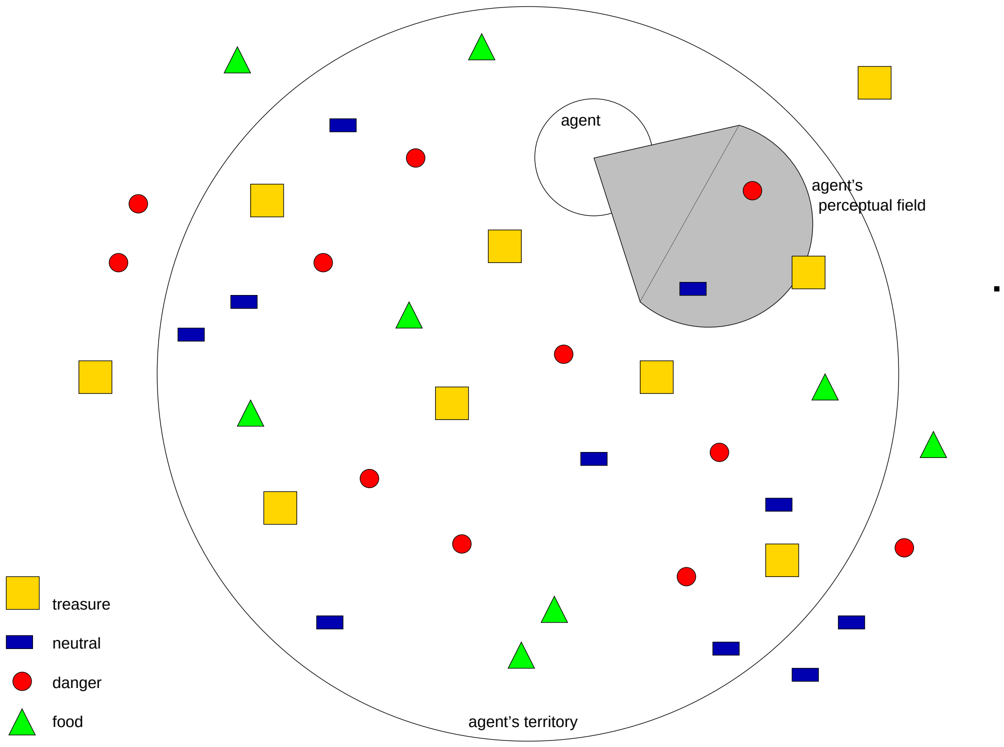


Figure 3.3: A simplified environment and the agent inhabiting it.

prevent being hurt or damaged. What we are interested in is to understand how an agent can discriminate among certain causal classes of objects, and how it can recognise the objects that instantiate these classes. Hence our aim is to show which are the architectural features that support the acquisition of such abilities as:

- acquiring a set of internal representations denoting the relevant classes of objects, and
- associating to these representations a set of rules to discriminate the objects that instantiate each class.

The environment that we have used to investigate the design requirements supporting the above abilities, contains objects that are simple (in the sense that they have no parts), do not move, and do not interact between themselves. Over time, however, the sensible qualities of the objects may change. These simplifications have been motivated by the attempt to keep the physics of the world as simple as possible, to start with, in order to be able to concentrate on the learning processes of the agent. Spatial relationships hold among the objects, but are exploited only to control the agent's movements, and are not used in the learning process.

In this environment, an agent is moving around with the ability to act upon objects. The agent has a set of internal sensors, a set of actions, and a set of external sensors to gather information from the environment. Different combinations of the values of its internal sensors determine different internal states, some of which are goal states. The agent's behaviour is caused by its internal states (including the evaluation of its own position with respect to other objects in its perceptual field). The particular set of internal states that has been adopted in this scenario (namely the levels of happiness, energy and pain) is just illustrative. On the one hand these internal states are not intended as an actual hypothesis about human cognition, on the other hand they do not constitute a restriction on the applicability of the architecture, provided the relevant associations between internal states and other parts of the architecture are mantained, as described in Chapter 4, Section 4.1.3.

The agent's goals are to increase its internal level of happiness, while keeping its internal level of energy above a certain threshold, and its internal level of pain below a certain threshold. In order to achieve its goals, the agent can perform the following external actions: move towards or away from an object, collect a treasure (from an object), eat an object. The agent can also perform some internal actions, like learning, deciding, etc. If performed on the appropriate objects, collecting treasure increases the level of happiness, while eating increases the level of energy. In certain cases, while trying to collect treasure or eat, the agent is hurt. Whenever the agent is hurt, it moves away from the object it is at, and this makes the level of pain decrease. This last one is an innate feature of the agent.

With respect to such a scenario, therefore, to acquire a model of the environment means, among other things, to acquire a set of concepts denoting classes of relevant objects such as: treasure items, food items, and danger items. An agent provided with such concepts is able to recognise the objects that make it happy (treasure items), in order to approach them and collect treasure; the objects that provide energy (food items), in order to approach them and eat them; and the objects that hurt it (danger items), in order to avoid them. Since such concepts categorise objects in terms of which actions on them have which effects that are relevant to the agent's concerns, we call them affordance concepts (see Chapter 4). For example, in the process of acquiring a model of the world the agent $A1$ may develop a concept, let's say $A1$-predicate-32, denoting objects that provide energy (if acted upon in a certain way). With respect to $A1$-predicate-32 we can speak of the $A1$-class-32 class, that is the class of all the objects that the agent $A1$ recognises as Food items. We uses the two labels to highlight the distinction between the extension of a concept (the class of objects denoted by the concept) and the intension of a concept (the internal representation used by a certain agent to denote such a class), and whenever we use the term 'concept' *tout court* we mean the intension. In principle two different architectures could develop two different concepts that actually denote the same class.[1] The classes Food (the causal class as set by the world laws) and $A1$-class-32 may or may not be equal; this depends on the concept $A1$-predicate-32; if the two classes are equal then $A1$-predicate-32 is a good concept, a concept which captures an objective feature of the world.

Learning concepts is only one part of the process of learning a model of the world; another important part concerns the acquisition of relations among concepts, above all causal relations that may express the causal laws governing the dynamics of the world and even the dynamics of the interactions between the agent and the world (these would constitute the epistemological model that the agent has of its own process of acquiring knowledge). Using a metaphor, if we take a model to be a set of propositions (describing the relevant objects, their properties and relations) and causal laws (describing the possible interactions and the dynamics of the environment), and if we take both propositions and laws as being expressed in a language, whose lexicon contains the nouns, adjectives, verbs, etc. referring to the relevant features of the world, then can say that concepts are the internal counterpart of such a lexicon. Remaining within the same metaphor, we can add that, in order to have the complete model, we still need the internal counterpart of some sort of a grammar.

[1] Another important distinction is the one between the concept and the linguistic symbols that are used to refer to it during a communication process, as we discussed in length in Chapter 1.

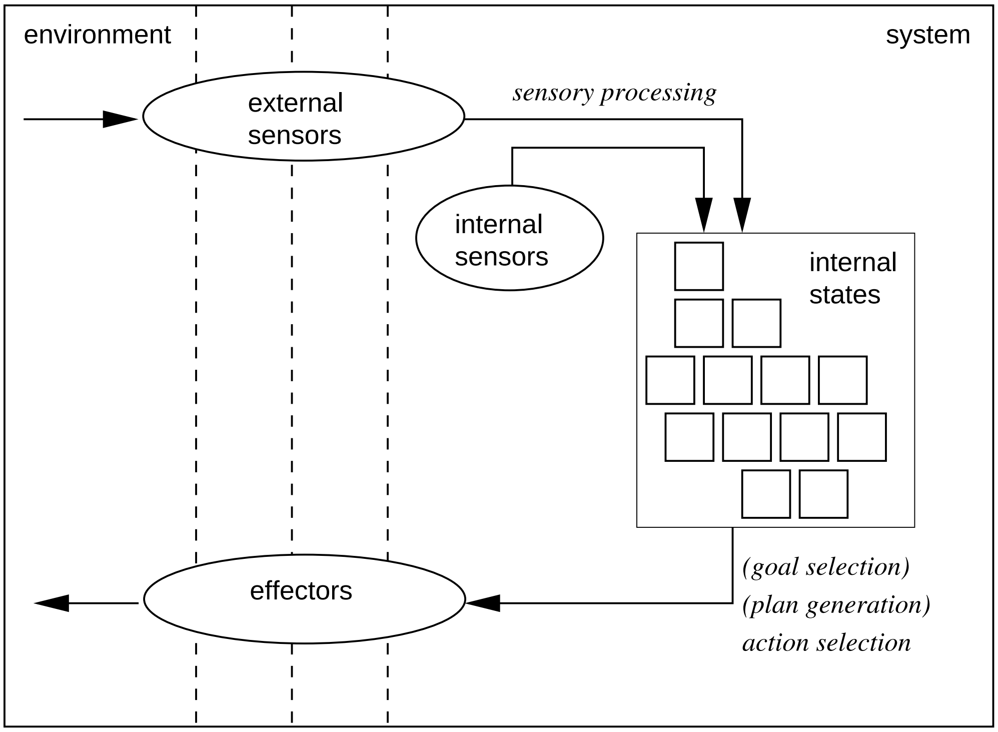


Figure 3.4: A schematic representation of a control system: the arrows represent flows of information.

3.2.2 The agent as an information processing mechanism

When the requirement/design approach is taken, that is whenever, in front of a phenomenon, we try to describe it in such a way that there are some hints about how to proceed in order to replicate it, the notion of architecture suggests itself, as we already noticed in Section 3.1. An architecture for concept formation, the phenomenon we are interested in, covers one aspect within a more complex architecture for a control system. We refer to this aspect as the *modelling dimension* and in the following paragraphs we make clear how this dimension enables a form of mediated control.

In [Sloman, 1993] a control system is defined as "a mechanism that interacts with a changing environment, including parts of itself, in a way that is determined by (a) the changeable internal state of the mechanism, (b) the state of the environment, and (c) the history of previous interactions (through which the internal state gets changed)." It is therefore a deeply causal notion, centered on the notion of action.

Since a control system is a mechanism interacting with an environment, it is possible to draw a line between the system itself and its surroundings. We should notice, however, that the separation line will be drawn in different ways, depending on the control aspects under consideration. For some aspects of the system (e.g. for meta-management) the environment could include parts of the system itself, or even the whole system. And for other aspects, it is the system that might include things in the environment, such as notes on paper or things looked at to help with reasoning, etc. This rejection of a rigid boundary between a mind and its environment is a major theme of chapter 6 of [Sloman, 1978]. Output channels can also be individuated, through which changes can be produced in the environment. 'Control' actually refers to the possibility of directing and regulating the output channels: it is control of action,[2] and the idea is that direction and regulation is the result of causally interacting internal states of the system; we refer to these as 'control states'. Part of the specification of a control system involves the description of the output channels and their ways of affecting the environment, the types and variability of the control states, and the causal influences relating them. It is very important to make clear that the control states are not 'total states' so that there is one control state at a time, as infinite state automata. Instead there are multiple control states in the same system, involved in different concurrent sub-processes. As well captured in [Sloman, 1993]: "The idea of a complete system as having an atomic state with a 'trajectory' in phase space is an old idea in physics, but it may not be the most useful way to think about a system that is made of many interacting sub-systems. For example, a typical modern computer can be thought as having a state represented by a vector giving the bit-values of all the locations in its memory and in its registers, and all processes in the computer can be thought of in terms of the trajectory of that state-vector in the machine's state space. However, in practice this has not proved a useful way for software engineers to think about the behaviour of the computer. rather it is generally more useful to think of various persisting sub-components (strings, arrays, trees, networks, databases, stored programs) as having their own changing states which interact with each other."

Speaking of control states directing and regulating an output channel describes only one direction of the causal relation between the system and the environment, namely the one that goes *from inside the system towards the environment*. Our initial definition of control system however contained also the idea of causal influence of the state of the environment on the internal states of the system: we are therefore assuming that, besides output channels, also input channels can be individuated, so that a causal relation directed *from the environment towards the system* also holds. In very simple control systems (for example, a computer program that is given a problem and then runs until it produces a solution) this outwards/inwards causal relation may hold only at the beginning, when the system is started up, and have the effect of determining the system initial state. In more interesting cases, on the other hand (and this seems to be true for any biological system, no matter how simple), such a relation persists, giving rise to a continuous feedback, and more generally continuous sensory information about what is happening, from the environment towards the system. This continuous flow of information can be used for a more effective control of action. Taking into account what we said up to now, we can therefore schematically represent a control system as in Figure 3.4: this schema is very general because it does not specify the details of the input/output channels and of the internal states; it merely states the existence of one or more a causal relations from the environment to the system and vice versa, it suggests the idea that many coexisting, interacting internal states (possibly, of different types) can be individuated, and it makes clear that the boundary outwards/inwards is not uniquely drawn. This implies that we allow for a part of the system to receive inputs from other parts. Sometimes this can be regarded as sensing, sometimes as being acted on.

Let's now consider a concrete example of a control system with a feedback-loop, where the output is discrete (on/off) and the sensing is continuous, with a threshold (the example is taken from [Sloman, 1993]): a thermostat, represented in Figure 3.5. A thermostat typically has three control (internal) states, one set by the temperature in the environment (`i-state1` in the picture), one set by the temperature threshold (`i-state2` in the picture) and one set by the control knob, maneuvered by the user (`i-state3` in the picture); the three in com-

[2] External as well as *internal* actions: if internal states and mechanisms are included as part of the environment, then 'output' will also include internal effects.

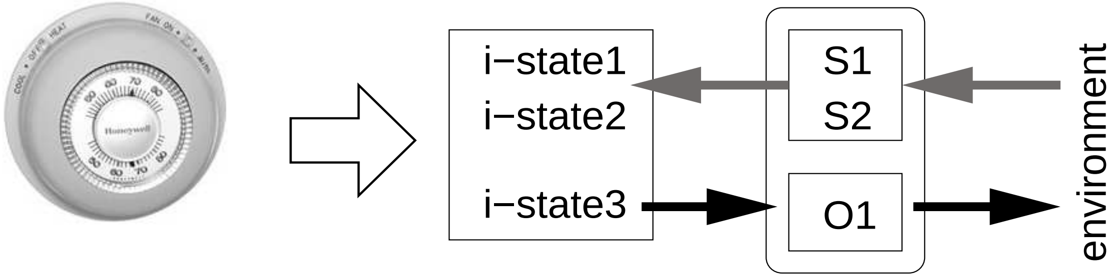


Figure 3.5: A thermostat and a schematic representation of its control architecture.

bination control the output channel, e.g. turning the heating system on or off. The arrows in the picture stand for causal relations: what we would like to stress here is the fact that each of these relations may involve different 'things', and can be described in different terms. For example, the causal flow going from the environment to the system via the input channel `S1`, and influencing the internal state `i-state1`, can be described in terms of the physical interactions between the temperature of the environment and the dilatation of a particular metal (e.g. mercurium) — a thermodynamical interaction — and then the dilatation of a fluid and its motion within a container with a certain shape — a mechanical interaction. If, instead of a mechanical thermostat, we were dealing with a digital thermostat, then the same relation could be described in terms of changes in the electrical resistance of the thermistor. At this level of description, the two control systems (that is, the mechanical and the digital thermostats) involve quite different causal relations. Both cases however could be subsumed under the same conceptualisation if causal links are described in terms of flow of information: `i-state1` can be considered a belief-like control state, a passive state produced and changed by causes 'flowing into' it; while `i-state2` can be considered a desire-like control state which, in combination with the other states, makes causation to 'flow away', thus initiating, maintaining or modifying processes.

First of all the thermostat example shows that in certain cases, when describing complex causal systems, it is possible to assume a particular point of view, according to which part of the causal interactions linking the various internal states of a control system can be viewed and described as a system of information processing mechanisms. Other points of view, however, are also possible, giving rise to descriptions based on quite different ontologies. In order to understand how two or more levels of description of one and the same thing can coexist, the idea of 'virtual machines' implemented in 'physical machines' can be used: this theoretical move actually asks for a proper understanding of the notion of implementation. A seminal theory of implemented virtual machines can be found in [Scheutz, 1999], but we will not discuss this issue in the present context because doing so would lead us too far away from the main topic of this research.

Secondly, the same example makes clear that internal states can have a different 'quality' both with respect to the flow of information, and with respect to their relationships with actions. For example, in [Sloman, 1993] different kinds of control sub-states are proposed, such as desire-like states, belief-like states, imagination-like states and plan-like states. Differences are given in terms of: whether or not they initiate processes (desire vs belief), what causes them (belief vs imagination), and what is their content (belief and desire vs plan).

Even though in complex architectures more sophisticated capabilities may involve states that are neither believed nor desired (e.g. plan-like or imagination-like states), belief-like states and desire-like states seem to constitute the minimal variety that has to be present in order to be able to speak of an information processing control system. In particular, the distinction between belief-like states and desire-like states makes clear the fact that the outward/inward causal relation present in a control system has at least two forms of instantiation: it can be a channel to gather information about what is needed, so that action will tend to be initiated, and it can be a channel to gather information about what is the case. This is captured, for example, in Section 2.4 of [Sloman et al., 2004], where a distinction is made between need sensors and fact sensors: "Need sensors have the function of initiating action, or tending to initiate action (in contexts where something else happens to get higher priority), to address a need, whereas fact sensors do not, though they can modify the effects of need sensors." For example, for most animals, merely sensing the fact of an apple on a tree would not initiate in itself any action relating to the apple. However, in the presence of a need for food that has initiated a process of seeking and consuming food items, the factual information about the apple could influence such a process and transform it into a more specific process of reaching for the apple, catching it and bringing it towards the mouth. So the authors conclude that in the case of a fact sensor sensing an apple "the sensing of the apple has no motivational role. It is a belief-like state, not a desire-like state". More precisely, fact sensors produce information that could be used for varieties of different needs: "for instance, knowing where a fig tree is could be useful whether there is a need for food or a need to climb out of reach of a predator. Information structures that have that kind of diversity of function could be called 'percepts' or 'beliefs' depending on their precise causal relationships and whether they can endure beyond the sensor states that produce them."

As we noticed above, a control system involves two directions of causal relation: an inward/outward flow, supporting the behaviour of the system, and an outward/inward flow, supporting the system's initialisation and the eventual feedback loops. Belief-like states and desire-like states are connected to this double flow, in the sense that belief-like states are the end-point of the outward/inward flow, while desire-like state are the start-point of the inward/outward flow.[3] Adopting the terminology introduced so far, we could be tempted to say that concepts are components of belief-like states, so that concept formation has to do with understanding how the components of belief-like states arise in the first place. But this is not correct because also desire-like states have semantic content. We prefer therefore to characterise concepts as the components of *explicit mediated control states*.

The distinction between direct and mediated control states is acknowledged and discussed in [Sloman et al., 2004], section 2.5. Mediated control states involve the use of enduring representations, while direct control states immediately tend to initiate action, even though we can still speak of direct states as having an implicit semantic content. "Whereas the earliest organisms had sensors and effectors very directly connected so that all behaviours were totally reactive and immediate, evolution 'discovered' that for some organisms, in some circumstances, there are advantages in having an *indirect* causal connection between sensed needs and the selections and actions that can be triggered to meet the needs. This indirectness involves the use of an intermediate state that 'represents' the need, and is capable of entering into a wider variety of types of information processing than simply triggering a response to the need." Possible uses of intermediate states therefore range from allowing different

[3]This is related to the idea that desires and beliefs are both related to changes in the state of the world but they differ in the 'direction of fit': when there is a mismatch, beliefs tend to get changed to produce a match (fit) and desires tend to cause something else in the world to be changed to produce or preserve a match.

sensors to contribute data for the same need, to allowing conflicting needs and alternative responses to the same need to be compared and evaluated, to allowing associations between needs and ways of meeting them to be learned, and used, etc.

In general, the use of the intermediate states *explicitly* representing needs and sensed facts requires extra architectural complexity, but it also provides opportunities for new kinds of functionalities [Scheutz, 2001]. For example, if need-representations and fact-representations can be separated from the existence of sensor states detecting needs and facts, it becomes possible for such representations to be *derived* from other things instead of being directly sensed. So, the distinction between direct and indirect control states constitutes a particular dimension of variation for information-processing architectures. It it this dimension that we refer to as to the 'modelling dimension'.

Along the modelling dimension, a particular form of learning is the improvement of the semantic 'quality' of the internal states, in order to move from implicit semantic content to explicit representations. Learning affordance concepts, as it is explained in Section 4.1.3, is a special case of this form of learning. In Chapter 5 we then discuss in detail a series of successive steps of the possible transformation of an agent's architecture, that show how an innate fixed behaviour can be transformed (e.g. by evolution) into an innate adaptive behaviour where such a learning process is present.

3.3 Summary

In this Chapter we have introduced the notions of architecture, design space and niche space. An architecture is a structure, which can be analysed in terms of component parts, and which includes those aspects of a system that are constant over time and across different application domains. The component parts of an architecture are usually identified in terms of their function: it is this property that is exploited in an architecture-based analysis of a cognitive phenomenon, which means that the cognitive phenomenon (e.g. concept formation) is explained by providing an architecture that supports its constitutive functions.

An architecture-based analysis requires the adoption of a design-based approach. The design space refers to the space of all possible designs, while a niche is a set of constraints and requirements for a set of designs. An architecture-based analysis involves exploring the subset of the design space relevant to a particular 'niche'. A design is a description of how an actual system works in order to meet the demands of its niche (e.g. performing the individuated tasks, with the individuated sensing and acting capabilities, etc.) The description should be given from the standpoint of an engineer that is trying to figure out how the system should work, in order to satisfy the demands of the niche.

Section 3.1.3 gave an overview of research on architectures, while in Section 3.2 we introduced the scenario that has been used to study the formation of affordance concepts. In general, it is difficult, if not impossible, to *prove* that some bits of an architecture are *needed* to enable a certain cognitive function. Therefore, we first show that a particular architecture actually works (more precisely, we show that a particular architecture supports the autonomous learning of affordance concepts – which is not obvious at first sight), then we analyse such an architecture to get a deeper understanding of why it works, and finally we argue for the relevance of certain particular features of the various architectures. This corresponds to the exploration of a region of design-space and of a particular niche.

The scenario that we have been using to explore the architectural characteristics of an agent that can learn autonomously a set of concepts consists of an agent autonomously moving around and acting in an environment where different kinds of objects exist. The situation is depicted in Figure 3.3. What we are interested in, is to understand how an agent can discriminate among certain causal classes of objects, and how it can recognise the objects that instantiate these classes.

Part II

Development of an architecture for the autonomous formation of affordance concepts

Chapter 4

Affordance concepts

4.1 Affordances: an ecological perspective

This Chapter is devoted to a discussion of the notion of *affordance*. In this section we give an overview of Gibson's theory of affordances and discuss the tension present in his writings between considering affordances as 'perceptual invariants' [Gibson, 1979] and considering them as 'possibilities for actions' [Gibson, 1950, Gibson, 1966]. The notion of *affordance* provides a good candidate for the building blocks of a goal-driven world-model, even though it does not exhaust the variety of conceptual structures that might be needed by an intelligent autonomous agent. We therefore make a proposal to define *affordance concepts* and to capture them in architecture-based terms. As we made clear with the long review provided in Chapter 2, concepts show very different faces when looked at from different perspectives (e.g. how they are used to perform different cognitive tasks such as categorising, identifying, or solving a problem), thus giving rise to different, often incompatible concept theories. The solution that we propose is along the lines of what we may call a dual theory where, to account for the different functions that can be performed by concepts, many different representational formalisms are 'attached' to the 'same' concept, whose identity is given by its acquisition history and its sustaining mechanism.

Section 4.2 is devoted to discussing the question of whether distinct kinds of concepts can be individuated, and on which bases. Our analysis shows that processing-based distinctions facilitate the identification of concept kinds, basically affordance concepts and goal-derived concepts on one side, and taxonomic concepts on the other. Taxonomic concepts account for regularities in the environment, independently of the agent, and rely on observation. Goal-derived and affordance concepts account for regularities in the coupling of the agent with its environment, and rely on interaction. This will provide the background for the design choices that have been guiding our architecture design and implementation. We discuss affordance concepts in section 4.1.3, while the issue of the existence of different concept kinds and our proposal for an initial coarse grouping based on processing differences is left for Section 4.2.

4.1.1 Overview of Gibson's theory of affordances

The notion of *affordance* is due to J. J. Gibson, but has never been given an ultimate definition by its creator [Jones, 2003]. In Gibson's writing in fact there is a tension between considering affordances as 'perceptual invariants' [Gibson, 1979] and considering them as 'possibilities for actions' [Gibson, 1950, Gibson, 1966].

In [Gibson, 1979] we find the following definition of what constitutes an *affordance*:

> The affordances of the environment are what it offers the animal, what it provides or furnishes, either for good or ill. The verb to afford is found in the dictionary, but the noun affordance is not. I have made it up. I mean by it something that refers to both the environment and the animal in a way that no existing term does. It implies the complementarity of the animal and the environment [...] If a terrestrial surface is nearly horizontal (instead of slanted), nearly flat (instead of convex or concave), and sufficiently extended (relative to the size of the animal) and if its substance is rigid (relative to the weight of the animal), then the surface affords support. [...] Note that the four properties listed — horizontal, flat, extended, and rigid — would be physical properties of a surface if they were measured with the scales and standard units used in physics. As an affordance of support for a species of animal, however, they have to be measured relative to the animal. They are unique for that animal. They are not just abstract physical properties. (p. 127)

According to this definition, affordances are perceptual invariants. They are properties of the environment, that become apparent when perception is approached from an ecological perspective, that is when the system animal-environment is observed in its totality. In previous writings however, even though the term 'affordance' is not yet used, affordance-like notions are discussed in a much closer and explicit relation to actions. For example, in [Gibson, 1950] we find:

> Our own experience of the visual world can be described as extended in distance and modelled in depth; as upright, motion-less as a whole, and unbounded; as coloured, textured, shadowed, and illuminated; as filled with surfaces, edges, shapes, and interspaces. But this description leaves out the fact that the surfaces are familiar and the shapes are useful. No less than our primitive ancestor, we apprehend their uses and dangers, their satisfying or annoying possibilities, and the consequences of action centering on them. (p. 198)

The ecological perspective is already there, implying that the animal and the environment can only be adequately described when considered in relation to one another. But what is stressed, is that animals have been designed to detect properties of the environment that are directly relevant to them, usually because those properties provide cues to the possibilities for action afforded by environmental objects.

From a Gibsonian perspective there are inevitably two orders of problems connected to perception: for sure it must be explained, and most probably in information processing terms, how sensations are used to provide information about the external reality; but a further, and likely prior problem is to establish exactly what has to be the output of the perceptual system of a given organism, and how it is related, at the system level, to the organism as a whole. With respect to this second problem, affordances as perceptual invariants provide an answer to the question: what is it that we perceive. On the other hand, affordances as possibilities for actions are meant to explain how it comes about that we perceive certain perceptual invariants and not others.

The tension between these two problems, and the necessity to refer to the notion of action to establish what has to be the output of a perceptual system, is also at the center of the theoretical debate between Gibson and Marr on vision. For Gibson affordances are directly perceived by an organism through the optic array (i.e. flowing patterns of stimulation directly reflected from the object in question) without any intermediate synthesis or processing, so that perceiving is always perceiving what something means. Even though this correctly highlights the richness of the information conveyed by the optic array, no detailed explanation of what exactly is in the optic array that enables animals to have affordances is provided, leading to an underestimation of the complexity of the optic array itself, as pointed out in [Marr, 1982]. According to Marr, Gibson was misled by the apparent simplicity of the act of seeing, and underestimated the complexity of the information processing involved in vision, and more generally in perception. In Marr's own words:

> Gibson's important contribution was to take the debate away from the philosophical considerations of sense-data and the affective qualities of sensation and to note instead that the important thing about the senses is that they are channels for perception of the real world outside or, in the case of vision, of the visible surfaces. He therefore asked the critically important question, How does one obtain constant perceptions in everyday life on the basis of continually changing sensations? This is exactly the right question, showing that Gibson correctly regarded the problem of perception as that of recovering from sensory information "valid" properties of the external world. His problem was that he had a much oversimplified view of how this could be done.

The accusation of oversimplification, however, can be turned back to Marr. In fact, even though Marr recognises the importance of the Gibsonian notion of perceptual invariants, he fails to recognise Gibson's concern for affordances, that is the fact that the role of perception is not to detect *any* property of the external world, but only those that somehow matter to the perceiving agent. According to Marr the output of vision, at least in the case of humans, are descriptions of (possibly changing) 3-D spatial structures and locations, and among Marr's main contributions to the research field of artificial vision is a set of hypotheses on the information processing mechanisms that might produce such output from the optic array. This idea of treating vision as an integrated module with a clearly defined and very limited set of inputs and outputs is what Sloman calls 'the modular theory of vision' in [Sloman, 1989]: Sloman offers both a critique of such theory and a proposal for an alternative account in terms of what he calls 'the labyrinthine theory of vision' which treats vision as "a collection of modules having a much wider, and more changeable, collection of input and output links to other sub-systems." Sloman's arguments are not based on empirical evidence against the modular theory, but rather on the consideration that such theory puts forward a 'poor design'. A crucial point concerns the creative aspect of visual interpretation and the related visual ability of perceiving functions and potential for change:

> It is not often noticed that on the modular model, the description of scenes requires a much richer vocabulary than the description of images or the optic array. This requires the visual system to have what we might call "conceptual creativity": a richer set of concepts is required for its output descriptions than for its input. [...] Conceptual creativity is characteristic of all perception, since the function of perception is rarely simply to characterise sensory input. It often includes the *interpretation* of that input as arising from something else.

Apparently the visual system can make the conceptual leap from 2-D image descriptions to 3-D scene descriptions, however an analysis of the many purposes of vision leads Sloman to consider the possibility of a mechanism much more general and flexible, concluding that:

> Although perception of 3-D structure is important, it is often equally important to perceive potential for change and causal relationships that we describe as something having a certain function: for example seeing the cutting capability of a pair of scissors requires seeing the potential for relative motion of the two blades and the potential effect on objects between them [...] Not all theoretical possibilities are usually perceived. For example, every surface has, in principle, the mathematical potential for all kinds of deformations [...] However, only a subset of these logical or mathematical possibilities will be relevant to a particular perceiver in a particular situation, and different subsets may require different kinds of descriptive apparatus, some of it expressed in terms of changes that can occur in the objects and some expressed in terms of opportunities for action or constraints on action by the perceiver.

This idea of seeing the opportunities for and the constraints on action by the perceiver enables a further deepening of the notion of affordance, as is made clear in the next Section.

4.1.2 From species-specific to individual-specific affordances

In Gibson's theory, the ability to detect affordances is accounted for by learning in an evolutionary sense, which implies not the life-time of one single animal, but rather the history of the whole species. This explains why different organisms will be attuned and have the ability to afford different aspects of the environment. For example, certain animals may be sensitive to ultraviolet light or certain odours that are outside human perceptual experience, because of their particular evolutionary history; in turn, this different sensitivity can account for 'oddities' in animal behaviour.

Because of the tight connection with evolution, Gibson's affordances also provide a way of explaining innately preprogrammed reactions to the environment, such as babies' interest and preference for human faces. On the other hand, Gibson's account does not provide sufficient insight for behaviour responses that are situationally and culturally determined through an individual history (as opposed to a species history), for example the fact that a red pillar box affords postage.

If we look at the ecological psychology tradition, which originated from Gibson's work, we see that many studies in the past quarter century have examined the physical basis for perception of affordances, such as the climbability of steps [Warren, 1984], the safety of gaps [Burton, 2000, Mark et al., 1999] and the suitability of different surfaces for locomotion by infants at different stages [Adolph and Eppler, 1998]. These studies address the issue of unveiling the invariants grounding the affordance in question (e.g. a stair raiser being less than 88% of a person's leg length), but the issue of whether and how affordances are learned is rarely addressed. One exception is the theory proposed by E. Gibson in [Gibson, 2000a, Gibson, 2000b], according to which learning affordances is a process of differentiation and selection which involves learning the relations between the organism's power of control and some opportunities or constraints offered by the environment. The first step in affordance learning is an exploratory activity, whose main output is learning control of an event. This requires learning how to orchestrate a sequence of self-generated actions in preparation for an anticipated outcome. An important part of the exploratory activity is the observation of consequences of actions. Human perceptual systems are designed for cycles of perception-action, and when some kind of environmental contact is achieved through action, the contact will be perceived in its turn, and its usefulness (uselessness) can be evaluated.

The next step is a selection process whose output is the selection or recognition of an affordance relation. E. Gibson proposes two main principles of selection: i) selection for an affordance fit, and ii) selection for unity, order and economy. What has to be differentiated and then selected is a sequence of actions. The selection is determined (at least partly) by the kind of fit between actions performed and the ensuing consequences. Types of fit that are of interest (for human infants, as observed in the experiments) are: providing further information (e.g. turning an object to see its back); providing comfort; providing control of an event; causing a predictable change; etc. In most cases, good fits are those obviously serving functions that are of biological value to the organism. Since economy of action and reduction of perceptual information also works as principles of selection for increasing specificity, over development perception-action cycles become more and more refined and better differentiated thanks to constant practice. An example is the achieving of economy by generating a characterisation of classes of objects presenting a certain affordance fit, such as graspable or edible, in terms of external features. Sometimes economy is achieved by the discovery of order (e.g. perceiving an object as a unit; perceiving intermodal redundancy; generating a characterisation of classes of objects presenting a certain affordance fit, such as graspable or edible, in terms of external features); sometimes it is achieved by discovery of simpler means that improve the effectiveness of the behaviour (for example, invariant relations between events, such as the invariant relation between performing an action and the accompanying proprioceptive information.)

The theory proposed by E. Gibson exploits the notion of affordance as effectivity for an animal and assume that affordances guide the animal's behaviour as to what are possible or impossible actions within the environment. If acquired and idiosyncratic patterns of action are taken into consideration, alongside instinctive, species-specific ones, the notion of affordance can be enlarged to cover any aspect of the environment that is relevant (in a given situation) because either it affords or prevents the bodily interaction of the animal with the environment. A full account of conceptualisation in terms of patterns of bodily interactions is given in [Glenberg, 1997], where the author claims that "memory evolved in service of perception and action in a three-dimensional environment, and that memory is embodied to facilitate interaction with the environment."

Two broad work-models for memory are proposed, with a distinction between (i) an automatic use of memory, where patterns of action based on the environment are automatically, and without intention meshed with patterns based on previous experiences; and (ii) a conscious and effortful use of memory, where patterns from the environment are suppressed so that conceptualisation is guided only by previous experience. The situation of a person following a path provides an example:

> Recognition of the path as a path, arises from an exploration of the environment and a fit between the environment and embodied knowledge. [...] Thus projectable properties of the environment (arrangement of rocks, twigs, and soil) are encoded in terms of how you (with your particular body) can interact with that environment (e.g. whether the distances between the rocks in the creek can be broached). Other patterns of interaction come from memory, for example,

> patterns representing goals such as "get home without getting wet". In conceptualizing the environment as a path, the spatial-functional patterns based on projectible properties from the environment are combined or meshed with the patterns from memory. The meshed pattern dictates how (or if) the body can be moved in a way that simultaneously satisfies both sets of patterns of action (e.g., "Can I, with my body, get from rock to rock without getting wet?"). This sort of mesh is a possibility because all of the patterns are embodied, that is, they are all encoded in terms of how your body constrains actions. When the patterns can be meshed into a plan for coherent action (e.g., stepping across the rocks), the rocks, soil, and twigs become (for you) a path.

According to Glenberg, the representations encoded in memory are patterns of action derived from the projectable properties of the environment (i.e. properties of the environment that can be specified by information available in the optical-flow field) *meshed* with patterns of interaction based on previous experiences. These two sets of patterns are integrated by virtue of their analogical shapes, and both constrained by how our bodies work. Such embodied representations are meaningful *tout-court* because they arise from the world, and are directly grounded by virtue of being lawfully and analogically related to properties of the world and how those properties are transduced by our perceptual-action systems. So, for example, the embodied meaning of the cup on my desk "is in terms of how far it is from me (what I have to do to reach it), the orientation of the handle and its shape (what I have to do to get my fingers into it), characteristic of its size and material (the force I must exert to lift it), and so forth. Furthermore, the meaning of the cup is fleshed out by memories of my previous interactions with it: pouring in coffee and drinking from it. Those memories make the cup mine." The overall idea is that objects fall into the same basic category because they can be used to accomplish the same interactive goal, and since the same object may be useful for accomplishing a variety of goals, categorisation can be flexible and context dependent.

4.1.3 Our proposal to capture affordance concepts

In the previous Section we made a distinction between species-specific affordances and individual-specific affordances and we saw how the notion of bodily interaction can be used to enlarge Gibson's original notion of species-specific affordance to cover also individual-specific affordances. Taking into account the examples given in the previous Sections we now detail our own proposal to define affordance concepts and to capture them in architecture-based terms.

In order to define affordance *concepts*, we exploit the notion of affordances as 'possibilities for actions'. To start with, however, we adopt a simplified model of interaction, according to which there are only primitive actions, and we do not take into account structural relationships. This will enable us to define an initial architecture for the autonomous acquisition of affordance concepts. In Chapter 6 we will discuss the limitations due to the simplicity of our scenario and consider how they could be overcome with further research.

According to Gibson, affordances become apparent when perception is approached from an ecological perspective. A basic assumption of the ecological perspective is that the animal and the environment have been co-designed through evolution and are therefore 'mutually compatible'. This implies that the animal and the environment can only be adequately described when considered in relation to one another. So, to say that animals can detect affordances actually is to say that animals have been designed by evolution to detect properties of the environment that are directly *relevant* to them, usually because their survival depends on these properties (for example, surfaces, textures, substances, but also properties such as ‘graspable’, ’stand-on-able’, sit-on-able’, etc.) To make this notion of relevance more precise, we say that given an autonomous agent inhabiting an environment where different kinds of objects exist, these objects can be partitioned into relevant and irrelevant (for that agent) depending on whether they can modify the agent’s internal states, and hence be useful or dangerous, when acted upon. Slightly differently, if the agent has goals pertaining to its internal states (e.g. the goal of increasing its internal energy), we also say that a category of objects is relevant for an agent either because the objects can be used to achieve the agent’s goals, or because they prevent such an achievement. For example, if the agent has the goal of increasing its internal energy and eats an object O to achieve such goal, then O is said to belong to a relevant category, the category ‘Food’ or ‘Edible items’.

Our proposal is to consider an affordance concept to be a concept denoting a relevant category of objects, in the sense defined above. Affordance concepts are tightly related to the notion of action, insofar as the objects subsumed by affordance concepts are individuated by acting on them. Given an agent and an action a performed by that agent, we can in fact distinguish the objects that satisfy the agent’s expectations on the results of acting with a upon them, and those that do not. This leads to the notion of *positive* and *negative* affordance concepts, respectively. For example, if we call Food the class of all the objects that satisfy an agent’s expectations on the results of eating (e.g. an increase in its energy level), then the concept associated to it is a positive affordance concept, and we say that ‘food items’ afford eating.

How can we tell whether an agent can think of an affordance concept, for example one that corresponds to the category Food? Assume we observe an agent and see that, when it is hungry, it does not just try to eat whatever happens to be near it, but continues moving around until it finds something that has a certain appearance, something that ‘looks like food to it’. To generate the observed behaviour, the agent’s architecture might contain a condition/action rule expressing something like *If hungry and sense item with features $f_1, ..., f_n$ in visual field, then approach it and eat it; else move on*. Shall we say that such an agent has an affordance concept FOOD that corresponds to the category Food, as described above? Our answer is no, or at least not if this is the whole story about our agent. Probably, something like the rule we sketched above should be present in the agent’s architecture, but what really matters in order to establish whether the concept FOOD is present, is *how* such a rule happened to be part of the architecture, in particular whether it is innate or acquired, and whether it has special relations with other rules. Our intuition is that we should say that the concept FOOD is present if i) the rule has been acquired and ii) the acquisition process exploited the relation between the action of eating and the expectation of an increase of the internal energy as a result of eating.

Let’s consider again the scenario that we introduced in Chapter 3, which consists of an agent inhabiting an environment where four different kinds of objects exist (namely, treasure-store items, food items, danger items and neutral items). With respect to this scenario, the relations existing between the object categories, the actions that the agent can perform, and the possible results on the agent’s internal state are represented in Table 4.1.

Apart from moving around, the agent can perform two other primitive actions, namely eating and collecting a treasure. The motivational structure of the agent is such that the agent’s default behaviour is to approach the closest object in its perceptual field and collect

Object kinds	Agent's actions	Results on internal state
treasure items	`collect_treasure`	increase happiness
	`eat`	nothing
food items	`collect_treasure`	nothing
	`eat`	increase energy
neutral items	`collect_treasure`	nothing
	`eat`	nothing
danger items	`collect_treasure`	increase pain
	`eat`	increase pain

Table 4.1: Relations between: object categories, actions that can be performed by the agent, and possible results on the agent's internal state. The concept learned by the agent correspond to the object kinds.

treasure from it. If the agent gets hurt while acting, it will move away from the object. The default behaviour changes to approaching the nearest object and eating it whenever the agent gets hungry (the reaction to being hurt remains the same). The agent does not know in advance what treasure items, food items or danger items look like, but it has a mechanism to acquire such a knowledge, which exploits the fact that food items afford eating, treasure-stores afford the collection of treasures and danger items prevent each of the previous actions, since they cause the agent to be hurt. The architectural feature that we propose to enable the agent to acquire this knowledge is the existence of a set of condition-action rules where the condition part expresses the performance of an action, while the action part expresses an expectation on the results of the action. This means that the agent knows which action can possibly achieve which internal goal. Such knowledge is implicit in the sense that internal goals are not explicitly represented, and is captured by the fact that each internal state of the agent which deviates from stability (apart from being dead) triggers both the intention to perform a certain action, and an expectation, with a definite content, on the result of the action (in terms of a resulting internal state).

Expectations are not generic, but rather have a definite content in terms of observable modifications of the agent's internal state. In particular, for each action that has an associated expectation there is an affordance concept that could possibly be learnt, through a processing cycle that involves the following steps:

- at the beginning of each processing cycle, the agent gets information on the external world and information on its own internal status;
- once the internal state has been checked, a motive is generated that sets both the intention to perform an action and an expectation on the result of that action;
- once the action has been performed, the actual result is checked against the expected result and thus provides input for the categorisation mechanism.

This sequence of events that starts with the generation of a motive and ends with the checking of an expectation is what we call *an experience*, as represented in Figure 4.1. When checking an expectation, we consider two cases: either the expectation is fulfilled or it fails, which implies that to start with we will not consider the case of fulfilling an expectation in degrees. If the expectation is fulfilled, a new motive arises. If it fails, a new action is selected and

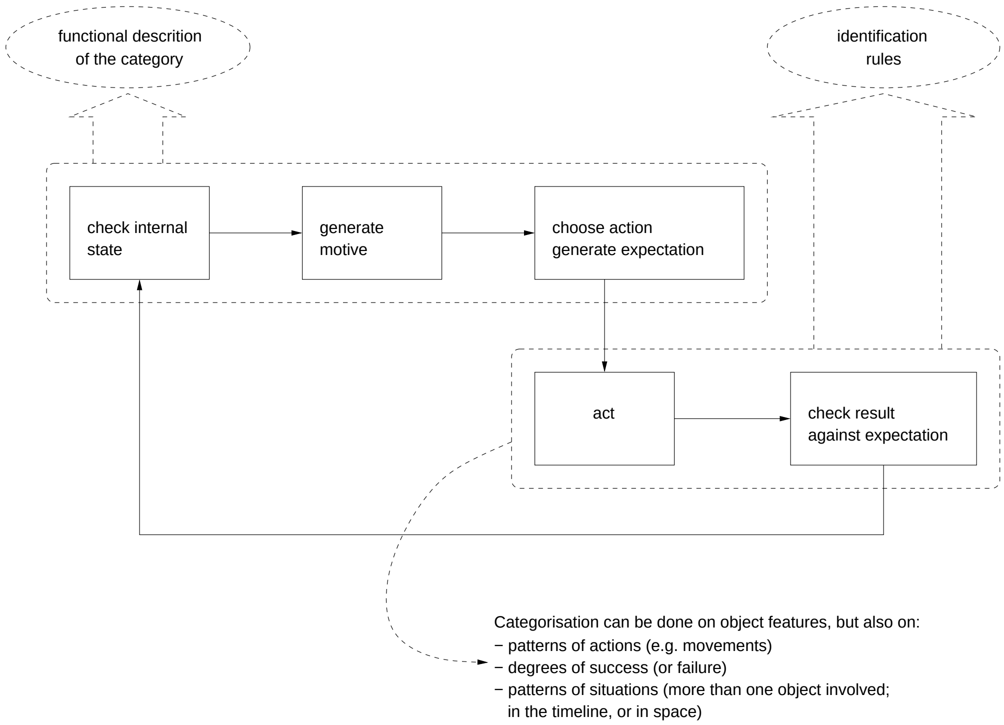


Figure 4.1: Diagram of an experience.

the cycle continues until it is appropriate to perform a new check. So, the two more general kinds of experience cycles through which the agent can go are:

- ... [(motive) (action + expectation) (check: expectation fulfilled)] ...
- ... [(motive) (action + expectation) (check: expectation failed) (action + expectation) (check: expectation failed) ... (check: expectation fulfilled)] ...

In the first case there is only one expectation-check and the expectation is fulfilled; in the second case there are two or more expectation-checks and at least one failed expectation. In both cases there is room for acquiring information to be used for successive categorisation, because an experience will always provide a sensor reading of the target object's features and knowledge of whether the target object is an 'useful' object with respect to the current motives.

For any agent that has these initial characteristics there is therefore a *space of possible affordance concepts* that the agent can learn. If the action-expectation model is built-in (or innate) then exploratory actions that have no definite expectation are ruled out. In this case, no new affordance concepts can be learnt – i.e. concepts relating to learnt, but not innate, expectations. On the other hand, if the model is acquired then the agent has that more general capability, which might lead to the discovery of types of affordances that were not innately determined. In the current implementation the agent has innate expectations about the results of its actions. We envisage however a modification of the agent architecture that will not need the expectations to be innate, as is explained in Chapter 6.

4.2 Are there different kinds of concepts?

Even though research on concepts has been abundant in many fields (psychology, philosophy, cognitive science, machine learning), the question of whether there are distinct kinds of concepts has been rarely addressed, because most of the research on concepts has focused almost exclusively on natural object concepts (chair, bird, tool, etc.), within categorisation tasks [Smith and Medin, 1981, Solomon et al., 1999]. One of the few attempts to bring together candidates for kinds of concepts that have emerged across the different theoretical perspectives is to be found in [Medin et al., 2000], where the authors propose criteria based on structural differences, processing differences, and content differences for distinguishing concept kinds.

Already in [Medin et al., 2000] the idea that kinds of concepts can be individuated by referring to content-laden differentiating principles is criticised. The content-based criterion in fact is grounded on the claim that different domains of conceptual knowledge can be individuated (e.g. naïve biology, naïve psychology, naïve physics). Such a claim however, though having some empirical foundations [Carey, 1985], is far from being validated, and it is very difficult to give a precise definition of domain [Hirschfeld and Gelman, 1994]. For this reason we maintain that the actual choice for providing a characterisation of concept kinds is between structural and processing differences.

4.2.1 Concept kinds individuated on the basis of structural differences

Criteria for kinds of concepts based on structural differences would distinguish among different kinds of concepts by looking at differences in the kinds of their constituent features, and in the relations among them. The implicit assumption is therefore that concepts can be analysed into constituent features. Usually, evidence is first provided for a category distinction. Then the hypothesis of structural differences among the corresponding concepts is stated, assuming that concepts must be distinct if the associated categories are distinct.

In the literature, distinctions have been proposed that rely on the following oppositions: i) necessary and sufficient defining features *vs.* defining features that are only probable; ii) compositional features *vs.* functional features; and iii) perceptual features *vs.* relational features.

Case i): In Chapter 2 we saw how the classical view on concepts, according to which concepts are structured entities encoding necessary and sufficient defining features for their instances, is opposed to the prototype theory, according to which only probable features are encoded. The opposition between the two views is cast in terms of how concept constituent features are to be used in the process of identifying members of a given category: an all or none matching process is used in the classical case, while a partial or probabilistic matching process is used in the prototype case. However, in [Armstrong et al., 1999] it is argued that, since graded responses are found also for well-defined categories such as PRIME NUMBER, the distinction between well-defined and fuzzy categories categories is ill-grounded. This suggests one should discard the idea of a distinction between classical (or well-defined) concepts on the one hand, and prototype concepts on the other.

Case ii): In [Keil, 1989, Barton and Komatsu, 1989, Rips, 1989] another category distinction has been proposed, which opposes natural kinds to artifacts on the basis that people consider features referring to internal structure to be criterial for membership within natural kinds, while they consider functional features to be criterial for membership within artifact categories. For example, [Keil, 1989] shows that perceptual change does not matter when judging about the identity of a natural kind (a raccoon dyed and painted to look like a skunk is still considered to be a raccoon), while it does when judging about artifacts (a coffeepot transformed into a bird feeder is not a coffeepot anymore). On the other hand issues about origin matter for natural kinds while they seem to be irrelevant when judging about artifacts. As another example, [Rips, 1989] shows that external transformation is less likely to change natural kinds' identity than internal transformation (an animal transformed from a bird-like creature into an insect-like creature either due to chemical hazards or to internal maturation), while artifacts' membership tends to change if the transformation involves the function of the artifact (e.g. a tire not made of rubber versus a tire that cannot roll).

However, a succeeding study [Malt and Johnson, 1992] shows that in certain cases physical properties of artifacts (for example, for boats, the fact that they are wedge-shaped, with a sail and anchor, etc.) were judged to be more important than, or just as important as functional properties (such as carrying people over a body of water for purpose of work or recreation), thus making the alleged distinction less clear-cut. A deeper explanation of the various data, which provides a coherent picture of the phenomena, is to be found in [Ahn, 1998]. In this work Ahn proposes a causal status hypothesis, according to which the centrality of a feature to a category does not depend on the kind of feature but rather on the causal role that the feature plays (relative to the other features in the category). In other words, given a set of causally related features, the experiments done by Ahn show that in general people are biased toward considering features that serve as causes of other features more central to category membership than their effects. These findings considerably reduce the claim that natural kinds and artifacts are two separate domains and that different features are more central in categorisation as a result of this domain distinction.

Case iii): In linguistics and psychology a distinction is made between object (or noun) concepts, corresponding to clusters of perceptual features, and relation (or verb) concepts, corresponding to relational properties. Cross-linguistical studies such as [Levinson, 1999] show that lexical entries corresponding to perceptual chunks are consistent through different languages/cultures, while lexical entries corresponding to relational structures tend to vary. Also, studies on how children acquire language, such as [Tomasello, 1992, Bloom et al., 1993, Waxman and Markov, 1995, Woodward and Markman, 1997], seem to support an early bias towards noun acquisition and production.

In order to explain which words are earliest learnt by children and why, the authors in [Gentner and Boroditsky, 1999] show that the word acquisition process can be characterised by either *cognitive dominance* or *linguistic dominance*. In the case of cognitive dominance, aspects of perceptual experience form inevitable conflations that are conceptualised and lexicalised, while in the case of linguistic dominance, the world presents perceptual bits whose clumping is not pre-ordained, and it is the linguistic interaction that guides how the bits get conflated into concepts. The authors claim that dominance is divided along the continuum from open-class terms to closed-class terms: "At one extreme, concrete nouns — terms for objects and animate beings — follow cognitive-perceptual dominance. They denote entities that can be individuated on the basis of perceptual experience. At the other extreme, closed-class terms — such as conjunctions and determiners — follow linguistic dominance." The consequences in terms of early word learning is an early predominance of names for objects and individuals, and a later increase in the proportion of relational terms. This is so because noun referents are easier to individuate than verb and other closed-class terms referents. The child's task when learning language is one of attaching words in the stream of speech to their referents in the stream of experience, via the conceptual space. Given the fact that concrete objects have already been individuated prelinguistically, as reported in [Baillargeon, 1993], the acquisition of concrete nouns reduces to finding the correct linguistic label. In contrast, as argued in [Gentner and Boroditsky, 1999] "for verbs and other relational terms, isolating the word is only part of the job. The child must also discover which conflation of the available conceptual elements serves as the verb's referent in her language. Not only the intensions but even the extensions of relational terms must be learned in part from language. This is not to suggest that young children fail to perceive relations, or that they are not interested in them. On the contrary, movement, change, agency and causality are fascinating to infants. But although relational fragments are perceived from early on, there is no one best way in which they cohere into referential units." So, the hypothesis of the division of dominance predicts that nouns are acquired earlier than verbs, and this predictions relies on the fact that noun referents are easier to individuate than verb referents. Once the structural difference between nouns and verbs has been cast in terms of division of dominance, nuances among nouns can also be clarified and explained. For example [Caplan and Barr, 1991] show that certain nouns are intrinsically defined (dog, spoon, flower, etc.) whereas others are more relational in character (grandmother, uncle, etc.)

In the light of evidence from the fields of psycholinguistics and language development, the distinction between object concepts, based on the clustering of perceptual features, and relation concepts, based on the clustering of relational features, appears to be granted. However, the hypothesis of the division of dominance refers to the acquisition process of the proposed concept kinds (object concepts are acquired pre-linguistically while relation concepts are acquired through linguistic interaction), and therefore establishes a processing-based distinction rather than a structural one.

4.2.2 Concept kinds individuated on the basis of processing differences

When processing differences matter, the candidates for concept kinds that have been discussed in the literature are taxonomic concepts *vs.* goal-derived concepts.

Taxonomic concepts: Taxonomic categories and the correlated concepts have been proposed and studied by Rosch [Rosch, 1978]. According to Rosch, two general principles underlie the human ability to form taxonomies: the cognitive economy principle, asserting that the task of a category system is to provide maximum information with minimum cognitive effort; and the perceived world structure principle, asserting that the perceived world comes as structured information rather than as arbitrary or unpredictable attributes.

This last principle in particular explains the characteristic exemplar-based formation process of taxonomic concepts, which can be summarised as follows [Margolis, 1999]: It is an empirical fact that certain combinations of attributes are quite probable, in the sense that they often appear together (for example, wings co-occur with feathers more than with fur, shape co-occurs with similarity of movement more than with similarity of colour, etc.) Humans seem to be predisposed to take certain attributes as reliable indicators for kind membership ('kind syndrome'). Therefore, they tend to group together objects that are similar under those attributes as belonging to the same kind. This provides sets of clustered exemplars that can be successively analysed to extract cognitively more economical representations for the clusters.

Taxonomic concepts and the related categories are therefore created to capture the correlational structure of the physical world, and their acquisition process is a case of exemplar-based learning. Since category knowledge is induced from experiences with exemplars (and then internally represented in a more economical form), clusters of exemplars must be readily available before the actual learning process starts. In particular, the clustering is done according to the 'kind syndrome' as discussed by Margolis in [Margolis, 1999].

Goal-derived concepts: In [Barsalou, 1983, Barsalou, 1985, Barsalou, 1991] the author contrasts taxonomic categories (also referred to as object categories) and goal-derived categories on the basis of several differences: unlike object categories, goal-derived categories may activate context-dependent features of category members; their members are not especially similar to one another, so they violate the correlational structure of the environment that is exploited by basic-level concepts and in general taxonomic concepts; their typicality is based on proximity to ideals rather than on central tendency[1]; they are acquired through conceptual combination rather than exemplar-based learning. It is this last issue that grounds the distinction between object categories and goal-derived categories in terms of processing mechanisms.

According to Barsalou, goal-derived concepts and the corresponding categories are built when constructing plans to achieve goals, for example the concepts 'people to visit in California' or 'foods to eat on a diet'. Most of these concepts are *ad hoc* in the sense that they are derived on the fly to achieve the current goal and then dismissed, some of them however might become well-established in memory because of frequent processing. The methodology proposed in [Barsalou, 1991] to study goal-derived concepts and categories is based on a protocol analysis of planning. When planning an event (e.g. a vacation), the first thing that seems to be done by people is to retrieve a frame for that event, that is a set of attributes that can take different values across different instantiations of the same event. Usually, the frame is retrieved only partially and not as a rigid structure: sometimes important attributes are just 'forgotten' by one subject; the order in which attributes are considered varies with the subject; etc. However, once the appropriate frame has been retrieved, people begin to instantiate the various attributes, adopting particular values for use in the current frame. This instantiation process normally follows a process of successive refinements: first, a general class of instantiations is produced, and then this class is refined down to more specific instantiations. For example, when trying to instantiate the temporal attribute of a vacation, a subject might begin with the concept 'in October'; that same subject might then continue with the refinement 'towards the end of the month, possibly a week-end', ending up with the more specific concept 'a week-end towards the end of October'.

These successive sets of instantiations for frame attributes are what Barsalou calls 'goal-derived concepts' and the previous example shows both how goal-derived concepts are involved in planning, and also how their formation exploits a process of conceptual combination. Such a process is substantially different from the exemplar-based learning involved in the acquisition of taxonomic concepts. Goal-derived concepts (and their corresponding categories) are acquired through conceptual combination because the process of frame attributes instantiation giving rise to them, actually involves the combination of increasingly specific properties with the existing, current attribute description. This implies that exemplars for these concepts are not known before hand, but are actually formed while the concept and its

[1] An ideal is a characteristic that exemplars should have to serve a goal optimally. For example, *zero calories* is an ideal for the category *food* in the context *foods to eat on a diet*. Central tendency, instead, is the average characteristics of a category's exemplars. Both ideals and central tendency information may provide content for a prototype, and proximity to ideals works also for taxonomic concepts, e.g. in expert judgements about typicality. Actually, it seems that central tendency plays a role only for relative novices [Lynch et al., 2000].

corresponding category are formed.

Affordance concepts: Let's now turn to affordance concepts. In the previous Sections we discussed both the original theory proposed by Eleanor Gibson for learning affordances, and our own ideas on characterising and learning affordance concepts (which will be deepened in Chapter 5). Here we relate E. Gibson's insight with with our position, and then conclude with a proposal of concept types individuated on the basis of processing mechanisms.

According to E. Gibson, learning affordances is a process of differentiation and selection, involving first an exploratory activity, whose main output is learning control of an event (such as how to orchestrate a sequence of self-generated actions in preparation for an anticipated outcome) and then selection or recognition of an affordance relation. Whenever an affordance fit is selected, over time economy of action and reduction of perceptual information work as further refining principles of the perception-action cycles. A prominent example of such refinement is generating a characterisation of classes of objects presenting a certain affordance fit, such as graspable or edible, in terms of external features.

Besides our own work, other recent findings reported in the AI literature appear to corroborate E. Gibson's insight. In particular, [Stoytchev, 2005] shows that given the desired effect (in this case, the attachment of an object to a robot's body, so that the object's movements can be controlled) and information about an object shape, the appropriate behaviours to be performed to obtain the desired effect can be induced. This corresponds to the exploratory activity step individuated by E. Gibson. Examples of the second step individuated by E. Gibson, the selection process, are given in this thesis and in [Cos-Aguilera et al., 2003, Cos-Aguilera et al., 2004] where it is shown that given a hard-wired association between survival-related internal variables, motivations and behaviour execution, it is possible to infer whether a particular object in the environment is suited to a particular interaction. This actually corresponds to learning information about the objects' potential for action, that is learning affordance concepts in the sense described above.

In Table 4.2 we summarise the landscape of different concept kinds that have been covered (taxonomic, goal-derived and affordance concepts), mapping each one to the dimensions of variation that have been mentioned while characterising it.

With respect to the learning process, the acquisition of affordance concepts follows a similar pattern as the acquisition of goal-derived concepts. In both cases, clusters of exemplars are not available before the concept and its corresponding category are formed; instead, a prior formation of clustering criteria is required. In the case of affordances, the subject does not know in advance whether or not a certain exemplar is useful (or useless) in a given situation, so the first step of the learning process is the formation of a notion of usefulness, which constitutes the functional aspect of the affordance concept; this notion is then used to judge in any particular situation whether an encountered exemplar is or is not a good candidate for that particular notion of usefulness. In the case of goal-derived concepts, the subject infers from previous knowledge a set of increasingly refined characteristics that exemplars should have to optimally serve the goal at hand. Both affordance and goal-derived concepts can therefore be opposed to taxonomic concepts, which are exemplar-based. On the other hand, affordance concepts and goal-derived concepts are distinct with respect to the intermediate clustering step that is involved: goal-derived concepts require a process of conceptual combination, while in the case of affordance concepts the formation of the notion of usefulness is grounded on the actual and direct interaction with the environment, and not on some previously established knowledge of other concepts.

Goal-derived concepts and affordance concepts are also similar insofar as they can be

	Are clusters of exemplars available prior to learning?	What clustering criteria are at work?	Are the concepts serving goals?	Is previous conceptual knowledge required?
Taxonomic concepts	**YES**	**Central tendency** (based on world structure and kind-syndrome)	**NO**	**NO**
Goal-derived concepts	**NO**	**Proximity to ideals** (based on concept combination)	**Short-term and ad hoc**	**YES**
Affordance concepts	**NO**	**Proximity to ideals** (based on affordance fit)	**Long-term and fundamental**	**NO**

Table 4.2: Summary of concept kinds.

said to serve plans to achieve goals, rather than mirroring and representing the correlational structure of the environment, as it is the case for taxonomic concepts. However, the types of goals involved by goal-derived concepts are usually short-term and *ad hoc*, resulting in the corresponding concepts being derived and processed on the fly, and then dismissed. On the other hand, the goals served by affordance concepts are more fundamental in the sense that they correspond to affordance fits serving functions of biological value to the subject (e.g. survival-related functions).

To conclude, we can say that affordance concepts and goal-derived concepts can be grouped together as concepts that are learnt in a mediated way (clusters of exemplars are not available before the learning process; instead, a prior formation of clustering criteria is required) and as such are opposed to taxonomic concepts, which are exemplar-based. On the other hand, affordance concepts and taxonomic concepts are somehow more primitive than goal-derived concepts, in the sense that their acquisition process does not rely on previous conceptual knowledge, rather it involves a direct interaction with the environment.

4.3 Summary

This Chapter was devoted to a thorough discussion of the notion of affordance. In Section 4.1 we have provided an overview of Gibson's seminal ideas on affordances and a characterisation of the notion of affordance based on those ideas. Our proposal is to consider an affordance concept to be a concept denoting a category of objects that are relevant for an agent either because they can be used to achieve the agent's goals, or because they prevent such an achievement.

We have then addressed, in Section 4.2, the issue of whether there are different kinds of concepts. Not much work has been done in accounting for different kinds of concepts, even though the literature suggests that concepts can be grouped together according to different criteria. Linguistic and developmental psychology data suggests a grouping based on the structure of concepts; this means that kinds of concepts are individuated on the basis of the structure of their instances, which would vary from concept to concept. Evidence from the fields of psycholinguistics and language development grant a distinction between object concepts, based on the clustering of perceptual features, and relation concepts, based on the clustering of relational features. With respect to another proposed distinction, opposing natural kind concepts to artifact concepts, the claim that the two constitute separate domains has been reduced by Ahn's experiments, showing that the centrality of a feature to a category does not depend on the kind of feature but rather on the causal role that the feature plays (relative to the other features characteristic of the category).

We then examined the distinction opposing taxonomic concepts and goal-derived concepts. In this case the distinction is based on how concepts are processed, rather than on structural differences. Goal-derived concepts are acquired through conceptual combination. This learning process differs from the exemplar-based acquisition of taxonomic concepts because the new goal-based concepts are derived by manipulating existing knowledge in memory, rather than experiencing with already available sets of exemplars. Affordance concepts are similar to goal-derived concepts, in the sense that they are also learnt in a mediated way. Instead of conceptual combination to optimize a plan, however, affordance concepts rely on a direct interaction with the environment to establish appropriate clustering criteria.

Chapter 5

An architecture supporting the autonomous formation of affordance concepts

5.1 From a fixed behaviour to an adaptive one: exploring a neighbourhood in design space

In the previous chapters we have introduced a set of ideas spanning from semantics to psychology and machine learning. It is now time to put those different ideas together into a single, unified picture. As we stated in the beginning, we are interested in individuating and describing the design principles that underlie a cognitive architecture supporting the ability to autonomously acquire concepts. In Chapter 4 we saw how affordance concepts constitute a particular type of concepts, that can be distinguished from other types on the basis of the requirements put on the processing mechanisms for their acquisition: it is affordance concepts that we focus our attention on.

Learning affordance concepts is part of the process of learning a model of the world that an agent inhabits, as we explained in Section 3.2.1, where we also described the concrete scenario that we adopted to develop our prototype architecture. Affordance concepts prompt the agent with part of an ontology for its model of the environment; more precisely, affordance concepts are internal representations of categories of things that are relevant to the agent, because either those things enable or they prevent the agent from doing something, in accordance with Gibson's analysis. For example, if we consider the class of things that can be eaten or that enable sitting, then the agent's internal representations of these classes, provided the agent has them, are affordance concepts.

The general architecture that supports the learning of a model of the world is represented in Figure 5.1. We distinguish between two control mechanisms, the innate control unit and the acquired control unit, and the idea is that the acquisition of a world model causes the switching from one control mechanism to the other. Furthermore, whenever the world model becomes inappropriate (as assessed by an optional self-reflective procedure at the meta-management level) control should be given back to the innate mechanism.

The architecture for learning affordance concepts described in Chapter 4 pages 77-78 is a special case of this general architecture. To prove our claim we discuss a series of successive steps of the possible transformation of an agent's architecture, that show how an innate fixed

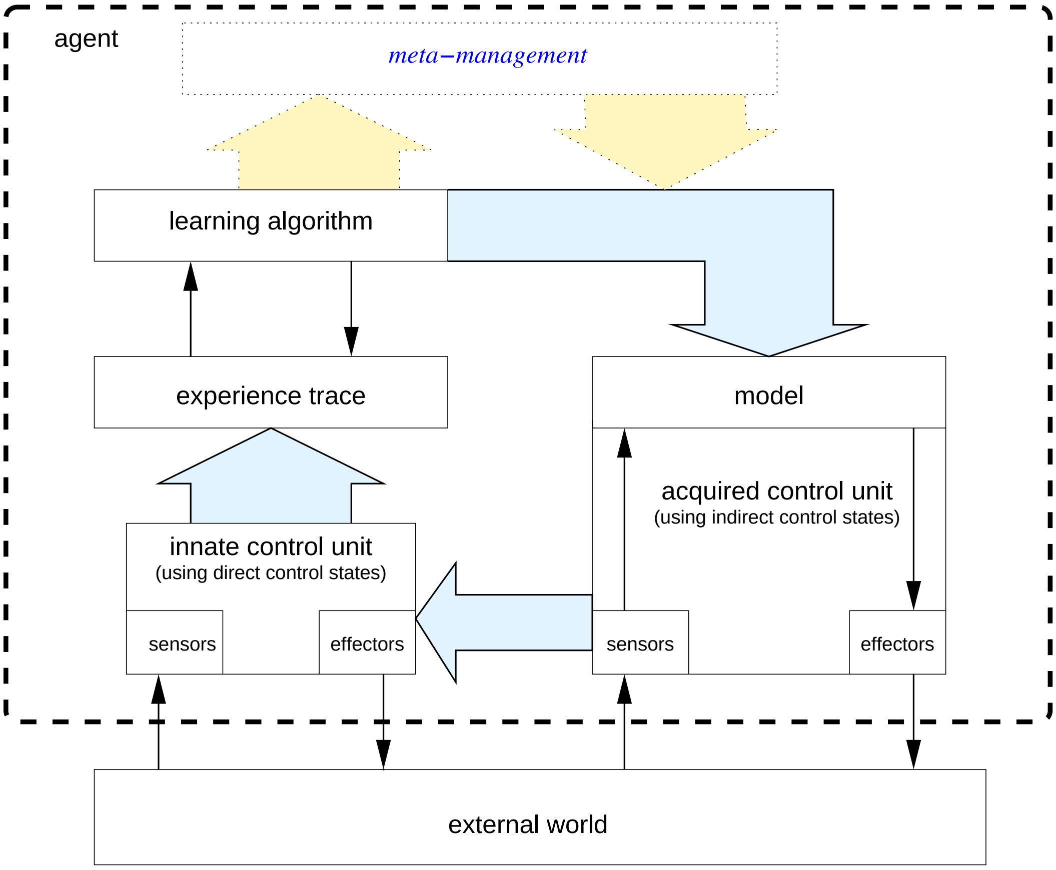


Figure 5.1: General architecture for the acquisition of a world model.

behaviour can be transformed (e.g. by evolution) into an innate adaptive behaviour.

5.1.1 A basic architecture for an agent with innate fixed behaviour

To start with, we discuss the architecture of an agent exhibiting an innate fixed behaviour (we refer to this as the IFB-architecture) allowing it to survive in a friendly environment. Before detailing the IFB-architecture, we briefly recall the scenario that we have been exploring (see Section 3.2): it consists of an agent autonomously moving around and acting in an environment where different kinds of objects exist (the situation is depicted in Figure 3.3). The objects in the environment have properties (among others, a location and a size) and over time their sensible qualities may change; they are simple, in the sense that they have no parts, do not move, and do not interact between themselves. These simplifications have been motivated by the attempt to keep the physics of the world as simple as possible, to start with, in order to be able to concentrate on the architecture of the agent. The agent can sense the objects in its environment, and some objects may change its internal state when acted upon. For this to be the case, the setup of the environment involves the generation of the object instances that populate the environment and their grouping into 'causal classes', i.e. classes of objects for which causal laws exist in the world. In particular, the physical interactions among agents and objects conform to such laws. The four object classes that have been considered in our example are: food items, danger items, treasure store items, and neutral items. To summarise, the main steps involved in the setup of the environment consist of:

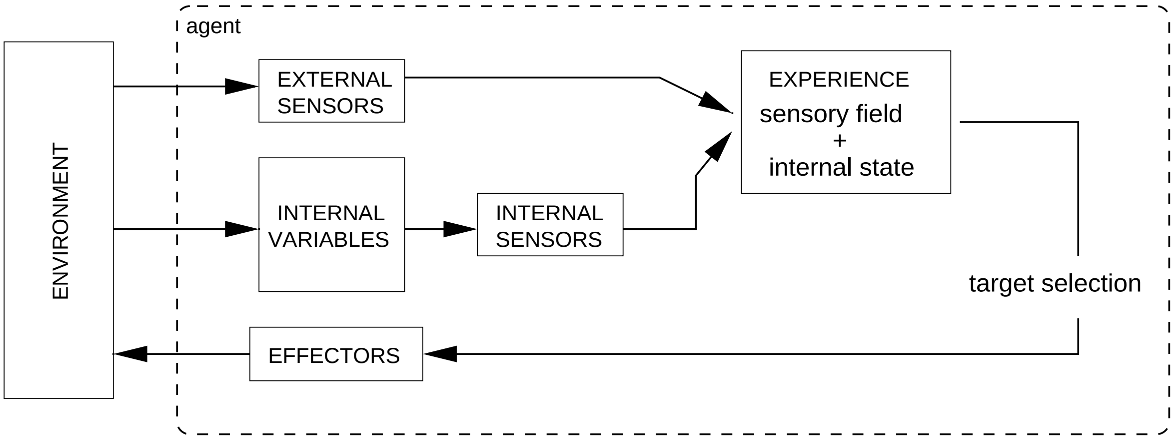


Figure 5.2: The high-level architecture of an agent with innate fixed behaviour.

- determining the logical possibilities, by setting the objects' attributes and their possible values;
- determining the physical possibilities, by choosing the causal classes;
- setting the initial conditions, that is the number of object instances, the class proportions and the regional characteristics; for example, this might involve setting one region as more "friendly" than another, because object classes on which the agent relies for its survival (that is, resources) are abundant, while object classes that constitute a danger are scarce.

In this environment, an agent moves around with the ability to act upon objects. The agent has a set of internal variables, namely for its energy, happiness, and pain level, whose values are gathered by a set of internal sensors; it also has an internal sensor for motion. Different combinations of the values of these internal variables determine different internal states of the agent. Some of these states are motivational states that represent urges to act based on bodily needs related to survival. For example, if the level of pain is above a critical threshold then the agent is hurt and urged to move away; if the energy level is above the death threshold but below another critical threshold then the agent is hungry and urged to eat; if the agent has not been moving for a certain time then the agent becomes bored and it is urged to move. The agent's behaviour is therefore caused by its internal states and its actual movements are based also on an evaluation of its own position with respect to other objects in its perceptual field.

With respect to survival, the agent's goals are to keep its internal level of energy above a certain threshold, and its internal level of pain below a certain threshold. If both the level of pain and the level of energy are on the right side of their critical threshold then the agent's default goal is to increase its internal level of happiness.

In order to achieve its goals, the agent can perform one of the following external actions: move towards or away from an object, collect a treasure (from an object), eat an object. The agent gathers information from the environment through a set of external sensors. As we said, the object classes that have been considered in our example are: food items, danger items, treasure store items, and neutral items. If performed on the appropriate objects (treasure stores and food items respectively), collecting treasure increases the level of happiness, while eating increases the level of energy. If an action is performed on a danger item, the agent is hurt.

The IFB-architecture is depicted in Figure 5.2, and it simply ensures that the agent is well-coupled with the environment: it does not contain any implicit model of the objective relevant classes (those determined by the world laws) but it contains an implicit model of how the agent's motivational states and actions are linked together to produce a coherent behaviour. The IFB-architecture is therefore a control architecture that implements a motivational structure that keeps the agent alive, without the agent "knowing" about the world. It makes use of simple object detectors (and not object class detectors) and an innate association between internal states and action selection procedures.

The IFB-architecture determines a behaviour that is completely driven by the environment, and is purely reactive, in the sense of being incapable of deliberative reasoning; it uses no memory or state produced by previous processing, and can be seen just as an input-output mapping. It is to be distinguished from the architecture that controls the construction of a model of the environment, and from the architecture that controls the agent's behaviour as based on the acquired model of the environment.

At the beginning of each processing cycle, an agent controlled by the IFB-architecture gets some information:

- on the external world, that is the distance and direction of the objects that are in its perceptual field;
- on itself, that is the level of its internal variables (for energy, happiness, and pain), and its own heading and speed.

First of all, the agent's internal state is evaluated. Being in a particular internal state motivates the agent to act, hence each state triggers an action to be performed upon an object. The agent must therefore select an object as its target, and approach it. The target is chosen among the objects in the perceptual field: if the perceptual field is empty, the agent simply moves forward, otherwise it chooses as its new target the object which is closest to it. The innate association between internal states and action is as follows. Whenever its internal state is not critical, the agent selects a target, approaches it and tries to collect treasure. When hungry, the agent selects a target, approaches it and try to eat it. When hurt, the agent seeks to avoid pain by moving away from the target; the state of being hurt thus works as an alarm, in the sense that it makes the agent stop whatever it was doing, and move away. Since the objects are modelled as relatively small compared to the size of the agent, the basic unit of motion is such that, after moving, the agent actually finds itself past the object at which it was standing, thus avoiding the need to model collision and avoidance of collision. Being bored means that the agent is at target and has not been moving for a while. The state of being bored is used to make sure that the agent does not stay forever at a particular target, but will move around and explore its environment.

The agent does not remember, at least in this particular architecture, if a particular object has already been visited or not, and in order to avoid vicious cycles (i.e. the agent going back and forwards between the two same objects) the agent has a perceptual field of a certain angle, and slowly turns on itself only when there is nothing right ahead. Once the agent has acted, the system manages the interactions between the agent and the world, as a result of the actions performed by the agent. Also the agent's internal variables (and memories) are updated.

The main processing steps for an agent controlled by the IFB-architecture can be thus summarised:

- get the values of the internal variables and get information on heading, speed, and number of cycles without motion;
- evaluate the internal state;
- set an action to be performed; for example: if hungry and not hurt, then set the action `eat` to be performed;
- get the values of the external sensors; this allows the agent to build its sensory field; we are here assuming that an initial form of interpretation of the sensory input has already occurred, namely the one that chunks the sensory field into foreground objects and background;
- if the sensory field is not empty, focus on an object, that is choose the target object;
- evaluate the current position with respect to the target; the previous step and this one correspond to the agent's interpretation of the sensory field in order to build a perceptual field;
- activate the effectors, either to move forward (if the sensory field is empty or the target is out of reach) or to perform the selected action upon the target; when the agent moves forward after being at target, it finds itself automatically past the target object, because objects are modelled as relatively small compared to the agent's size.

The interpretation of the external sensors input in order to create a perceptual field allows for a distinction between the target and all the other objects in the perceptual field; in other words, there is an innate concept of 'target item'. We are here assuming a distinction between a sensory field and a perceptual field, where a perceptual field is an interpreted sensory field, that is a sensory field that contains also information on the objects and possible relations between the objects and the agent.

The explicit storing of information about the internal state and the selected action is not necessary for the IFB-architecture. If such an option is chosen, then specific internal symbols representing these data are needed. In general, we follow a strategy of parsimony, according to which new symbols and data-structures should be introduced with care. So we propose that only the following situations should require them:

- to store information that could change before it is used again;
- to reduce the costs associated with direct sensing: accessing and manipulating the symbolic representation could be easier than producing each time the information through a perceptual process;
- to centralise all the information relevant to some further processing, so that it might be easier to access it.

5.1.2 Main characteristics of the IFB-architecture

Given the high-level design of the IFB-architecture, we can analyse its main characteristics from a cognitive point of view. An agent implementing such an architecture has an observable autonomous behaviour that allows it to survive and also to satisfy its further goal of being happy. Even though survival is dependent on certain environmental conditions, such as the fact that enough survival-related items (e.g. food items) can be reached by the agent given its initial internal resources (we refer to this as to the 'environment-friendliness'), the very same agent can survive in environments with different causal classes. This is so because the action-selection mechanism of the agent is dependent only on its internal states and does not rely on a previous knowledge of the environment. Making the agent's survival depend on its ability to maintain the stability of its survival-related internal variables is an idea that goes back to Ashby's notion of viability [Ashby, 1952], and other architectures drawing on the same principles have been proposed such as [Meyer, 1995] and [Cañamero, 1997]. The agent's behaviour is determined by an innate arbitration of tendencies, which is achieved through a hierarchisation of the agent's motivational internal states. The agent has innate knowledge of the associations between motivational internal states and actions (this is something like having instincts), but in order to select its actions it does not need to gather a lot of information on the environment.

If the environmental conditions are friendly, the only information about the objects in the perceptual field that has to be gathered by the agent in order to survive is the distance and direction of these objects relative to the agent's own position. However, most of the actions performed by the agent will not bring any benefit to it, because they are performed 'blindly' and most of the time upon objects that are inappropriate (for example, trying to eat a treasure store or a neutral item, rather than a food item). We can therefore speak of a low ratio benefits/actions for an agent controlled by the IFB-architecture.

Assuming that the evolutionary cost depends on the complexity of the architecture and assuming that there is a cost associated with the performance of an action and with the activation of an external sensor, the above characteristics can thus be summarised by saying that the IFB-architecture is:

- evolutionarily cheap (i.e. easy to produce, because architecturally quite simple);
- survival expensive;
- flexible but sensitive to 'environment-friendliness';
- sensing cheap.

5.1.3 Towards model formation: the ability to learn affordances

The IFB-architecture discussed in the previous section can be extended first to support the autonomous acquisition of a model of the affordances present in the environment, and then to obtain an agent with an adaptive behaviour which is driven by such a model, as depicted in Figure 5.4. The learning of the affordances provided by the environment is achieved with the introduction of a further set of associations (captured in terms of condition/action rules) relating the performance of an action to definite expectations — for example in terms of values of internal variables — about the result of that action. These associations enable the agent to judge whether or not the target object is a positive instance of the affordance concept

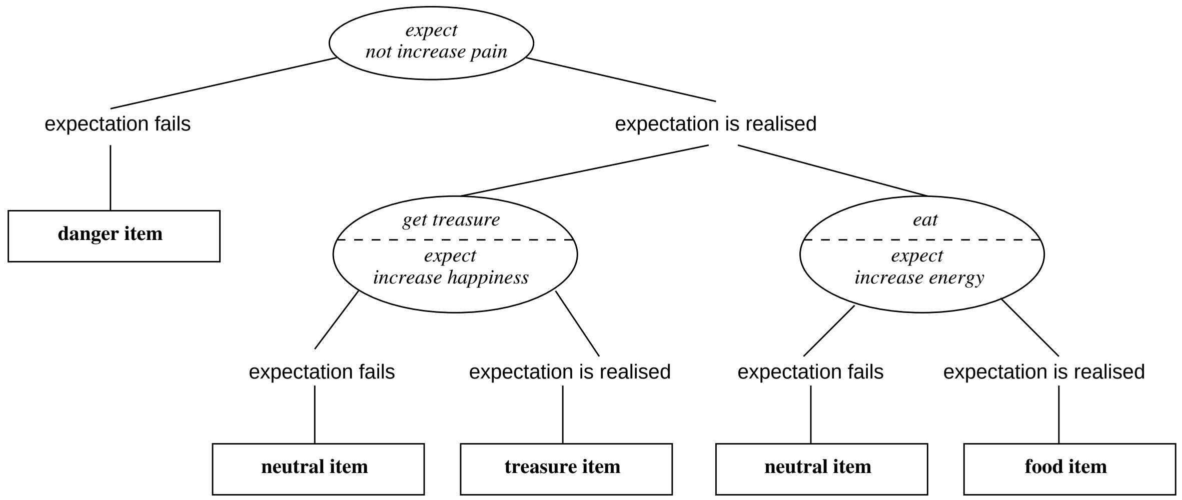


Figure 5.3: An example of functionally defined categories.

being learned. As we said, an affordance concept refers to objects in terms of perceivable possibilities for action. This means that any affordance concept can be defined in terms of the goal that the agent wants to achieve by performing a certain action, and the action chosen to achieve it; in particular the goal can be used as the definite content of an expectation on the results of performing the chosen action.

In Section 2.3, while discussing machine learning, we saw how two successive phases are involved in concept formation [Langley, 1996]. First of all, certain items have to be clustered together, and then a characterising description for the cluster must be sought. In a later stage, this description can be used to decide whether or not a new item is also to be considered a member of the cluster. One could argue that something like this description, that is searched for later on, should already be there in the first place, as the principle according to which the clustering operation is performed. The two however can be distinct: for example, the clustering could be done using a fast pattern-matching procedure relying on some superficial similarity (as when someone initially puts together the pieces of a puzzle because they have roughly the same surface colour patterns) and then the characterising description can be searched for at a more structured similarity level (like checking whether the colour patterns of the various pieces can fit together to form a connected spatial region, such as the region representing part of the tulip field that the puzzle depicts).

The two phases of clustering and characterisation can be mapped into two distinct aspects of affordances: on the one hand we have the *functional* aspect of the affordance, the fact that an affordance enables grouping together things according to their potential use; while on the other hand we have the *discriminating* aspect of the affordance, the fact that things serving the same use can sometime be recognised and grouped together also because they are similar in some other respect (for example, graspable things can be similar in shape). It is important to notice that whenever such a similarity relation is accessible *before acting* on one object, then its recognition could be used to select and guide action.

On the basis of this distinction we propose that an internal representation standing for an affordance concept should be a two-tiered structure, with one component capturing the functional aspect of the affordance, and the other one capturing the discriminating aspect of the affordance.

Once we have an idea about how affordances should be represented, the next step is to work out a strategy that an agent can use to acquire them. Our focus is not on what kind of learning techniques (e.g. reinforcement learning) could achieve the desired results. Instead, we are interested in presenting a picture at the system-level of an architecture embedding one or more sub-modules (such as learning and/or controlling modules) that, if adequately orchestrated, can support the acquisition and use of complex internal representations such as affordance concepts.

Assume the agent has no initial concepts according to which external objects can be categorised, but has a set of condition/action rules relating actions and expectations about the result of performing those actions. The idea is that any such rule could be exploited to perform the initial clustering of external sensor data, so that later on a characterisation algorithm could be run on those clusters to produce a set of discrimination rules. The agent is therefore provided with innate functional concepts denoting the relevant categories of objects in terms of associations between actions and expectations. An example of such a definition for the scenario previously discussed is represented in Figure 5.3.

The categories denoted by these functional concepts can be referred to by a set of internal symbols which are generated and then stored in the agent's memory when expectations are evaluated. For each expectation, a symbol is generated that will later be used to refer to objects that make that expectation true, when acted upon. As an example, assume the agent expects the value of its internal variable $energy$ to increase; then a symbol is generated to refer to objects fulfilling that expectation.

This however is not enough to enable the agent to discriminate between objects. What is needed is a set of identification rules, establishing what an instance of a certain category looks like, in terms of the instance's perceivable features. The set of identification rules is the actual result of the *learning module*. Learning is incremental, which means that an hypothesis is generated as soon as an experience trace is available, where an experience trace contains data on the following steps of the agent's cycle: the check of the internal state, the motivation to act, the setting of the expectations, the action performance, and finally the evaluation of the internal state against the expectations.

Here is a detailed example: In cycle c_n the agent, who has no concepts to categorise the objects it encounters, is at target object O, performs the action `eat` and expects the value of its internal sensor for energy to increase. Now, it happens that O is actually food, and therefore the interaction actually causes the agent's energy to increase. In cycle c_{n+1} the agent can therefore evaluate its expectations against the actual modifications of the values of its internal sensors: in this case, the judgement is that expectations have been fulfilled. Assume now that in cycle c_n, besides acting, the agent has also been storing the values of its external sensors as activated by the presence of the object O, for example that it is red, sweet and soft; then in cycle c_{n+1} the judgement about the fulfillment of the expectations on eating can be used to store some further information about O, namely that it is a positive example of the concept THINGS THAT AFFORD EATING. Had the expectations failed, the agent could have stored the information that O is a negative example of that same concept.

As time goes by, an agent like the one considered in the example will develop two related internal databases, one storing the condition/action rules (where the condition part is 'performing an action' and the action part is 'setting an expectation') and the other storing external sensor data labelled as positive or negative. The keys relating the two databases can be seen as the internal designators of the agent's concepts (we can think of them as symbols in the agent's "mentalese" to denote affordance concepts, i.e. *affordance-labels*). In the service of cognitive economy, the database containing the external sensor data can then

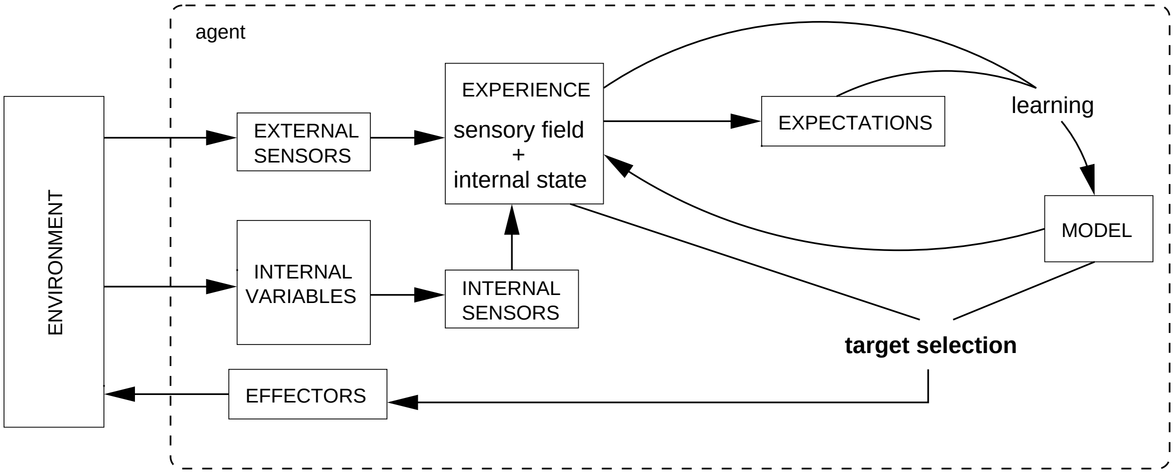


Figure 5.4: The high-level architecture of an agent that acquires a model of its environment, and uses it to adapt to the environment. Adaptation (as behaviour change) is noticeable at the level of the plan selection.

be processed using a supervised learning algorithm, so that, for every class of positive and negative items associated to the same key, only a characteristic description is actually stored and kept in memory (rather than the whole list of items). The formation of these two related databases however does not yet cover the whole concept formation process, because simply the presence of the two databases does not make any difference to the agent's course of action. So the next design modification to be considered is the use of the database information to change, whenever possible, the agent's behaviour.

5.2 An architecture for an agent with adaptive behaviour

As discussed in the previous Section, the IFB-architecture can be extended to support the autonomous acquisition of affordance concepts. With a further extension it is possible to obtain an agent with an adaptive behaviour (we refer to this as the AB-architecture). The AB-architecture is characterised by the presence of a module that controls the agent's behaviour in a way that is driven by the affordances that have been learned: the affordance concepts are used by the agent during its active phase, and more precisely when the target object to be approached and acted upon is chosen.

As we said, affordance-labels are associated both to the condition/action rules linking actions to expectations, and to the characterising descriptions of affordances, so the idea has been to provide the agent's architecture with the following control mechanism:

- whenever the motivation to perform a certain action is triggered, an internal check is also performed to see whether an affordance-label is associated with that action;
- if so, a further check is performed to see whether a non-empty characterising description is also associated to the affordance-label;
- if so, the description is used in the procedure to select the target object upon which to perform the intended action (e.g. approach objects that match the description; or avoid

objects that match the description), otherwise the agent relies on its default and innate target-selecting procedure.

5.2.1 Overview of the AB-architecture

The behaviour produced by the AB-architecture integrates the behaviour of the simpler IFB-architecture with both a module controlling the construction of a model of the environment, as described in Section 5.1.3, and a module that controls the agent's behaviour as described above. As with the IFB-architecture, the agent's processing cycle starts with the collection of information on the external world and on itself, by means of its external and internal sensors. More precisely, the agent gets information:

- on the external world, that is the feature values, distance and direction of the objects that are in its sensory field;
- on itself, that is the level of its internal variables (for energy, happiness, and pain), its own heading and speed, how long it has been without moving.

Then an evaluation of the agent's internal state takes place, but this time such an evaluation is accompanied by a monitoring of the internal sensors changes: whether they increased, decreased or remained constant with respect to their values in the previous cycle.[1]

As with the IFB-architecture, being in a particular internal state motivates the agent to act; this time, however, we have a further step which is the setting of a definite expectation on the results of the action. The agent has a default expectation on the level of pain not to increase. When the internal state is not critical, the agent has a motivation to collect treasure, and expects its level of happiness to increase. When hungry, the agent has a motivation to eat and expects its energy level to increase. When hurt or bored, the agent simply moves away from the target. This means that the action triggered by being hurt and being bored is the same, that is, moving away. However, when the agent moves away because it was hurt, it also has an expectation concerning the decrease of its level of pain.

Once the action to be performed has been triggered, the agent goes through the usual steps of selecting a target object, possibly approaching it, and then acting upon it. The novelty of the AB-architecture has to do with the existence of definite expectations on the results of the action to be performed, which can fail or be realised. In the next cycle these expectations are evaluated against the modifications of the values of the internal sensors, thus determining the information that the agent stores about the object it has acted upon: that is, whether the object is a positive or negative example of the functional concept the agent is trying to learn. This information constitute the experience data used by the learning algorithm to generate identification rules for that functional concept.

Once these rules are available, they can be exploited to modify the agent's innate mechanism to select a target object. To select a target, the agent examines its perceptual field: if it is empty, the agent simply moves forward, otherwise it selects one of the objects in view as the target object, i.e. the object to approach and act upon. At the beginning, since the agent does not remember if a particular object has already been targeted or not, and since all the objects look the same to it, it always chooses as its target the object which is closest to it. Afterward, the learning of affordance concepts affects the target selection behaviour, and

[1] Just as a remainder, the possible internal states are `dead, hurt, hungry, bored` or `OK`, while the default internal state is `OK`.

more precisely the agent will avoid objects that look like danger items to it, and, depending on its motivation, approach objects that look like food items or treasure items to it.

As an example, let's assume that the action to be performed is `get_treasure` and that the agent knows that the relevant objects for such an action are those falling under the concept TREASURE. When scanning its sensory field, the agent is faced with one of the following situations:

- the perceptual field is empty;
- the perceptual field is not empty, but no identification rule is available for the concept TREASURE nor for the concept DANGER;
- the perceptual field is not empty, there is at least one identification rule for the concept TREASURE, but there is no identification rule for the concept DANGER;
- the perceptual field is not empty, no identification rule is available for the concept TREASURE but at least one identification rule is available for the concept DANGER;
- the perceptual field is not empty, and there is at least one identification rule for the concept TREASURE and for the concept DANGER.

Within the AB-architecture, the mechanism controlling the agent's behaviour is such that, when no identification rules are available, the agent selects the nearest object in its sensory field as its target; when identification rules for the concept DANGER are available, items matching those rules are avoided; and when identification rules for the concept TREASURE are available, items matching those rules are considered potential targets. This is summarised in table 5.1 (where 'i-rules' means identification rules).

Sensory field state	**Model state**	**Behaviour (target selection)**
empty	N/A	move on
not empty	no i-rules for TREASURE no i-rules for DANGER	make nearest object your target
not empty	no i-rules for TREASURE i-rules for DANGER = $\{i_1, \ldots, i_n\}$	make nearest object that is not a DANGER your target
not empty	i-rules for TREASURE = $\{i_1, \ldots, i_m\}$ no i-rules for DANGER	make nearest object that is a TREASURE your target
not empty	i-rules for TREASURE = $\{i_1, \ldots, i_m\}$ i-rules for DANGER = $\{i_1, \ldots, i_n\}$	make nearest object that is a TREASURE your target, avoiding any object that is a DANGER

Table 5.1: Overview of the AB-architecture controlling mechanism for target selection, when the action to be performed is `get_treasure`.

What remains to be discussed is what happens whenever there is at least one identification rule prescribing what kind of objects should be selected as a target, but nothing in the sensory field matches the rule prescriptions. Since in our architecture prototype the identification rules are learned through a search of the hypotheses space that goes from specific to general, if the agent sticks to its initial rules and always looks for a complete match, it will never have the occasion to have new experiences from which to learn and improve its model. This means that, whenever the prescriptions of the rule are not matched, the agent should better approach anyway something in its sensory field, and act upon it. The behaviour that we chose to implement is therefore to make the agent approach the nearest object in the perceptual field, avoiding dangerous items. We acknowledge however that the choice between selecting a target anyway and continue moving until something appears in the sensory field that is exactly as the model prescribes could be made dependent on the agent's beliefs about the quality of its model: if the model is still under construction, the first behaviour should be chosen because it allows the agent experience with unknown objects; when the model is considered (by the agent) to be a good and reliable one, then the second behaviour should be chosen, because at that point if something does not match a concept description it is very likely that the thing into question is not part of that concept extension. Moreover, the requirement of a strict match could be relaxed, and a notion of similarity introduced.[2]

Once the target has been chosen, the agent evaluates its position with respect to it and then acts if at target, otherwise moves forward in the direction of the target, approaching it. After that, the interactions between the agent and the world, as a result of the actions performed by the agent, are managed and so is the agent's short term memory. In particular, the old values of the internal sensors are remembered.

The behaviour produced by the AB-architecture constitutes a further development of the behaviour of the IFB-architecture and the main processing steps of an agent controlled by such an architecture can be summarised as follows (we assume that the associations between actions and expectations are built-in):

- get the values of the internal variables and get information on heading, speed, and number of cycles without motion;
- build the internal model; this comprises four steps:
    - monitor changes in the internal variables;
    - compare the actual internal changes with the expected ones (i.e. the expectations on the action results set on the previous cycle);
    - store in the learning DB information about the object that was acted upon in the previous cycle;
    - apply the learning algorithm to the learning DB and produce a multi-decision list;
- evaluate internal state;
- set an action to be performed;
- if the model is not empty, plan what kind of object should be a target;
- get the values of the external sensors;
- if the sensory field is not empty, and if the model is not empty, then categorise the sensory field;
- choose the target;

[2]Whether or not an object instance matches (is subsumed by) a given concept is determined by the *matching algorithm* being used. In particular, an instance can be subsumed *only if it matches exactly* the model prescriptions (strict or narrow matching algorithm) or *also if the match is only partial* (similarity algorithm).

- evaluate the current position with respect to the target;
- activate the effectors, either to move forward (if the sensory field is empty or the target is out of reach) or to perform the selected action upon the target;
- set expectations on the result of the action;
- manage short term memory.

5.2.2 Main characteristics of the AB-architecture

For an agent architecture embedding the control mechanism just described we can definitely speak of 'ability to acquire affordance concepts' and, after some time, also of 'presence of affordance concepts' and 'ability to use affordance concepts'. On the one hand, by making reference to the agent's architecture these abilities have been made precise, as suggested in [Sloman and Scheutz, 2002]. On the other hand, to get the analysis started and to reach a provisional end-point, certain simplifications and initial assumptions have been made: we limited ourselves to the consideration of primitive, unanalysable actions, and we assumed the presence right from the start of the associations between initial internal state and action, and between action and expected final internal state. This prevents the acquisition of completely new affordances, so that we should say that affordances are learned in the sense of made explicit: they are already present in a sort of hard-wired way, but a mechanism is present that allows them to be re-presented [Karmiloff-Smith, 1992] in a new representational form, that allows some new form of processing. We also assumed the innateness of the notion of external object: to start with, the agent has no concepts to categorise the objects it meets, but it knows when it is in presence of an object.

Given the high-level design of an AB-architecture, we can analyse its main characteristics from a cognitive point of view. The main difference with respect to the IFB-architecture previously discussed concerns the ability to acquire and store a model of the environment, and then use this model to modify the agent's innate behaviour. In particular, the agent learns a set of affordance concepts and identification rules (which specific affordance concepts are learned depends on which are the innate associations between actions and expectations that the agent has) that are used to interpret the sensory field and to guide the behaviour: in particular, to prevent the performance of actions that do not bring any benefit to the agent. In the long run, this means that most of the actions performed actually bring some benefit to the agent. Moreover, the agent can survive in environments with different causal classes, even if the initial conditions are very unfriendly. More precisely, if the initial conditions are very unfriendly, there is more chance to survive if the initial experiences are relevant ones and are not misleading, because this would prompt the acquisition of an appropriate model at an early stage.

If the acquired model is used to influence the behaviour (i.e. which object is chosen to be the next target), but not the process of gathering information on the external environment (e.g. to prevent the gathering of irrelevant sensory information), then the cost of sensing, that is the cost of building a sensory field and a perceptual field, can still be very high.

With respect to what can be characterised as innate knowledge, an agent implementing the AB-architecture has built-in associations between motivational internal states and actions and between actions and expectations (where expectations have definite content in terms of internal state modifications). The agent also has an innate learning mechanism and an innate concept of 'target item', used to interpret the sensory field. However, it has no innate knowledge of the environment.

The above characteristics can be summarised saying that, compared with the IFB-architecture discussed above, the AB-architecture is:

- evolutionarily more expensive;
- assuming that there is a cost associated with the performance of an action, provided with a higher ratio benefits/actions, hence survival cheaper;
- flexible but sensitive to initial 'training';
- sensing expensive;
- supporting changes of behaviour over time; changes are observable in terms of the choice of the target object and the avoidance of dangerous items.

5.2.3 Details of the learning module and of the experience data

Our work was motivated by the double aim of clarifying the notion of affordance by giving such notion a more precise content in terms of information processing mechanism, and exploring cognitive behaviours that lead to the acquisition of affordance concepts.

In order to test the ideas presented in the previous sections, we actually implemented an agent controlled by an instance of the AB-architecture using Sim_agent, which is a Pop11-based toolkit for agent development. Sim_agent can be used to develop agents and objects that interact in a simulated environment, and its great flexibility allows for the exploration of different architectures and design options for the agents and objects.[3] The top-level feature of Sim_Agent is its *scheduler*, which controls the execution of the agents, in pseudo-parallel if they are all on one machine. An agent execution is a sequence of sense-process-act cycles, where sensing processing and acting may all be internal to the agent. Each agent is run concurrently with other agents by the scheduler, whose execution is also divided up into cycles. In one scheduler-cycle, each agent is allocated a time-slice, during which the agent executes a discrete number of its agent-cycles (it could also be 0.)

For each agent, the agent architecture is encoded in a set of rules called a *rulesystem* which is dived up into modules called *rulesets*. The rulesystem encoding the agent is then run by a *rule-interpreter*, which runs each ruleset in the order specified in the agent architecture definition. The running of an agent-cycle can be traced, and the toolkit provides many tracing facilities. Some of these allow an agent to trace its own behaviour, as required for meta-management [Sloman, 1998a].

The rulesystem encoding the AB-architecture is made available in Appendix A. In particular, the procedures `mdl` and `isc` together implement the learning module of the architecture, where `mdl` produces a multi-class decision list by calling recursively the actual learning algorithm, `isc`, which performs an incremental separate and conquer search for disjunctive concepts.

[3]For an overview of the toolkit see [Sloman and Logan, 1999], [Sloman and Poli, 1996] and the related teach and help files which are available within the Poplog system. Further documentation is available as part of the Cognition and Affect Project at `ftp://ftp.cs.bham.ac.uk/pub/group/cog_affect/`.

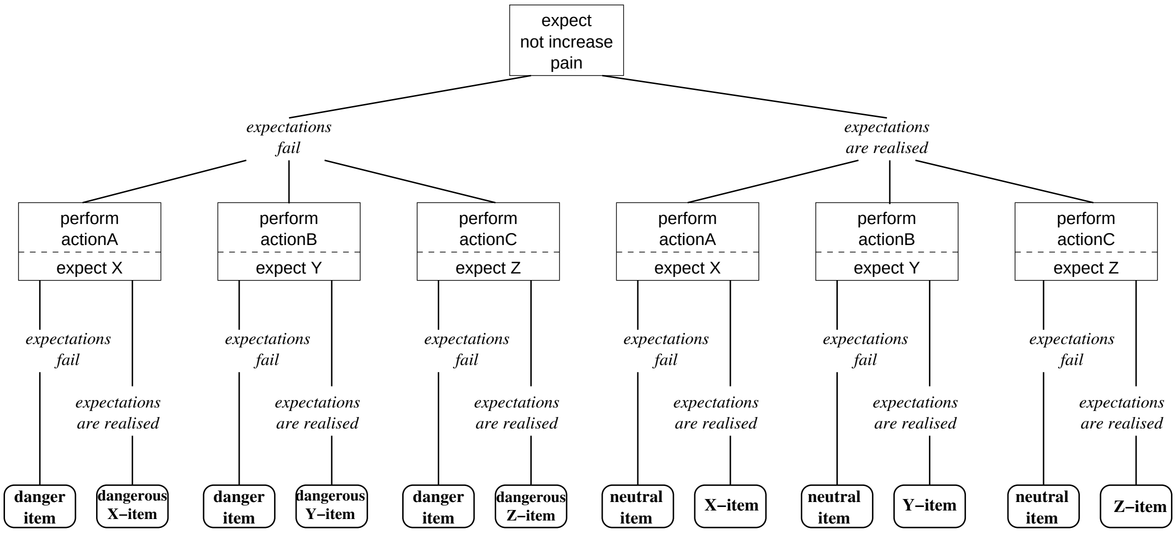


Figure 5.5: The focused-while-acting learning strategy.

As we said, an agent controlled by the AB-architecture learns something about an object after it has acted upon it. With this picture in mind, there are two main strategies that the agent can follow to obtain experience data:

- first strategy: try to learn as much as possible about an object, when in position to act upon it;
- second strategy: just perform the action that is required to satisfy the current goal, and if the expectations on the result of the action fail, move to another object.

If an agent follows the first strategy we can say that the agent is being *explorative*, in the sense that the primary goal that it is trying to achieve is the goal of collecting as much information as possible on the current object. For example, the agent could be hungry (but not 'critically' hungry) and therefore approach an object in order to eat it; however, should it turn out that the object into question is not Food, the agent would continue trying all its possible actions on the object, in order to discover whether the object is relevant for it anyway.

If an agent follows the second strategy we can say that the agent is being *focused while acting*: it tries to achieve its current goal, and in case of failure moves on to another object. Of course the two strategies can be put together, so that the agent follows one or another depending on the circumstances: whether or not the goal is a critical one, whether or not the agent has time to do some further investigations, etc.

The difference between the two strategies concerns the learning outcome, and can be shown in terms of which categories can be formed. In Figures 5.5 and 5.6 we have represented these differences for an agent who has three possible actions plus an initial internal check on whether or not it has been hurt. Notice that when an agent uses a focused-while-acting learning strategy things end up being classified "from a certain perspective" in the sense that, for example, the first time it is acted upon, object1 can be known to be a treasure but not an energy source, even though it happens to be both; or it can be thought to be potentially irrelevant, since not a treasure, even though it happens to be an energy source. Also, the agent does not know right from the beginning that classes may overlap and overlapping

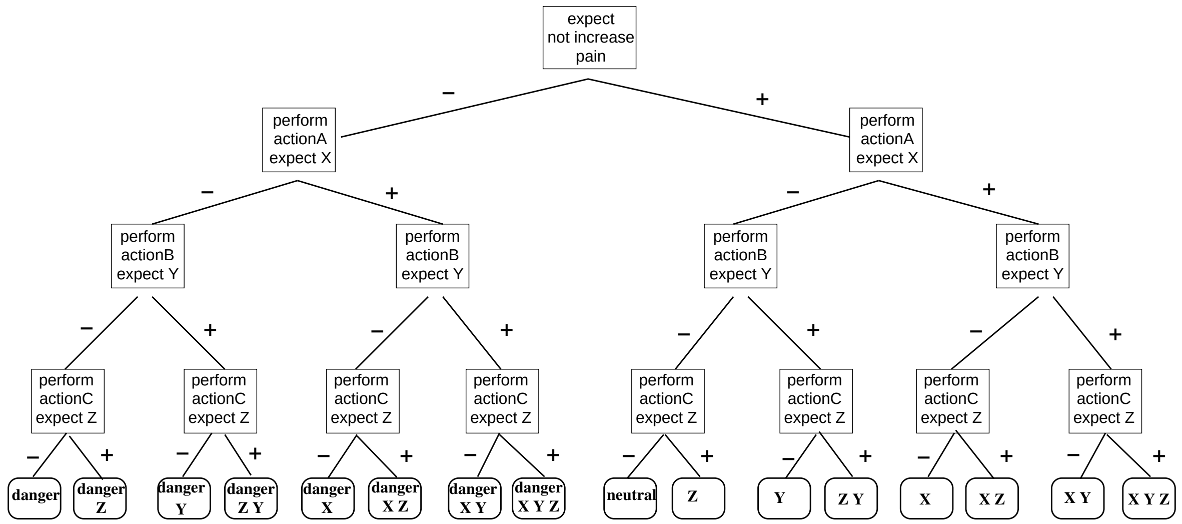


Figure 5.6: The explorative learning strategy. The $-$ symbol means that expectations have failed, while the $+$ symbol means that expectations have been realised.

is something that must be discovered.[4]

In our actual implementation, we have opted for the second strategy (focused learning), because it seemed cognitively more plausible for an agent whose survival depends on its ability to maintain the stability of its survival-related internal variables. Moreover, the advantage of the explorative strategy in terms of ruling out classification mistakes is only apparent in a world which is not homogeneous. In general, we can say that an agent is facing a *classification mistake* whenever it happens to be in a situation where it recognises an object as an instance of the class α-items (and only of that class) and then discovers, by acting upon it, that the object is an instance of the class β-items. The mistake can have two origins: i) the concept that the agent has for denoting α-items is too general, meaning that the object actually is not an α-item but is a β-item; ii) the concept that the agent has for denoting β-items is too narrow, meaning that in fact the two classes overlap, and hence the object is both an α-item and a β-item. In both cases the agent's conceptual structure should be revised. Even though the first strategy seems to rule out these mistakes, this is not the case, since the agent might have explored a part of the world which contains objects of a more specific kind with respect to other parts of the world (this may lead it to form concepts that are too general) and the agent may learn new actions (this may lead it to revise certain concepts as too narrow).

Let's now turn to the actual concept formation process. As we said, the agent can learn something about an object when it is at the object and has acted upon it, expecting a certain state of affairs to occur as a result of the action performed. For example, the agent eats the object and expects the value of its internal sensor for energy to increase. If the agent's expectations are realised, then the agent should store a data structure conveying more or less the following information: the sensory features I am now experiencing qualify an object that is a positive instance of the class Food.

In Section 5.1.3 we distinguished between the functional aspect of an affordance and its discriminating aspect, and we said that an agent controlled by an instance of the AB-architecture starts with an innate knowledge of the functional aspects of the affordance con-

[4] This last problem is overcome if second order learning is allowed. This means running again the learning algorithm, but with training data coming from the model, rather then from the experience trace.

cepts that must be learnt. In our implementation this innate bit of knowledge is captured by enabling the agent to access a three-fold data-structure containing information on one action, one expectation and one predicate symbol, associated in such a way as to convey the followings: i) the expectation is an expectation on the result of that particular action, and ii) the predicate symbol is the symbol that has to be used to denote the class of objects that realise the expectation, when acted upon with that particular action.

In the prototype scenario that we have been exploring (the Treasure scenario) the agent as 3 such data-structures; represented as triples, they are:

- $< [expect\ increase\ energy],\ [do\ eat],\ [Food] >$
- $< [expect\ increase\ happiness],\ [do\ get\_treasure],\ [Treasure] >$
- $< [expect\ not\_increase\ pain],\ [\,],\ [Danger] >$

In particular, the last item captures the fact that the default expectation of the agent is an expectation about not being hurt.

The expectation data-structure stands for a combination of internal settings (a combination of values for the agent's internal sensors); the action data-structure stands for the activation of one particular effector; the predicate symbol stands at the same time for a functionally defined affordance concept and a class (initially empty) of external objects.

The mechanism that we exploited to produce the three-fold data-structures mentioned above is as follows: first of all, the action planned to be performed is associated with the expectations on its results and the expectations are associated with a predicate symbol, then the action planned to be performed is associated with the predicate symbol by means of the common link with the expectations.

The three-fold data-structures $< expectations,\ action,\ predicate\ symbol >$ provide the agent with the knowledge required to decide what kind of information should be stored about the target object. Such information constitutes the proper learning database of the agent, and it contains data such as "the conjunction of values small, sweet and red is a negative instance of the concept of Danger" or "the conjunction of values small, sweet and red is a positive instance of the concept of Treasure".[5]

The learning database is first analysed and re-organised so that all the items related to the same concept are clustered together,[6] and then given to a multi-class decision list procedure[7] which actually produces the concept discriminating rules.[8] Each rule is represented as a 3-item list of the form `[isa <category name> [<list of feature values]]`, and can be read as "if in presence of feature values `[<list of feature values>]` then recognise the object as an instance of `<category name>`". The list of feature values is one disjunct of the concept intension generated by the learning algorithm.

---

[5]Implementation note: the learning database is built during the interpretation of the `learning_data_ruleset`, and it contains items such as `[danger negative [small sweet red]]` or `[treasure positive [small sweet red]]` etc.

[6]Implementation note: this is done by the procedure `exp_analyse`, whose output has the following representational structure: `[<category name> [[<positive/negative> [<feature list>]] ...]]`

[7]Implementation note: the procedure is `mdl`; it is this procedure that calls recursively the actual learning algorithm, `isc` (for 'incremental separate and conquer').

[8]Implementation note: Both the learning and the model database are implemented as poprulebase databases.

The learning algorithm that we use performs an incremental and heuristic search for disjunctive concepts [Langley, 1996]. The heuristic strategy used is called 'separate and conquer'. Being incremental means that an hypothesis is generated as soon as an experience is available. The algorithm retains a single hypothesis in memory, which is a disjunctive set of logical or threshold terms, and is initialised to a single term at the outset. It also retains up to the most recent k training instances for use in evaluating alternative hypotheses. It revises its hypothesis only when it makes an error in classification.

It takes as input a list of classified instances of one category and produces as output a disjunction of single-region descriptions to be used as a recognition test for the category. The input is represented as a list of lists with the following form: `[<positive/negative> [<list of features]]` For example, something like:

```
[[negative [large bitter shaval1 smeval2 colval4]]
 [negative [large bitter shaval1 smeval3 colval3]]
 [negative [small sour shaval2 smeval1 colval1]]
 [negative [medium bitter shaval2 smeval2 colval3]]
 [positive [small sweet shaval1 smeval2 colval5]]
]
```

Once the discriminating rules are stored in the model database and hence are available, the agent uses them during the process of selecting its target object: instead of selecting the nearest object in its perceptual field as its target, the agent will examine its perceptual field and look for an object that might belong to the appropriate category (that is, the kind of object that would enable it to achieve its current goal).

5.3 Discussion of related work

In [Cos-Aguilera et al., 2003] and [Cos-Aguilera et al., 2004] two different implementations of an agent learning affordances are presented, which however can be analysed in terms of the AB-architecture schema discussed above.

In both papers the problem addressed is the so-called *action selection problem*, that is deciding what behaviour to execute in a particular situation so that survival is guaranteed. For a given situation, two of the more crucial factors affecting the appropriatenss of such decision are i) the degree to which the chosen behaviour contributes to the satisfaction of the agent's needs, and ii) the choice of an appropriate object to interact with (whenever the chosen behaviour involves the interaction with an object). The authors explore the case where the information about whether a particular object in the environment is suited to a particular interaction is not hard-wired, but must be learnt. This actually corresponds to a learning of affordances in the sense of learning information about the objects' potential for action. The proposed cognitive architecture distinguishes between a module for behaviour selection and a module for extracting affordances. The behaviour selection module is based on [Cañamero, 1997] and consists of a set of survival-related internal variables associated with a set of motivations which control a repertoire of behaviours (actions that can be performed). A winner-take-all arbitration mechanism resolves conflicts among competing motivations, ensuring that the agent will try to satisfy only one need (its most urgent one) at a time.

The behaviour selection module is an instance of the IFB-architecture discussed in Sections 5.1.1 and 5.1.2: we have implemented it as a rulesystem, while Cos-Aguilera et al. have implemented it as a dynamical system (a set of equations expressing the generic behaviour of homeostatic internal variables and motivations).

The affordance extracting module corresponds to the top-right part of the AB-architecture diagram, represented in Figure 5.4. [Cos-Aguilera et al., 2003] exploits a set of feed-forward NNs, each linked to one action and devoted to estimating the probability of successfully performing that particular action with the object at hand. The NNs are trained via back-propagation, where the error to back-propagate is derived by monitoring the homeostatic variables and looking for sudden variations. The hard-wired association between homeostatic variables, motivations and behaviour execution captures what we called the functional definition of an affordance concept. So, the system proposed in [Cos-Aguilera et al., 2003] actually learns discriminating rules for affordances, as in our case.

[Cos-Aguilera et al., 2004] exploits a growing when required (GWR) self-organising feature map (SOFM) to combine the learning of affordances with the learning of a neural representation of regularities in the environment. The use of a GWR network allows the authors to avoid the symbolic integration of object features into objects, and hence enables experimenting in real world environments. In the case of [Cos-Aguilera et al., 2003] and in our case, the regular features of the objects in the environment do not need to be extracted because, in the simulation, objects *are defined* as sets of features.

The AB-architecture that we have presented captures the fact that affordance discriminating rules can be induced if it is known a priori which action satisfies which need. Somehow, the reverse is also possible: [Cos-Aguilera et al., 2005] shows that behavioural patterns can be induced if affordances are known a priori. Given a behaviour selection module analogous to the one used in the previous works, the authors explore the case where a priori knowledge of the affordance values of objects is available, but knowledge of the appropriateness of each behaviour to satisfy one need or another is lacking. Knowledge of affordances is provided to the system is terms of how likely an action is to succeed given the presence of a certain object, while learning obtains at the level of the arbitration mechanism. Both behaviour selection and learning are embedded in an actor-critic reinforcement learning algorithm: the actor performs behaviour selection by calculating the likelihood for each behaviour of leading to maximum cumulative reward, given the current motivational state (which consists of the drives and the affordances of the closest object in the agent's perceptual space); the critic estimates the cumulative reward resulting from the behaviour execution (decided by the actor) leading from the current motivational state to the optimal zone. In other words, what is learnt is how rewarding it is to perform a certain action (chosen among those that can actually be performed, given the objects at hand), given a motivational state where internal drives have values $v_1, ..., v_n$. The set of experiments described in the paper shows that, in an environment where affordances matter and perception is not noisy, the actor-critic algorithm provides appropriate policies to maintain physiological stability in a variety of scenarios. Of course this does not imply that the system learns to associate drives with the right action/s to satisfy them.

Our work as well as Cos-Aguilera's work shows that if an agent i) knows which action to perform to satisfy its present need and ii) is able to judge whether its need has been satisfied after performance of such action, then the agent can induce classification rules based on external object features. Later, these rules can be used to decide whether a given object can or cannot be acted upon with a certain action, that is whether or not it affords being acted upon in such a way. In addition, however, our work provides a deeper theoretical analysis of affordance concepts, and a more solid methodological framework to study their acquisition.

We offered a detailed discussion of different concept kinds where affordance concepts were opposed to taxonomic and goal-derived ones, and we explicitly proposed a layered structure for affordance concepts distinguishing between the functional aspect of the affordance and the discriminating one. In our implementation, this distinction is reflected by the existence of two distinct, though related, data structures, as described in Section 5.2.3. In particular, the discriminating rules for the affordance concepts are the output of a multiclass decision list procedure, and so, in principle, could grow indefinitely. This means that, were the agent able to learn also the functional rules (which is not the case, neither in our setting, nor in Cos-Aguilera's), nothing else in its architecture should be changed to support the acquisition of new, not pre-determined, discriminating rules. Less flexibly, in the system described by [Cos-Aguilera et al., 2003], whenever a new action-expectation association (corresponding to the functional aspect of an affordance concept) is learned, a new neural net should be added to the system. This is so because the discriminating aspect of the affordance concept is captured by Cos-Aguilera with a 1 to 1 association of a NN to an action, where the NN is trained to estimate the probability of successfully performing that particular action with the object at hand.

[Stoytchev, 2005] shows that if an agent i) has external information about an object (e.g. the object's shape) and ii) is able to judge whether the performance of a given action actually produced the desired effect, then the agent can learn the appropriate action to be performed to obtain the desired effect. The paper describes preliminary work toward a developmental approach for learning *binding* actions, that is actions that allow a robot to attach an object to its body so that the object's movements can be controlled reliably by the robot. The robot does not learn to categorise objects in terms of what they can be used for (e.g. edible, graspable, etc.). Instead, it learns which action (actually, which sequence of primitive actions) should be performed to obtain binding, when presented with a given object. So, it learns which actions are 'afforded' by a given object. The representation of the affordances uses a behaviour-based approach: this means that objects are internally represented in terms of the actions they afford. The learning involves a behavioural babbling stage in which the robot randomly chooses different exploratory behaviours, applies them to objects, and detects perceptual invariants in the resulting set of observations. The perceptual invariants are expressed as visual functions that detect regularities in the movements of objects relative to the body of the robot.

The work of Stoytchev is complementary to our work (and Cos-Aguilera's one) in the sense that, within Stoytchev's setting, the agent learns how to act properly on a given object, rather than learning how to recognise the 'good' target objects for a chosen action. Of course, these two abilities could be combined in order to obtain an even more sophisticated agent. More precisely, in [Stoytchev, 2005] the following results are presented: given a simulated model of a manipulator arm, with a simulated camera mounted above the robot's working area, and an environment containing seven different objects, the robot learns a series of behaviour sequences for binding with the objects. The objects have different shapes (namely, stick, spindle, mallet, dumbbell, beam, H-frame and π-frame), which make them easier or harder to grasp. For example, the dumbbell can be grasped only in its middle section, while the beam is too thick to fit in the robot's gripper. All the objects however are color coded, so that they can be uniquely identified by the robot. For each of the seven objects in the environment, a separate learning trial was conducted, and the experiments show that an adequate behaviour sequence for binding has been acquired for each object. Since most of the objects fit the robot's gripper, the sequence *touch-object* and *close-gripper* is the most frequently learned. However, for the H-frame and π-frame objects it is the *open-gripper* behaviour that is learned to achieve the binding. And also for the beam object an appropriate binding behaviour is learned: the robot learned to slide it sideways by sticking its gripper into the object and rotating its arm.

Using an experimental set-up analogous to that of Stoytchev, our work could be used to extend his results so that not only does the robot learn different behaviour sequences to achieve a binding behaviour, but it also learns what are the external features that characterise a class of objects for which the same binding behaviour sequence is adequate. Moreover, instead of using a set of separate learning trials for each object, the robot training could be performed along one single, incremental, learning path. For example, given the initial internal motivation of achieving a binding behaviour, the robot could focus its attention on a randomly chosen object in its sphere of reach, and try different sequences of behaviours on the given object until it discovers (and then masters) one, let's call it $binding1$, that successfully achieve binding. Both the binding action and the perceptual features of the object acted upon are stored in the memory. The robot can then focus on a new object. Since only one action is known to achieve binding, without any further exploration of its movement capabilities, the robot will try to satisfy its motive by performing $binding1$: if the action is successful, the robot will generalise over the perceptual features of the two objects acted upon so far, and start forming an affordance concept for $binding1$; if not, then an exploration of its movements repertoire will start, hopefully leading to a new behaviour sequence capable of achieving binding, let's say $binding2$. As time goes by, the robot should learn both a set of binding actions and a set of affordance concepts related to those actions that prescribe how objects should look like to be handled succesfully with a given action. Whenever a new object is faced, the affordance concept that best categorises it will determine which binding action the robot will attempt on it.

5.4 Summary

In this Chapter we have presented and discussed an agent architecture that supports the autonomous formation of affordance concepts. In particular we have shown that the architectural feature that supports the unsupervised formation of affordance concepts is the existence of a set of condition-action rules where the condition part expresses the performance of an action, while the action part expresses an expectation on the results of the action. Expectations are not generic, but rather have a definite content in terms of observable modifications of the agent's internal state, and for each action that has an associated expectation there is an affordance concept that could possibly be learnt.

In Section 5.1 we discussed the architecture of an agent exhibiting an innate fixed behaviour allowing it to survive in a friendly environment. The IFB-architecture simply ensures that the agent is well-coupled with the environment: it does not contain any implicit model of the objective relevant classes (those determined by the world laws) but it contains an implicit model of how the agent's motivational states and actions are linked together to produce a coherent behaviour. The IFB-architecture is therefore a control architecture that implements a motivational structure that keeps the agent alive, without the agent "knowing" about the world. It makes use of simple object detectors (and not object class detectors) and an innate association between internal states and action selection procedures. In Section 5.1.3 however we saw how the IFB-architecture can be extended to support the autonomous acquisition of a model of the affordances present in the environment. The learning of the affordances provided by the environment is achieved with the introduction of a further set of associations (captured in terms of condition/action rules) relating the performance of an action to definite expectations — for example in terms of values of internal variables — about the result of that action. These associations enable the agent to judge whether or not the target object is a positive instance of the affordance concept being learnt. As we said, an affordance concept refers to objects in terms of perceivable possibilities for action. This means that any affordance concept can be defined in terms of the goal that the agent wants to achieve by performing a certain action, and the action chosen to achieve it; in particular the goal can be used as the definite content of an expectation on the results of performing the chosen action.

In Section 5.2 a further extension of the IFB-architecture is presented, to obtain an agent with an adaptive behaviour. The AB-architecture is characterised by the presence of a module that controls the agent's behaviour in a way that is driven by the affordances that have been learned: the affordance concepts are used by the agent during its active phase, and more precisely when the target object to be approached and acted upon is chosen. The behaviour produced by the AB-architecture integrates the behaviour of the simpler IFB-architecture with both a module controlling the construction of a model of the environment, and a module that controls the agent's behaviour as described above. Among other things, behaviour control is achieved through the following steps:

- whenever the motivation to perform a certain action is triggered, an internal check is also performed to see whether an affordance-label is associated with that action;
- if so, a further check is performed to see whether a non-empty characterising description is also associated to the affordance-label;
- if so, the description is used in the procedure to select the target object upon which to perform the intended action (e.g. approach objects that match the description; or avoid objects that match the description), otherwise the agent relies on its default and innate target-selecting procedure.

Finally, in Section 5.3 we discussed the works of Cos-Aguilera on affordance learning, where a different type of implementation is presented which however can be analysed in terms of our AB-architecture schema. We also briefly reviewed the work of Stoytchev on a developmental approach for learning the binding affordances of objects.

Chapter 6

Conclusions

6.1 Contributions to knowledge

In this work we addressed the problem of autonomous concept formation from a design point of view, and autonomous agents, that is systems capable of interacting independently with their environment in the pursuit of their own goals, provided the framework in which we studied our problem. We focussed on the acquisition of affordance concepts, thus providing an initial answer to the question: What are the design features of an architecture supporting the acquisition of affordance concepts by an autonomous agent?

The main contributions to knowledge of our work have been:

- An argument showing that concepts should be thought of as belonging to different kinds, where the differences among these kinds are to be captured in terms of architecture features supporting their acquisition.
- A definition of affordance concepts and a description (and partial implementation) of an adaptive architecture (the AB-architecture) supporting the acquisition of affordance concepts.

The ability to form concepts is an important and recognised cognitive ability, thought to play an essential role in related abilities such as categorisation, language understanding, object identification and recognition, reasoning, etc., all of which can be seen as different aspects of intelligence. Even though the importance of concepts has been recognised, the nature of concepts is controversial, in the sense that there is no commonly agreed theory of concepts, and it is still far from obvious which representational means are most suited to capture the many cognitive functions that concepts are involved in. Therefore, among our goals was also the attempt to bring together different lines of argumentation that have emerged within philosophy, cognitive science and AI, in order to establish a solid foundation for further research into the representation and acquisition of concepts by autonomous agents.

With respect to the semantical aspect of concepts, we adopted a cognitive approach. More precisely, we took an internalist stance on the nature of concepts, and considered them as internal representations dependent, for their form and content, on the cognitive processes of organisms (natural or artificial) and on the specific types of interactions between organisms and their environment.

The relation between concepts and the world is an epistemological one, and it can be accounted for either in a pragmatic way or through some kind of correspondence theory. We adopted a pragmatic approach according to which a concept reflects a representational need that might have arisen to solve problems when acting in the world. Our motivations for these initial choices were given in Section 1.1.

In Sections 2.1 and 2.2 we presented some of the different theories of concepts proposed in the philosophical and psychological literature. The idea was to cover the major theoretical positions, highlighting their respective advantages and disadvantages in accounting for the acquisition and use of concepts. Then, in Section 4.1, we made our proposal to capture affordance concepts.

Affordances are the research subject of ecological psychology: they are emergent properties of the animal-environment system and they constitute what the animal normally pays attention to, in order to control its actions. Affordances are therefore related to an animal's goals and the actions that such animal can perform, and to say that animals can detect affordances actually is to say that animals have been designed by evolution to detect properties of the environment that are directly *relevant* to them, usually because their survival depends on these properties (for example, surfaces, textures, substances, but also properties such as 'graspable', 'stand-on-able', sit-on-able', etc.)

To make this notion of relevance more precise, we said that given an autonomous agent inhabiting an environment where different kinds of objects exist, these objects can be partitioned into relevant and irrelevant (for that agent) depending on whether they can modify the agent's internal states, and hence be useful or dangerous, when acted upon. So, if the agent has goals pertaining to its internal states (e.g. the goal of increasing its internal energy), we also say that a category of objects is relevant for an agent either because the objects can be used to achieve the agent's goals, or because they prevent such an achievement. Therefore, we proposed to define an *affordance concept* as a concept denoting a relevant category of objects, that is objects that either can be used to achieve the agent's goals, or prevent such an achievement. For example, if the agent has the goal of increasing its internal energy and eats an object O to achieve such goal, then, if the action succeeds, O can be categorised as something that affords eating. We then turned to the problem of how affordance concepts can be acquired.

When addressing concept formation in AI, what can be called the 'system level' is often overlooked, which means that concepts and categories are rarely studied from the point of view of a system, autonomous and complete, that might need such constructs and should be able to acquire them also by means of interactions with its environment, under the constraints of its cognitive architecture. Also within psychology, the focus is usually on structural aspects of concepts rather than on developmental issues. Our work was an attempt (i) to show that a system level perspective on concept formation is indeed possible and worth exploring, and (ii) to provide an initial, maybe simple, but concrete example of the insights that can be gained from such an approach.

In Sections 1.2 and 1.3 we argued for the role of concept formation as a requirement for intelligence from the point of view of AI. The notion of intelligence as used in AI often makes reference to the posession, implicit or explicit, of a *model* of the world by the system qualified as intelligent, where concepts are understood as basic representational units. Even though models and concepts can be made available to systems from the start, by means of large knowledge-bases, the ability to cope with changing domains or tasks often requires the ability to enlarge or modify the world models, thus learning new concepts. It is tempting to make concept formation rely only on the communication among agents with different degrees of knowledge, however the main problem of overlooking the acquisition of knowledge from the interaction with the world (rather than with another agent) is the difficulty if not the impossibility of dealing with failures that somehow involve the ability to recognise an object given a description of that object in terms of basic units that are not shared.

In Section 2.3, we reviewed some of the proposals to be found in the Machine Learning literature to characterise conceptual structures. We proposed a framework for analysing a learning system and saw how the concept formation process can be divided into two steps: the clustering phase and the characterisation phase.

In Chapter 3 we discussed our methodology and presented our proposal to address concept formation in terms of an architecture-based analysis. The idea was to provide an explanation of concept formation by specifying the *architecture* of a system (in our case, the architecture of an autonomous virtual agent) that could actually work and exhibit the ability to form concepts. Even though the methodology we proposed is general, we focused on a particular type of concepts, namely affordance concepts.

Finally in Chapter 5 we studied the architecture of an agent who discriminates among certain causal classes of objects, and is able to recognise the objects that instantiate these classes. Hence our aim was to show which are the architectural features that support such abilities as:

- acquiring a set of internal representations denoting the relevant classes of objects, and
- associating to these representations a set of rules to discriminate the objects that instantiate each class.

The architecture that enables an agent to acquire affordance concepts has been described in Chapter 4 pages 77-78 and its main feature is the existence of a set of condition-action rules where the condition part expresses the performance of an action, while the action part expresses an expectation on the results of the action which is not generic, but rather has a definite content in terms of observable modifications of the agent's internal state.

We first discussed the architecture of an agent exhibiting an innate fixed behaviour allowing it to survive in a given environment. Such an architecture simply ensures that the agent is well-coupled with the environment: it does not contain any implicit model of the objective relevant classes (those determined by the world laws) but it contains an implicit model of how the agent's motivational states and actions are linked together to produce a coherent behaviour. We then showed how to extend this architecture to support the autonomous acquisition of a model of the affordances present in the environment. The learning of the affordances provided by the environment is achieved with the introduction of a further set of associations (captured in terms of condition/action rules) relating the performance of an action to definite expectations — for example in terms of values of internal variables — about the result of that action. These associations enable the agent to judge whether or not the target object is a positive instance of the affordance concept being learnt. A further extension makes it possible to obtain an agent with an adaptive behaviour. The final architecture is characterised by the presence of a module that controls the agent's behaviour in a way that is driven by the affordances that have been learned: the affordance concepts are used by the agent during its active phase, and more precisely when the target object to be approached and acted upon is chosen.

Our analysis of the process of acquiring affordance concepts shows that these concepts have a layered structure: on the one hand we have the *functional* aspect of the affordance, the fact that an affordance enables grouping together things according to their potential use;

while on the other hand we have the *discriminating* aspect of the affordance, the fact that things serving the same use can sometime be recognised and grouped together also because they are similar in some other respect (for example, graspable things can be similar in shape). More precisely, the set of condition/action rules relating actions and expectations about the result of performing those actions stands as a representation of the functional aspect. The idea is that any such rule should be exploited to perform the initial clustering of external sensor data, so that later on a characterisation algorithm can be run on those clusters to produce a set of discrimination rules. We considered the case where the agent is provided with innate functional concepts denoting the relevant categories of objects in terms of associations between actions and expectations. This means that exploratory actions, on the results of which there is no definite expectation, are ruled out. In this case, no new affordance concepts can be learnt – i.e. concepts relating to learnt, but not innate, expectations. In the next section we will address the question about how to reduce innateness, so that the agent has a more general capability, which might lead to the discovery of affordances that were not innately determined.

The question of whether distinct kinds of concepts can be individuated and on which bases was discussed in Section 4.2. Our conclusion was for a distinction between affordance concepts and goal-derived concepts on the one hand, and taxonomic concepts on the other, captured in terms of processing differences. Affordance concepts are similar to goal-derived concepts, and are opposed to taxonomic concepts with respect to how they are processed. The learning process of goal-derived concepts differs from the exemplar-based acquisition of taxonomic concepts because the new goal-derived concepts are formed by manipulating existing knowledge in memory, rather than experiencing with already available sets of exemplars. Affordance concepts are similar to goal-derived concepts, in the sense that they are also learned in a mediated way. Instead of conceptual combination to optimize a plan, however, affordance concepts rely on a direct interaction with the environment to establish appropriate clustering criteria.

6.2 Directions for future research

6.2.1 Reducing innateness: towards a minimal architecture

The AB-architecture described in Chapter 5 takes as a starting point an agent with a motivational structure inducing a simple autonomous behaviour. To begin with, we assumed that the initial information coded into the architecture, as if it were innate, was a set of condition/action rules associating (i) particular internal states with the intention to perform certain specific actions (for example, the energy level being below a certain threshold and the intention of eating) and (ii) the performance of an action with a content-specific expectation on its results (for example, eating and expecting the energy level to increase). We will now discuss how to weaken this assumption.

First of all, let's assume that when an agent is first turned on, it is only able to acknowledge the degree of stability of its internal milieu (something like feeling more or less good or bad), even though its physiology is such that many different internal states may actually contribute to these internal feelings, such as assuming a certain bodily posture, feeling more or less satisfied, replenished, aching, bored, etc. Let's also assume that the initial motivational structure of the agent is such that the agent has a motivation to act whenever the stability of its internal milieu is below a certain threshold. In other words, at the beginning the agent has only one goal, namely, the goal of feeling good, and when this is not the case the agent will start acting in order to re-establish the good internal feeling. Since at the beginning the agent knows very little (about its internal states, about its environment, about which of its actions transform which state into which other state, or which state will turn into which other state because of the world laws), at the beginning its actions will be random and very likely ineffective. With respect to this initial situation, our questions are: What kind of control structure would enable our agent to learn to recognise its needs and act in a directed way, to satisfy them? How could the agent also learn to exploit the environment so that actions are actually performed when the expected results are likely to obtain (e.g. because the proper objects to be acted on are indeed in the agent's action range)?

Using an example to support our analysis, we will propose a coarse classification of the knowledge contents that can (and should) be acquired and some insights on the sequence with which this acquisition is going to happen. A further question concerns the actual learning situations that should occur: for example it might be the case that at the beginning some innate reflex, a friendly environment or even a teacher are needed to ensure that the initial exploration is constrained in such a way as to ensure the agent survival (e.g. first of all, learn to eat). Also, the initial phases of learning might actually be a form of internal development or maturation, happening in some sort of nursery environment.

A worked-out example

Let's assume that the agent acts only when the stability of its internal milieu is below a certain threshold (the agent is not feeling OK), and let's consider the following control sequence governing the agent:

- check internal state I; if internal state is OK then do nothing; else act with action A;
- check internal state again; if new internal state I' still not OK continue acting, otherwise store in the memory that a transition is possible from negative internal state I to positive internal state I' via action A.

The idea is to store both rules such as *If [internal state* I*] and [want to achieve internal state* I'*] then [perform action* A*]* and rules such as *If [internal state* I*] and [intend to perform action* A*] then [expect internal state* I'*].* The first rules are to be used for planning while the second rules are to be used to learn affordance concepts along the lines sketched in the previous chapters. After a while, an agent with such learning abilities will have at its disposal a set of associations relating an internal negative state X with an internal positive state Y via an action A. The problem however is that only certain internal states (feeling OK vs. not feeling OK) have been taken into consideration. What about representational internal states, that is, internal states that have some semantic content? For example, what about a structure associating a certain visual experience with a certain tactile experience via an action of reaching out and touching? Or structures capturing the fact that when the head moves or an object is moved (and hence certain kinesthetic sensations are experienced), the visual scene also changes?

In other words, how can we make our mechanism look for discriminating data among the information that is provided by the external sensors?

Consider the following initial situation: the agent is hungry and object O is outside hand-reaching distance (for example, the agent must walk 3 steps to reach O with its hand). Also, the world laws (here what matters is probably the agent's physiology) are such that, in order to remove hunger, the agent should eat. Let's assume that O is indeed food, and that the agent knows nothing of its environment, and very little of itself (event though, as we will see, it has some innate capabilities). How could our agent possibly end up knowing that O affords eating?

Since it is hungry, the agent feels bad and an impulse for acting is triggered: this is an innate bit, in the sense that when the agent feels bad, it automatically starts acting with the goal of removing the bad feeling and with a generic expectation about not feeling bad anymore. At the beginning, however, the agent does not know exactly what to do: it does not know which action, or sequence of actions will remove the bad feeling in this particular case. This means that, in the initial situation, the agent has innate knowledge of feeling bad and feeling good, but does not know that feeling bad (and feeling good) can actually take many different forms (feeling hungry, having head-ache, feeling replenished, feeling that the head is OK, etc.) which depends on different configurations of the internal sensor values (the agent has no prior knowledge of all these possible forms; they are discovered by living, in the same sense in which I discover what is feeling the pain of delivering a child only the first time I actually give birth). So, there is something like an innate capacity of judging the degree of badness/goodness of an internal state, but not yet a taxonomical knowledge of types of bad/good states. Also, the agent does not know that some bad internal states can be changed into less bad or even good internal states by acting. The only thing that is so to say pre-programmed is the impulse to act somehow when feeling bad, with a generic expectation of feeling better as a result of the action. So, let's assume that, in order to remove hunger, the agent decides to move its hand, reaching for something to grasp.

While the agent moves the hand trying to grasp object O, it will experience a sequence of internal sensations of movement coupled with a sequence of external sensor contents, such as a sequence of visual scenes (since the visual scene keeps changing, for example in terms of relations of the visible parts of the body with the background). Let's assume that this stream of sensations is recorded (not only the two sequences, but the two sequences associated in such a way that the association is to be understood as the fact that the changes in the first sequence — the internal sensations of movement — are *causes* for the changes in the second sequence — the external sensor contents). What can be learnt from such records? Statistical data analysis techniques such as those discussed in [Cohen et al., 2002] could extract from such records (i) characteristic patterns of hand movements HMs, (ii) characteristic patterns of visual scenes VSs, and (iii) characteristic patterns of associations between HMs and VSs. These last association patterns could then be translated into rules such as 'if visual scene VS and hand movement HM then visual scene VS' (with probability p)' where the 'if ... then ...' is understood temporally, that is something like 'visual scene VS can be transformed into visual scene VS' by performing hand movement HM (with probability p)'. Another association pattern that could be learnt and then transformed into a rule similar to the one just discussed would relate feelings of posture (internal sensations about the body configuration) and hand movements (we are here assuming a distinction between kinesthetic sensations of movements and other kinesthetic sensations). All of these rules are extremely important because they can later be used by the agent to plan in a means-ends fashion. So, let's assume that a data-analytic mechanism capable of producing such results is part of our agent.

Up to now we have been assuming that the agent is moving the hand trying to grasp object O. We said however that O is out of reach, so this means that nothing happens in terms of removing the bad feeling of being hungry. So after a while the agent stops moving the hand and tries something else. Here we are making an assumption about another innate bit within the agent's architecture: a control structure that checks the internal state after an action has been performed, to see whether the generic expectation on feeling better has been fulfilled; if not, the impulse of acting should be directed towards another action. In order to perform the statistical data analyses mentioned in the previous paragraph, however, more than a single stream of data should be present: for this, we assume that 'trying an action' is not just trying one precise movement (x degrees East, y degrees North and z centimeters forward); rather, it means exploring for a while the workspace of one effector. So, the control structure that we have in mind would first of all recognise the presence of a 'bad' internal state; it would then set a generic expectation about a better internal state; then it would select an effector, reserve part of the memory to store the data that are about to come, and then yield part of the control to an action procedure that explores the workspace of the selected effector. The internal state will be monitored continously, and after a while, if nothing happens in terms of removing the badness, the higher control structure will resume control, give the stored data to the data analysis procedure to extract patterns, and then select another effector to make a new attempt at removing the bad internal state. This means that the agent's temporal flow of sensations will automatically receive some 'delimitations': one delimitation is given by the transitions from bad internal states to good internal state, and within these big chunks there will be delimitations corresponding to the exploration of the workspace of different effectors.

Now, let's assume that the agent starts exploring a new action, namely the workspace of the successive activation of three effectors, something like approach-grasp-eat — AGE for short — and that at a certain point the agent ends up in a situation where it is not hungry anymore. Like in the previous case, we assume that all the sensor data are stored in order to be analysed. This time the agent could learn something like: 'if bad feeling of type hungry and action AGE then good feeling of type replenished'. It seems therefore that there are three types of 'if ... then ... ' structures that could be learned:

1. if bad feeling of type X and action A then good feeling of type Y
2. if internal feeling of posture P and action A then internal feeling of posture P'
3. if visual scene V and action A then visual scene V'

With the induction of rules such as 1. and 2. the agent learns about causal chains that pertain to its inner flow of sensations; let's call them I-causal-chains. While with the induction of rules such as 3. the agent learns about causal chains that pertain to its outer flow of sensations (data coming from external sensors); let's call them E-causal-chains.

In order to learn that an object affords eating, the agent should learn that certain I-causal-chains and certain other E-causal-chains are related. More precisely, the agent should discover that sometimes an I-causal-chain is the case *also because* an E-causal-chain is the case, that is, because the action is actually an action upon the environment. In order to achieve this type of learning, we propose the following mechanism. At the beginning the agent only learns that certain actions enable the transition from a certain internal state to a certain other internal state and *it assumes* that it is so no matter what the external state is. This means that, whenever an internal state transition happens, the agent so to speak focuses only on the internal side of its flow of sensations, and learns about actions enabling the transition independently of the external state. When the rule expressing such an internal transition is learnt

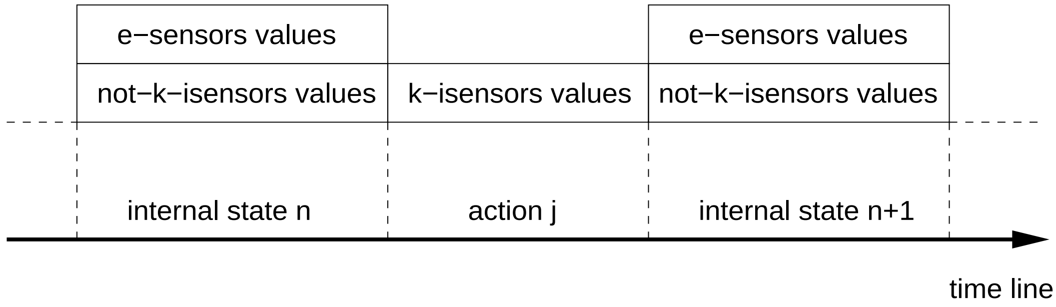


Figure 6.1: A schematic representation of the stream of experience of an agent.

and the agent can therefore have expectations on the results of its actions, there is room for the agent to learn about the external environment. An example of such a progression from internal experience to external experience is described in [Cohen et al., 1996]. In particular, whenever the expected internal transition does not happen the agent could reason as follows: Given the rule I learnt, my internal state X was such that, by doing action A I would then feel Y, but this was not the case now, so the reason for this has to be searched in the external state.

Final considerations

With the previous example we have illustrated how certain associations between internal states and actions and between actions and expectations about the results of such actions could be acquired in the first place, and how the learning of transformations of internal states seems to be prior to the learning of action preconditions that relate to the external state. These last considerations involved some preliminary distinctions among different types of internal states. We will now suggest how these different types of internal states can be related to different kinds of concepts, as concrete ideas for future work to be carried out. Along these lines we will also hint at research issues that need further attention.

Consider sensations like feeling hungry or experiencing a certain kinesthetic sensation on the one hand, and experiencing a certain visual scene on the other. Feeling hungry, like feeling a certain movement, depends on a configuration of the body (changes in the body cause changes in the feeling), while experiencing a certain visual scene, or having a certain tactile experience depends on a configuration of the body *and* the environment (changes in the body and/or changes in the environment cause changes in the experience).

We propose therefore to make a distinction between sensations like feeling hungry, feeling pain or kinesthetic sensations, and sensations like being-appeared-to-red. The first sensations correspond to values of internal sensors (i-sensors), the second to values of external sensors (e-sensors). Then, among inner sensors, we propose to distinguish between those that provide sensations of movement (k-isensors) and all the others (not-k-isensors). The k-isensors provide sensations that mediate the transformation of one internal state into another, that is: sensations that corresponds to actions in the sense of body movements. An internal state is a combination of values of not-k-isensors and e-sensors, while a combination of values of k-isensors is an action. For sure it is not easy to isolate, by introspection, something like ‘only kinesthetic sensations’ and we always have the feeling of a continous, uninterrupted stream of experiences, but if the distinctions are accepted, then it is possible to represent the phenomenon of perceiving the external world as a succession of phases such as in Figure

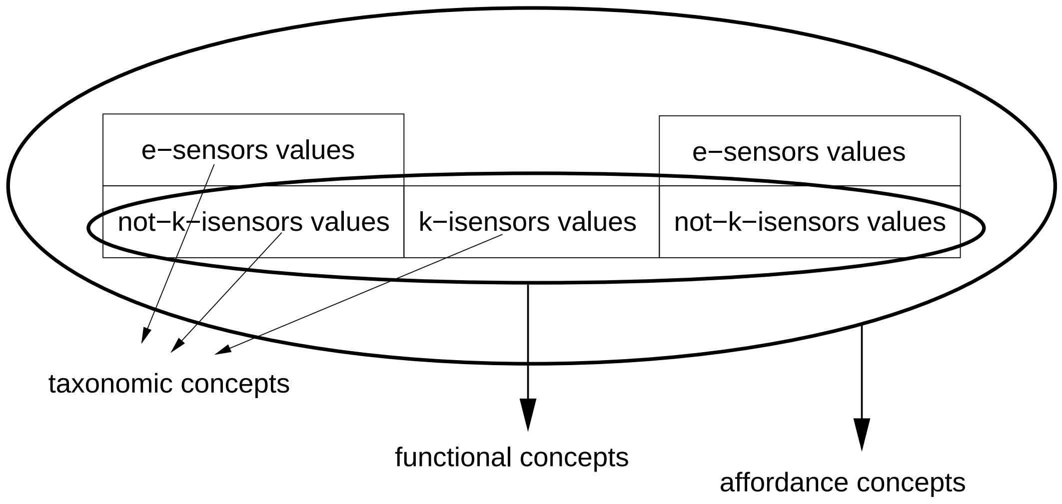


Figure 6.2: Different types of concepts and their relations with sensor field values.

6.1.

So, the stream of experience is a sequence internal state, action, internal state, action, internal state, etc. As a matter of fact, through time we witness a continuous and correlated transformation of the fields of e-sensors and not-k-isensors on one hand, and k-isensors on the other. The idea is that clustering upon series of sensor fields of the same type gives rise to taxonomic concepts (similar external states, similar internal states and similar actions), clustering upon series of transformations of not-k-isensors fields by means of k-isensors fields gives rise to functional concepts, while clustering upon series of e-sensors fields associated to the same functional concept gives rise to affordance concepts (see Figure 6.2.)

According to the notion of conceptual space discussed in [Gärdenfors, 2000], to each sensor corresponds a sensor field which has certain quality dimensions endowed with topological and/or ordering structures. These dimensions correspond to the different ways stimuli can be judged to be similar or different within that field. Whenever the sensor is active, its sensor field is 'shaped' in a certain way, that is, quality dimensions for that field happen to have certain specific values. An important issue that needs to be addressed is therefore determining the field characteristics of each sensor field in Figure 6.1. A further research issue that needs attention concerns the notion of action and the relations between the sensory-motor system and higher cognitive skills such as categorisation. Our analysis was based on the assumption that the sensory-motor system has the right kind of structure to characterise both sensory-motor and more abstract concepts. For primitive actions such as grasping an initial account is given in [Gallese and Lakoff, 2005], but there is a long way to go to establish what happens when more complex actions are involved.

6.2.2 Learning more than affordance concepts

As discussed in Chapter 3, an architecture provides a set of principles for organising a system, in our case an agent viewed primarily as a control system. In addition to supplying structure, the architecture imposes constraints on the relations among its sub-parts and on the way control problems can be solved. The phenomenon we have been interested in is the autonomous acquisition of concepts as part of a world modelling ability. We understand a world model to be a set of data structures that are part of a controlling unit. Those data structures are used to guide action in a mediated, indirect way: instead of having a direct link between sensors and effectors, the choice of which effector to activate is based on an interpretation of the information provided by the sensors by means of the information provided by the model. We do not claim that a world model is a centralised data structure (it could be distributed across parts of the architecture and even consists of a mixture of different data and procedures/mechanisms), but we maintain that a model is built out of conceptual representations that are primarily viewed as enduring structures and organisers of activity, rather than abstractions of some aspect of reality.

The general architecture that supports a world modelling ability is represented in Figure 5.1 and distinguishes between an innate control unit and an acquired control unit. The innate control unit implements a motivational structure where internal 'need' states are associated to actions meant to satisfy the need. Besides this, it also implements a memory for the experience trace and a learning component operating on such memory. The output of the learning component ranges from classificatory rules to prediction rules and constitutes the world model. When the world model is not empty it affects the controlling process, modifying the initial, built-in associations between internal states and action selection procedures. Through time we therefore see the system switching from the innate control mechanism to the acquired one.

This schema is very general and may cover the acquisition of different types of concepts and other modelling components depending on the specificity of the experience memory, the learning mechanisms, and the procedures that implement the interactions between the model (in its different phases of construction) and the innate control unit. In Chapter 5 we discussed at length the case for affordance concepts. Some initial ideas to adapt our schema to the acquisition of taxonomic concepts can be found in [Chella et al., 1997] and are discussed in the next paragraphs.

As we said, taxonomic concepts are created to capture the correlational structure of the physical world (not all possible correlations, but those relevant to the needs of the organism) and are induced from experience with exemplars. Since the system discussed by Chella *et al.* is a vision system for object recognition, the relevant correlations that matter here are at the visual level and more precisely involve shape, orientation and location. The proposed system architecture uses three levels of representation: at the bottom level, information is represented as $2^{1/2}$-D description of the input image; at the middle level, information consists of a combination of 3-D geometrical primitives based on superquadrics; finally, at the top level, information is represented using description logic. The top and the middle representational levels are linked by a *focus of attention* mechanism that controls the exploration process of the perceived scene. This link is implemented by attractor neural networks trained by a careful learning phase and what is learned is how to focus visual attention and what to look for when scanning the visual field. The focus of attention mechanism is needed in order to impose a sequential order in the analysis of the perceptual data, thus reducing the time and computational resources wasted detecting true but irrelevant details.

Even though Chella *et al.* call the three representational levels 'sub-conceptual', 'conceptual' and 'symbolic' respectively, with respect to our framework the conceptual representations (in this case, taxonomic concepts) are to be found at the top level. This is so, because it is only the activation of the top representational level that causes a switch from an innate, environment-driven control of attention, to an acquired, model-driven control of attention, in agreement with our proposed architecture. Let's see in more detail how the switching has been achieved in the system under examination. Chella *et al.* distinguish among three functioning modes of the focus of attention mechanism, which they label 'reactive', 'linguistic' and 'associative'. The only default functioning mode, however, is the reactive mode, which can therefore be considered the innate controlling mechanism of the focus of attention. In the reactive mode, the grouping and ordering of the perceptual data is determined only by the characteristics of the visual stimulus, such as the volumetric extension of the forms, or the aggregation density of the perceived objects. In the linguistic and associative mode, the focus of attention is driven by the information stored in the top representational level, and more precisely this is achieved by generating an expectation that a perceptual input with given characteristic is to be found in the scene (possibly with further constraints such as what should be next to what). As the authors themselves notice "the distinction between the associative mode and the linguistic mode is a soft one" in the sense that also the linguistic mode associates a perceived input with some expected other input, and in fact both modes are implemented in the same way. The main difference between them being that the linguistic mode potentially captures only the associations explicitly described using the representational mechanism chosen by the authors for the top representational level, that is part-whole relationships described using description logic, e.g. *has-part (hammer, handle)* or *has-part (hammer, head)*, while the associative mode potentially captures any other free association, e.g. associating the appearance of a hammer with the appearance of nails, because the two objects have been often perceived as belonging to the same visual scene. With respect to our framework, the distinction between linguistic and associative mode is irrelevant, and what really matters is that the knowledge bases supporting their functioning can be learnt and then used to drive the focus of attention.[1] Moreover, it is the content of these knowledge bases that constitutes an example of taxonomic concepts because the purpose of these data structures is to group together correlational structures in the perceptual field (in this case, the visual field) in order to constrain and guide successive perceptual actions aimed at gathering further information on the surrounding environment.

[1] In the system described by Chella *et al.*, the terminological knwoledge base that constitutes the top representational level is built-in, and only the associative knowledge base is actually learned. In principle, however, both could be acquired.

Bibliography

[Adolph and Eppler, 1998] Adolph, K. E. and Eppler, M. A. (1998). Development of visually guided locomotion. *Ecological Psychology*, 10:303–322.

[Agre and Chapman, 1987] Agre, P. E. and Chapman, D. (1987). Pengi: An implementation of a theory of activity. In *Proceedings of AAAI-87*, pages 268–272, Seattle, WA.

[Aha, 1991] Aha, D. W. (1991). Case-based learning algorithms. In *Proceedings of the DARPA Case-Based Reasoning Workshop*, pages 147–158, Washington, D.C. Morgan Kaufmann.

[Ahn, 1998] Ahn, W. (1998). Why are different features central for natural kinds and artifacts? the role of causal status in determining feature centrality. *Cognition*, 69:135–178.

[Allen and Bekoff, 1997] Allen, C. and Bekoff, M. (1997). *Species of Mind*. MIT Press, Cambridge, Mass.

[Anderson et al., 2004] Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., and Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, 111(4):1036–1060.

[Armstrong et al., 1999] Armstrong, S. L., Gleitman, L. R., and Gleitman, H. (1999). What some concepts might not be. In [Margolis and Laurence, 1999], pages 225–260.

[Ashby, 1952] Ashby, W. R. (1952). *Design for a Brain. The Origin of Adaptive Behaviour*. Chapman & Hall, London.

[Baillargeon, 1993] Baillargeon, R. (1993). The object concept revisited: new directions in the investigation of infants' physical knowledge. In Granrud, C. E., editor, *Carnegie Mellon Symposia on Cognition: Visual perception and cognition in infancy*, volume 23, pages 265–315. Lawrence Erlbaum, Hillsdale, NJ.

[Barsalou, 1983] Barsalou, L. W. (1983). Ad hoc categories. *Memory and Cognition*, 11:211–227.

[Barsalou, 1985] Barsalou, L. W. (1985). Ideals, central tendency, and frequency of instantiation as determinants of graded structure of categories. *Experimental psychology: learning, memory and cognition*, 11:629–654.

[Barsalou, 1991] Barsalou, L. W. (1991). Deriving categories to achieve goals. In Bower, G. H., editor, *The psychology of learning and motivation: Advances in research and theory*, volume 27, pages 1–64. Academic Press, San Diego, CA.

[Barton and Komatsu, 1989] Barton, M. E. and Komatsu, L. K. (1989). Defining features of natural kinds and artifacts. *Journal of Psycholinguistics Research*, 18:433–447.

[Bates et al., 1991] Bates, J., Loyall, A. B., and Reilly, W. S. (1991). Broad agents. In *AAAI spring symposium on integrated intelligent architectures*. American Association for Artificial Intelligence. (Repr. in SIGART BULLETIN, 2(4), Aug. 1991, pp. 38-40).

[Beaudoin, 1994] Beaudoin, L. P. (1994). *Goal Processing in Autonomous Agents*. PhD thesis, School of Computer Science, The University of Birmingham.

[Bekoff et al., 2002] Bekoff, M., Allen, C., and Burhgardt, G. M., editors (2002). *The Cognitive Animal: Empirical and Theoretical Perspectives on Animal Cognition*. MIT Press, Cambridge, Mass.

[Bloom et al., 1993] Bloom, P., Tinker, E., and Margulis, C. (1993). The words children learn: evidence against a noun bias in early vocabularies. *Cognitive Development*, 8:431–450.

[Bowerman and Levinson, 1999] Bowerman, M. and Levinson, S., editors (1999). *Language Acquisition and Conceptual Development*. Cambridge University Press, Cambridge, UK.

[Brachman and Levesque, 1985] Brachman, R. J. and Levesque, H. J., editors (1985). *Readings in Knowledge Representation*. Morgan Kaufmann Publishers, San Francisco, CA.

[Brooks, 1986] Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robot Automation*, 2(1):14–23.

[Brooks, 1991] Brooks, R. A. (1991). Intelligence without representation. *Artificial Intelligence*, 47.

[Brooks, 2002] Brooks, R. A. (2002). *Flesh and Machines*. Pantheon Books, New York.

[Bruner et al., 1999] Bruner, J., Goodnow, J., and Austin, G. (1999). The process of concept attainment. In [Margolis and Laurence, 1999], pages 101–123.

[Burton, 2000] Burton, G. (2000). The role of the sound of tapping for nonvisual judgement of gap crossability. *Journal of Experimental Psychology: Human Perception and Performance*, 26:900–916.

[Cañamero, 1997] Cañamero, L. D. (1997). Modeling motivations and emotions as a basis for intelligent behavior. In Johnson, W. L., editor, *Proc. First Intl. Conf. on Autonomous Agents*, pages 148–155, New York. ACM Press.

[Caplan and Barr, 1991] Caplan, R. J. and Barr, R. A. (1991). The effects of feature necessity and extrinsicity on gradedness of category membership and class inclusion relations. *British Journal of Psychology*, 84:427–440.

[Carey, 1985] Carey, S. (1985). *Conceptual Change in Childhood*. Bradford Books, Cambridge, Mass.

[Carnap, 1959] Carnap, R. (1932/1959). The elimination of metaphysics through logical analysis. In Ayer, A., editor, *Logical Positivism*, pages 60–81. The Free Press, New York.

[Carnap, 1956] Carnap, R. (1956). *Meaning and necessity*. University of Chicago Press, Chicago, 2nd edition.

[Chaudhri et al., 1998] Chaudhri, V. K., Farquhar, A., Fikes, R., Karp, P. D., and Rice, J. J. (1998). OKBC: A programmatic foundation for knowledge base interoperability. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, Madison, Wisconsin. American Association for Artificial Intelligence.

[Chella et al., 1997] Chella, A., Frixione, M., and Gaglio, S. (1997). A cognitive architecture for artificial vision. *Artificial Intelligence*, 89:73–111.

[Chomsky, 1986] Chomsky, N. (1986). *Knowledge of language*. Praeger, New York.

[Cohen et al., 1996] Cohen, P. R., Oates, T., Atkin, M. S., and Beal, C. R. (1996). Building a baby. In *Proceedings of the Eighteenth Annual Conference of the Cognitive Science Society*, pages 518–522.

[Cohen et al., 2002] Cohen, P. R., Oates, T., Beal, C. R., and Adams, N. (2002). Contentful mental states for robot baby. In *Proceedings of the 18th National Conference on Artificial Intelligence*, pages 126–131, Edmonton, Alberta, Canada.

[Cooper, 2002] Cooper, R. P. (2002). *Modelling High-Level Cognitive Processes*. Lawrence Erlbaum, Hillsdale, NJ.

[Corcho et al., 2003] Corcho, O., Gómez-Pérez, A., Leger, A., Rey, C., and Tormani, F. (2003). An ontology-based mediation architecture for e-commerce applications. In *Proceedings of IIS2003*.

[Cos-Aguilera et al., 2003] Cos-Aguilera, I., Cañamero, L., and Hayes, G. (2003). Motivation-driven learning of object affordances: First experiments using a simulated khepera robot. In Detje, F., Dörner, D., and Schaub, H., editors, *The Logic of Cognitive Systems. Proceedings of the Fifth International Conference on Cognitive Modelling, ICCM*, pages 57–62. Universitäts-Verlag Bamberg.

[Cos-Aguilera et al., 2004] Cos-Aguilera, I., Cañamero, L., and Hayes, G. (2004). Using a sofm to learn object affordances. In *Proceedings of the Workshop on Physical Agents (WAF04)*, University of Girona.

[Cos-Aguilera et al., 2005] Cos-Aguilera, I., Cañamero, L., Hayes, G., and Gillies, A. (2005). Ecological intergration of affordances and drives for behaviour selection. In *Proceedings of MNAS2005*, Edimburgh.

[Davidsson, 1992] Davidsson, P. (1992). Concept acquisition by autonomous agents: Cognitive modeling versus the engineering approach. *Lund University Cognitive Studies*, 12.

[Davis, 1999] Davis, D. N. (1999). Computational emergence and computational emotion. In *Proceedings of the IEEE Symposium on Systems, Man and Cybernetics*, Tokio, Japan.

[Davis, 2001] Davis, D. N. (2001). Control states and complete agent architectures. *Computational Intelligence*, 17(3).

[Donini et al., 1996] Donini, F., Lenzerini, M., Nardi, D., and Schaerf, A. (1996). Reasoning in description logics. In Brewka, G., editor, *Principles of Knowledge Representation and Reasoning*, Studies in Logic, Language and Information, pages 193–238. CLSI Publications.

[Ellman, 1989] Ellman, T. (1989). Explanation-based learning: a survey of programs and perspectives. *ACM Computing Surveys*, 21(2):163 – 221.

[Fensel, 2000] Fensel, D. (2000). Oil in a nutshell. In Dieng, R., editor, *Knowledge Acquisition, Modeling, and Management, Proceedings of the European Knowledge Acquisition Conference 2000*, Lecture Notes in Artificial Intelligence, New York. Springer-Verlag.

[Ferguson, 1995] Ferguson, I. A. (1995). Integrated control and coordinated behavior: a case for agent models. In *Proceedings of the workshop on agent theories, architectures, and languages on Intelligent agents*, pages 203–218, New York. Springer-Verlag.

[Fisher et al., 1992] Fisher, D. H., Pazzani, M. J., and Langley, P., editors (1992). *Concept formation: knowledge and experience in unsupervised learning*. Morgan Kaufmann, San Mateo, CA.

[Fodor, 1981] Fodor, J. A. (1981). The present status of the innateness controversy. In *Representations: Philosophical Essays on the Foundations of Cognitive Science*, pages 257–316. MIT Press, Cambridge, Mass.

[Fodor, 1990a] Fodor, J. A. (1990a). Information and representation. In Hanson, P., editor, *Information, Language and Cognition*, pages 175–190. University of British Columbia Press.

[Fodor, 1990b] Fodor, J. A. (1990b). *A theory of content and other essays*. MIT Press, Cambridge, Mass.

[Franklin, 1995] Franklin, S. P. (1995). *Artificial Minds*. The MIT Press, Cambridge, Mass.

[Frege, 1892] Frege, G. (1892). Über sinn und bedeutung. *Zeitschrift für Philosophie und philosophische Kritik*, NF100.

[Gabora and Aerts, 2002] Gabora, L. and Aerts, D. (2002). Contextualizing concepts. In *Proceedings of the 15th International FLAIRS Conference (Special Track 'Categorization and Concept Representation: Models and Implications')*, Pensacola, Florida. American Association for Artificial Intelligence.

[Gallese and Lakoff, 2005] Gallese, V. and Lakoff, G. (2005). The brain's concepts: the role of the sensory-motor system in conceptual knowledge. *Cognitive Neuropsychology*, 21.

[Gärdenfors, 1993] Gärdenfors, P. (1993). The emergence of meaning. *Linguistics and Philosophy*, 16:285–309.

[Gärdenfors, 2000] Gärdenfors, P. (2000). *Conceptual Spaces*. MIT Press, Cambridge, Mass.

[Genesereth and Nilsson, 1987] Genesereth, M. R. and Nilsson, N. J. (1987). *Logical foundations of artificial intellignce*. Morgan Kaufmann, Palo Alto, CA.

[Gentner and Boroditsky, 1999] Gentner, D. and Boroditsky, L. (1999). Individuation, relativity and early word learning. In [Bowerman and Levinson, 1999].

[Gettier, 1963] Gettier, E. (1963). Is justified true belief knowledge? *Analysis*, 23.

[Gibson, 2000a] Gibson, E. J. (2000a). Perceptual learning in development: some basic concepts. *Ecological Psychology*, 12(4):295–302.

[Gibson, 2000b] Gibson, E. J. (2000b). Where is the information for affordances? *Ecological Psychology*, 12(1):53–56.

[Gibson, 1950] Gibson, J. J. (1950). *The perception of the visual world*. Houghton Mifflin, Boston.

[Gibson, 1966] Gibson, J. J. (1966). *The senses considered as perceptual systems*. Houghton Mifflin, Boston.

[Gibson, 1979] Gibson, J. J. (1979). *The ecological approach to visual perception*. Laurence Erlbaum, Hillsdale, NJ.

[Glenberg, 1997] Glenberg, A. M. (1997). What memory is for. *Behavioral and brain sciences*, 20:1–55.

[Gómez-Pérez, 2000] Gómez-Pérez, A. (2000). Ontology domain modeling support for multilingual services in e-commerce: Mkbeem. In *Proceeding of the ECAI-2000 Workshop on Applications of Ontologies and Problem-solving Methods*, Berlin.

[Gómez-Pérez et al., 2003] Gómez-Pérez, A., Fernández-López, M., and O.Corcho (2003). *Ontological Engineering*. Springer-Verlag, New York.

[Goodman, 1955] Goodman, N. (1955). *Fact, Fiction, and Forecast*. Harwrad University Press, Cambridge, Mass.

[Gruber, 1992] Gruber, T. R. (1992). Ontolingua: a mechanism to support portable ontologies. Technical Report KSL 91-66, Knowledge System Laboratory, Stanford University.

[Gruber, 1993] Gruber, T. R. (1993). Towards principles for the design of ontologies used for knowledge sharing. In Guarino, N. and Poli, R., editors, *Formal ontology in conceptual analysis and knowledge representation*. Kluwer.

[Guarino and Welty, 2002] Guarino, N. and Welty, C. (2002). Evaluating ontological decisions with ontoclean. *Communications of the ACM*, 45(2):61–65.

[Guha and Lenat., 1990] Guha, R. V. and Lenat., D. B. (1990). Cyc: A midterm report. *AI Magazine*, 11(3):33–59.

[Hawes, 2003] Hawes, N. (2003). *Anytime Deliberation for Computer Game Agents*. PhD thesis, School of Computer Science, The University of Birmingham.

[Hirschfeld and Gelman, 1994] Hirschfeld, L. A. and Gelman, S. A. (1994). Domain specificity: an introduction. In Hirschfeld, L. A. and Gelman, S. A., editors, *Mapping the Mind: Domain Specificity in Cognition and Culture*, pages 3–55. Cambridge Univ. Press, Cambridge, UK.

[Hoffman et al., 2002] Hoffman, R. R., Hayes, P., Ford, K. M., and Hancock, P. (2002). The triples rule. *IEEE Intelligent Systems*, 17(3):62–65.

[Jackendoff, 1983] Jackendoff, R. (1983). *Semantics and Cognition*. MIT Press, Cambridge, Mass.

[Jackendoff, 1987] Jackendoff, R. (1987). On beyond zebra: the relation of linguistic and visual information. *Cognition*, 26:89–114.

[Jackendoff, 1990] Jackendoff, R. (1990). *Semantic structures*. MIT Press, Cambridge, Mass.

[Johnson, 1997] Johnson, T. R. (1997). Control in act-r and soar. In Shaflo, M. and langley, P., editors, *Proceedings of the 19th Annual Confernce of the Cognitive Science Society*, pages 343–348, Hillsdale, NJ. Lawrence Erlbaum Associates.

[Jones, 2003] Jones, K. S. (2003). What is an affordance? *Ecological Psychology*, 15(2):107–114.

[Kaelbling, 1989] Kaelbling, L. P. (1989). An architecture for intelligent reactive systems. In *Readings in planning*. Morgan Kaufmann.

[Kant, 1965] Kant, I. (1787/1965). *Critique of Pure Reason*. St. Martin's Press, New York.

[Karmiloff-Smith, 1992] Karmiloff-Smith, A. (1992). *Beyond Modularity: A Developmental Perspective on Cognitive Science*. MIT Press, Cambridge, Mass.

[Karp et al., 1995] Karp, P. D., Myers, K., and Gruber, T. (1995). The generic frame protocol. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 768–774, Montreal, Canada. International Joint Committee on Artificial Intelligence.

[Katz, 1972] Katz, J. (1972). *Semantic Theory*. Harper and Row, New York.

[Keil, 1989] Keil, F. C. (1989). *Concepts, Kinds, and Cognitive Development*. MIT Press, Cambridge, Mass.

[Kennedy, 2003] Kennedy, M. C. (2003). *Distributed Reflective Architectures for Anomaly Detection and Autonomous Recovery*. PhD thesis, School of Computer Science, The University of Birmingham.

[Kennedy and Sloman, 2002] Kennedy, M. C. and Sloman, A. (2002). Reflective architectures for damage tolerant autonomous systems. Technical Report CSR-02-1, School of Computer Science, The University of Birmingham.

[King, 1999] King, B. J., editor (1999). *The Origins of Language: What Nonhuman Primates Can Tell Us*. SAR Press, Santa Fe, New Mexico.

[Kintsch, 1974] Kintsch, W. (1974). *The representation of meaning in memory*. Lawrence Erlbaum, Hillsdale, NJ.

[Kirsch, 1991] Kirsch, D. (1991). Today the earwig, tomorrow man? *Artificial Intelligence*, 47:161–184.

[Kripke, 1959] Kripke, S. (1959). A completeness theorem in modal logic. *Journal of Symbolic Logic*, 24:1–24.

[Kripke, 1972] Kripke, S. (1972). *Naming and necessity*. Harwrad University Press, Cambridge, Mass.

[Laird et al., 1987] Laird, J. E., Newell, A., and Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, 33:1–64.

[Lakatos, 1995] Lakatos, I. (1995). The methodology of scientific rsearch programmes. In Worral, J. and Currie, G., editors, *Philosophical papers*, volume 1. Cambridge University Press.

[Langacker, 1987] Langacker, R. W. (1987). *Foundations of cognitive grammar, Vol. 1: Theoretical prerequisites*. Stanford University Press, Stanford, CA.

[Langley, 1996] Langley, P. (1996). *Elements of machine learning*. Morgan Kaufmann, San Francisco, CA.

[Langley and Laird, 2002] Langley, P. and Laird, J. E. (2002). Cognitive architectures: Research issues and challenges. Technical report, Institute for the Study of Learning and Expertise, Palo Alto, CA.

[Laurence and Margolis, 1999] Laurence, S. and Margolis, E. (1999). Concept and cognitive science. In [Margolis and Laurence, 1999].

[Levinson, 1999] Levinson, S. C. (1999). Covariation between spatial language and cognition and its implications for language learning. In [Bowerman and Levinson, 1999].

[Lewis, 1986] Lewis, D. K. (1986). *On the Plurality of Worlds*. Blackwell, Oxford.

[Locke, 1975] Locke, J. (1690/1975). *An Essay Concerning Human Understanding*. Oxford University Press, New York.

[Lynch et al., 2000] Lynch, E. B., Coley, J. D., and Medin, D. L. (2000). Tall is typical: Central tendency, ideal dimensions and graded category structure among tree experts and novices. *Memory and Cognition*, 28(1):41–50.

[MacGregor, 1991] MacGregor, R. M. (1991). Using a description classifier to enhance deductive inference. In *Proceedings Seventh IEEE Conference on AI Applications*, pages 141–147.

[Maes, 1988] Maes, P. (1988). Issues in computational reflection. In [Maes and Nardi, 1988], pages 21–35.

[Maes and Nardi, 1988] Maes, P. and Nardi, D., editors (1988). *Meta-Level Architectures and Reflection*. North-Holland.

[Malt and Johnson, 1992] Malt, B. C. and Johnson, E. C. (1992). Do artifacts concepts have cores? *Journal of Memory and Language*, 31:195–217.

[Margolis, 1999] Margolis, E. (1999). Concepts: core readings. In [Margolis and Laurence, 1999].

[Margolis and Laurence, 1999] Margolis, E. and Laurence, S., editors (1999). *Concepts: core readings*. MIT Press, Cambridge, Mass.

[Mark et al., 1999] Mark, L. S., Jiang, Y., King, S. S., and Paasche, J. (1999). The impact of visual exploration on judgements of whether a gap is crossable. *Journal of Experimental Psychology: Human Perception and Performance*, 25:287–295.

[Marr, 1982] Marr, D. (1982). *Vision. A Computational Investigation into the Human Representation and Processing of Visual Information*. W. H. Freeman, New York, NY.

[Matarić, 1997] Matarić, M. J. (1997). Behaviour-based control: examples from navigation, learning, and group behaviour. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(2-3):323–336.

[Matarić, 1998] Matarić, M. J. (1998). Behavior-based robotics as a tool for synthesis of artificial behavior and analysis of natural behavior. *Trends in Cognitive Sciences*, 2(3):82–87.

[McNeill et al., 2003] McNeill, F., Bundy, A., and Schorlemmer, M. (2003). Dynamic Ontology Refinement. In *Proceedings of ICAPS'03 Workshop on Plan Execution*, Trento, Italy.

[Mechelen et al., 1993] Mechelen, I. v., Hampton, J., Michalski, R. S., and Theuns, P., editors (1993). *Categories and concepts: Theoretical views and inductive data analysis*. Academic Press, London.

[Medin, 1989] Medin, D. (1989). Concepts and conceptual structure. *American Psychologist*, 44.

[Medin et al., 2000] Medin, D. L., Lynch, E. B., and O.Solomon, K. (2000). Are there kinds of concepts? *Annu. Rev. Psychol.*, 51:121–147.

[Meyer, 1995] Meyer, J. A. (1995). Artificial life and the animat approach to artificial intelligence. In Boden, M., editor, *Artificial Intelligence*. Academic Press.

[Miller and Laird, 1996] Miller, C. S. and Laird, J. E. (1996). Accounting for graded performance within a discrete search framework. *Cognitive Science*, 20:499–537.

[Minsky, 1986] Minsky, M. (1986). *The Society of Mind*. Simon and Schuster, New York.

[Minsky, 1990] Minsky, M. (1990). Logical vs. analogical or symbolic vs. connectionist or neat vs. scruffy. In Winston, P. H., editor, *Artificial Intelligence at MIT: Expanding Frontiers*. MIT Press, Cambridge, Mass.

[Mira and Prieto, 2001] Mira, J. and Prieto, A., editors (2001). *Connectionist Models of Neurons, Learning Processes, and Artificial Intelligence*, volume 2084 of *Lecture Notes in Computer Science*, 6th International Work-Conference on Artificial and Natural Neural Networks, IWANN 2001 Granada, Spain, June 13-15.

[Mitchell, 1997] Mitchell, T. M. (1997). *Machine Learning*. McGraw Hill, New York.

[Montague, 1974] Montague, R. (1974). *Formal philosophy*. Edited by R. H. Thomason. Yale University Press, New Haven, CT.

[Newell, 1973] Newell, A. (1973). You can't play 20 questions with nature and win: Projective comments on the papers of this symposium. In Chase, W. G., editor, *Visual information processing*. Academic Press, New York.

[Newell, 1990] Newell, A. (1990). *Unified Theories of Cognition*. Harvard University Press, Cambridge, Mass.

[Newell and Simon, 1990] Newell, A. and Simon, H. A. (1990). Computer science as empirical enquiry: symbols and search. In Boden, M. A., editor, *The philosophy of AI*, pages 105–132. MIT Press, Cambridge, Mass.

[Nilsson, 1998] Nilsson, N. (1998). *Artificial Intelligence: A New Synthesis*. Morgan Kaufmann.

[Petters, 2004] Petters, D. (2004). Simulating infant-carer relationship dynamics. In *Proceedings of the* AAAI-*2004 Spring Symposium*, Stanford. Americam Association for Artificial Intelligence.

[Popper, 1972] Popper, K. (1972). *Objective Knowledge*. Clarendon Press, Oxford.

[Putnam, 1962] Putnam, H. (1962). The analytic and the synthetic. In feigh, H. and G., M., editors, *Minnesota studies in the philosophy of science*, volume 3. University of Minnesota Press, Minneapolis.

[Putnam, 1970] Putnam, H. (1970). Is semantic possible? In Kiefer, H. and Munitz, M., editors, *Language, Belief and Metaphysics*, pages 50–63. State University of New York Press, New York.

[Putnam, 1975] Putnam, H. (1975). The meaning of meaning. In Gunderson, K., editor, *Language, Mind and Knowledge*. University of Minnesota Press, Minneapolis.

[Putnam, 1981] Putnam, H. (1981). *Reason, Truth, and History*. University of Minnesota Press, Minneapolis.

[Putnam, 1988] Putnam, H. (1988). *Representation and reality*. MIT Press, Cambridge, Mass.

[Quine, 1963a] Quine, W. O. V. (1963a). Two dogmas of empiricism. In [Quine, 1963b], pages 20–46.

[Quine, 1963b] Quine, W. V. O. (1963b). *From a logical point of view*. Harvard University Press, Cambridge, 2nd edition.

[Quinlan, 1986] Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1(1):81–106.

[Quinlan, 1994] Quinlan, J. R. (1994). *Comparing connectionist and symbolic learning methods*, volume I: Constraints and Prospects, pages 445–456. MIT Press.

[Rey, 1983] Rey, G. (1983). Concepts and stereotypes. *Cognition*, 15:237–262.

[Rips, 1989] Rips, L. J. (1989). Similarity, typicality and categorisation. In Stella Vosniadou, A. O., editor, *Similarity and Analogical Reasoning*, pages 21–59. Cambridge University Press, New York.

[Rosch, 1973] Rosch, E. (1973). On the internal structure of perceptual and semantic categories. In Moore, T., editor, *Cognitive development and the acquisition of language*, pages 111–144. Academic Press, New York.

[Rosch, 1978] Rosch, E. (1978). Principles of categorization. In Rosch, E. and Lloyds, B., editors, *Cognition and categorization*. Lawrence Erlbaum, Hillsdale, NJ.

[Rosch and Mervis, 1975] Rosch, E. and Mervis, C. (1975). Family resemblances: studies in the internal structure of categories. *Cognitive Psychology*, 7:573–605.

[Rosenbloom et al., 1988] Rosenbloom, P., Laird, J. E., and Newell, A. (1988). Meta-levels in SOAR. In [Maes and Nardi, 1988], pages 227–239.

[Russell and Norvig, 1995] Russell, S. J. and Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*. Prentice Hall, Englewood, New Jersey.

[Scheutz, 1999] Scheutz, M. (1999). When physical systems realize functions... *Minds and Machines*, 9(2):161–196.

[Scheutz, 2000] Scheutz, M. (2000). Surviving in a hostile multi-agent environment: How simple affective states can aid in the competition for resources. In *Proceedings of AI 2000: The Thirteenth Canadian Conference on Artificial Intelligence*, pages 389–399, New York. Springer-Verlag.

[Scheutz, 2001] Scheutz, M. (2001). The evolution of simple affective states in multi-agent environments. In Cañamero, D., editor, *Proceedings of the AAAI Fall Symposium 01*, pages 123–128, Falmouth, Mass. AAAI Press.

[Scheutz and Sloman, 2001] Scheutz, M. and Sloman, A. (2001). Affect and agent control: Experiments with simple affective states. In *Proceedings of IAT-01*, pages 200–209. World Scientific Publisher.

[Simon, 1982] Simon, H. A. (1982). *Models of bounded rationality*. MIT Press, Cambridge, Mass.

[Sloman, 1978] Sloman, A. (1978). *The Computer Revolution In Philosophy: Philosophy Science and Models of Mind*. Harvester Press.

[Sloman, 1989] Sloman, A. (1989). On designing a visual system (towards a gibsonian computational model of vision. *Journal of Experimental and Theoretical AI*, 1(4):289 – 337.

[Sloman, 1993] Sloman, A. (1993). The mind as a control system. In Hookway, C. and Peterson, D., editors, *Philosophy and the Cognitive Sciences*. Cambridge University Press.

[Sloman, 1995] Sloman, A. (1995). Exploring design space and niche space. In *SCAI-95: Proceedings of the 5th Scandinavian Conference on AI*, Trondheim, Sweden.

[Sloman, 1998a] Sloman, A. (1998a). Damasio, descartes, alarms and meta-management. In *Proceedings Symposium on Cognitive Agents: Modeling Human Cognition*, pages 2652 – 2657, IEEE International Conference on Systems, Man, and Cybernetics, San Diego.

[Sloman, 1998b] Sloman, A. (1998b). Semantic of evolution: trajectories and trade-offs in design space and niche space. In Coelho, H., editor, *Progress in Artificial Intelligence*, pages 27 – 38. Springer-Verlag, New York.

[Sloman, 1998c] Sloman, A. (1998c). What is artificial intelligence? Available at `http://www.cs.bham.ac.uk/research/cogaff/talks/whatsai.openday.pdf`.

[Sloman, 1999] Sloman, A. (1999). Artificial intelligence – an illustrative overview. Available at `http://www.cs.bham.ac.uk/~axs/courses/ai.html`.

[Sloman, 2000a] Sloman, A. (2000a). Interacting trajectories in design space and niche space: a philosopher speculates about evolution. In *Lecture Notes in Computer Science*, volume 1917, pages 3–16. Springer-Verlag, Berlin.

[Sloman, 2000b] Sloman, A. (2000b). Types of research in computing science, software engineering and artificial intelligence. Available at `http://www.cs.bham.ac.uk/~axs/misc/cs-research.html`.

[Sloman, 2001] Sloman, A. (2001). Varieties of affect and the cogaff architecture schema. Available at `http://www.cs.bham.ac.uk/research/cogaff/talks/gatsby.slides.pdf`.

[Sloman, 2002] Sloman, A. (2002). Architectures and the spaces they inhabit. Available at `http://www.cs.bham.ac.uk/research/cogaff/ibm02.html`.

[Sloman, 2005] Sloman, A. (2005). What the brain's mind tells the mind's eye. Available at `http://www.cs.bham.ac.uk/research/cogaff/sloman-vis-affordances.pdf`.

[Sloman et al., 2004] Sloman, A., Chrisley, R., and Scheutz, M. (2004). The architectural basis of affective states and processes. In Fellous, J. M. and Arbib, M. A., editors, *Who Needs Emotions?: The Brain Meets the Robot*. Oxford University Press, Oxford.

[Sloman and Logan, 1999] Sloman, A. and Logan, B. (1999). Building cognitively rich agents using the sim_agent toolkit. *Communications of the Association of Computing Machinery*, 43(2):71–77.

[Sloman and Poli, 1996] Sloman, A. and Poli, R. (1996). Sim_agent: A toolkit for exploring agent designs. In Wooldridge, Mike, J. M. and Tambe, M., editors, *Intelligent Agents Vol II (ATAL-95)*, pages 392–407. Springer-Verlag, New York.

[Sloman and Scheutz, 2002] Sloman, A. and Scheutz, M. (2002). A framework for comparing agent architectures. In *UKCI'02: Proceedings of the UK Workshop on Computational Intelligence*, Birmingham, UK.

[Smith, 1985] Smith, B. C. S. (1985). Prologue to reflection and semantics in a procedural language. In [Brachman and Levesque, 1985], pages 31–40.

[Smith and Medin, 1981] Smith, E. and Medin, D. (1981). *Categories and concepts*. Harvard University Press, Cambridge, Mass.

[Smith et al., 1984] Smith, E., Medin, D., and Rips, L. (1984). A psychological approach to concepts: Comments on rey's "concepts and stereotypes". *Cognition*, 17:265–274.

[Smith and Osherson, 1984] Smith, E. E. and Osherson, D. N. (1984). Conceptual combination with prototype concepts. *Cognitive Science: A Multidisciplinary Journal*, 8(4):337–361.

[Solomon et al., 1999] Solomon, K. O., Medin, D. L., and Lynch, E. B. (1999). Concepts do more than categorize. *Trends in Cognitive Sciences*, 3(3):March.

[Staab et al., 2000] Staab, S., Maedche, A., Nedellec, C., and Wiemer-Hastings, P., editors (2000). *Proceedings of the ECAI-2000 Workshop on Ontology Learning*, Berlin.

[Steels, 1995] Steels, L. (1995). When are robots intelligent autonomous agents? *Robotics and Autonomous Systems*, 15:3–9.

[Steels, 1999] Steels, L. (1999). *The Talking Heads Experiment. Volume 1. Words and Meanings*. Special pre-edition for LABORATORIUM, Antwerpen.

[Steels, 2000] Steels, L. (2000). The emergence of grammar in communicating autonomous robotic agents. In *Proceedings of the 14th European Conference on Artificial Intelligence*.

[Stoytchev, 2005] Stoytchev, A. (2005). Toward learning the binding affordances of objects: A behavior-grounded approach. In *Proceedings of AAAI Symposium on Developmental Robotics*, March 21-23, Stanford University.

[Talmy, 1988] Talmy, L. (1988). Force dynamics in language and cognition. *Cognitive science*, 12:49–100.

[Tarski, 1944] Tarski, A. (1944). The semantical concept of truth and the foundations of semantics. *Philosophy and Phenomenological Research*, 4.

[Thagard, 1996] Thagard, P. (1996). *Mind: Introduction to Cognitive Science*. MIT Press, Cambridge, Mass.

[Thompson, 1995] Thompson, E. (1995). Color vision, evolution, and perceptual content. *Synthese*, 104:1–32.

[Tomasello, 1992] Tomasello, M. (1992). *First verbs: a case study of early grammatical development*. Comabridge University Press, New York.

[Vauclair, 1996] Vauclair, J. (1996). *Animal Cognition*. Harward University Press, Cambridge, Mass.

[Walton, 1973] Walton, K. L. (1973). Linguistic relativity. In Pearce, G. and P., M., editors, *Conceptual Change*, pages 1–30. D. Reidel, Dordrecht, Holland.

[Warren, 1984] Warren, W. H. (1984). Perceiving affordances: Visual guidance of stair-climbing. *Journal of Experimental Psychology: Human Perception and Performance*, 10:683–703.

[Waxman and Markov, 1995] Waxman, S. R. and Markov, D. B. (1995). Words as invitaions to form categories: evidence from 12-months old infants. *Cognitive Psychology*, 29:257–302.

[Whorf, 1956] Whorf, B. (1956). *Language, Thought and Reality*. Cambridge University Press, Cambridge.

[Wijngaards et al., 2006] Wijngaards, N., Kempen, M., Smit, A., and Nieuwenhuis, K. (2006). Towards sustained team effectiveness. In Lindemann, G., editor, *Selected revised papers from the workshops on Agent, Norms and Institutions for Regulated Multi-Agent Systems (ANIREM) and Organizations and Organization Oriented Programming (OOOP) at AAMAS'05*, volume 3913 of *LNCS*, pages 33 – 45. Springer-Verlag, Berlin. In press.

[Winston, 1993] Winston, P. H. (1993). *Artificial Intelligence*. Addison-Wesley, Reading, Massachusetts, third edition.

[Wittgenstein, 1961] Wittgenstein, L. (1961). *Tractatus Logico-Philosophicus*. Kegan Paul, London.

[Wittgenstein, 1978] Wittgenstein, L. (1978). *Philosophical Investigations*. Oxford University Press, Oxford.

[Woodward and Markman, 1997] Woodward, A. L. and Markman, E. M. (1997). Early word learning. In Damon, W., Kuhn, D., and Siegler, R., editors, *Handbook of child psychology*, volume 2, pages 371–420. Wiley, New York.

[Wooldridge and Jennings, 1995] Wooldridge, M. and Jennings, N. R., editors (1995). *Intelligent Agents: Theories, Architectures, and Languages*. Springer-Verlag, New York.

Appendix A

The implemented system

The next Sections contains the code[1] that implements the system described in the thesis. The main program file is `learning4.p` where the agent architectures described in Chapter 5 are implemented (the reactive agent and the adaptive agent). File `isc.p` implements the learning algorithm; currently a version of the separate and conquer incremental learning for disjunctions presented in [Langley, 1996]. Finally, in file `setup-world.p` some procedures are defined that allow the user to create environments with various characteristics to be used to test the main system. All the program files contained in this Appendix are also available at `http://www.cs.bham.ac.uk/research/projects/cogaff/viezzer-thesis/code` and in the CD accompanying the thesis[2], where some examples of environment are also made available, together with the log files of some experiments done with the implemented system.

In order to run the program files contained in the next Sections, the Poplog system should be available. For more information on how to obtain Poplog see `www.poplog.org` or `www.cs.bham.ac.uk/research/poplog/freepoplog.html`. Once Poplog is installed and available, the files can be loaded and compiled through the VED editor, using the following instructions. For example, in the directory where the system files are stored, type the command

```
xved learning4.p
```

to start the graphical version of VED and load the main system file. Using the 'compile' menu available at the top of the VED window, compile the current file. This also creates an `output.p` file with the results of the compilation. Once `learning4.p` has been compiled, the procedures defined in it can be executed. The results of the execution will also be printed in the `output.p` file.

A.1 How to run a demonstration

In order to run a demonstration of the implemented system, the following commands, contained in the Test section at the beginning of `learning4.p`, must be executed[3]:

[1] The code is written in Pop11 and has been developped and tested with the Sussex Poplog Version 15.53.

[2] The thesis is itself available at `http://www.cs.bham.ac.uk/research/projects/cogaff/viezzer-thesis`.

[3] To execute a command, put the cursor at the beginning of the command line and using the 'compile' menu available at the top of the VED window, compile the current line.

- to run a demonstration with the reactive agent, execute the command

  ```
  demo_rea ('test_env.p', 1500);
  ```

  to create a demonstration window with the reactive agent and the environment specified in file `test_env.p`, and then to run the agent in the given environment for 1500 lifecycles; the results of the run will be printed in file `output.p`;

- to run a demonstration with the adaptive agent, execute the command

  ```
  demo_adap ('test_env.p', 1500);
  ```

  to create a demonstration window with the adaptive agent and the environment specified in file `test_env.p`, and then to run the agent in the given environment for 1500 lifecycles; the results of the run will be printed in file `output.p`.

The demonstration window (see Figure A.1) shows the agent with its visual field (a red agent is a reactive agent, while a green agent is an adaptive agent) and the agent's environment as a collection of objects represented as coloured dots: magenta dots represent Danger items, orange dots represent Treasure items, brown dots represent Food items and blu dots represents other object items, that are not relevant to the agent. The agent is located at the center of the window and it is facing to the right; this location is the agent's home while the limits of the agent's territory (beyond which the agent will not go) are represented by the large black circle close to the edges of the window.

The locations of the objects in the environment can be controlled by setting the value of the variable `ranseed` to a specified value (e.g. 12) with the command:

```
12 -> ranseed;
```

Two or more runs of a system demonstration with the same value of `ranseed` will produce the same results (and what is printed in the `output.p` file will also be the same, provided the same tracing methods have been used.) If dots appear with a different colour than those mentioned above, it means that two or more object items, possibly with different characteristics, overlap. If the user is willing to do so, before running the demonstration, objects can be moved around by clicking on them with the first mouse button and then dragging them. The agent can be moved around in the same way.

Once the commands to run the demo have been executed, the demo window will show the agent moving around within the limits of its territory: a beep signals that the agent has been acting upon a Danger item and has hurt itself.

In the case of the reactive agent, the default information printed out in the `output.p` file as a result of a system run says how many cycles the agent survived, how many objects were met, how many times the agent was hurt as a result of acting upon a dangerous item, how many attemps to recharge were made and how many were succesful, and finally how many attempts to collect treasures were made and how many were succesful. Further tracing can be produced using the tracing facilities provided by Pop11. In the case of the adaptive agent some predefined tracing is available which provides, for each lifecycle, information about the agent's visual field, its actions, and its knowledge of the objects in the environment; further information can also be produced using the tracing facilities provided by Pop11.

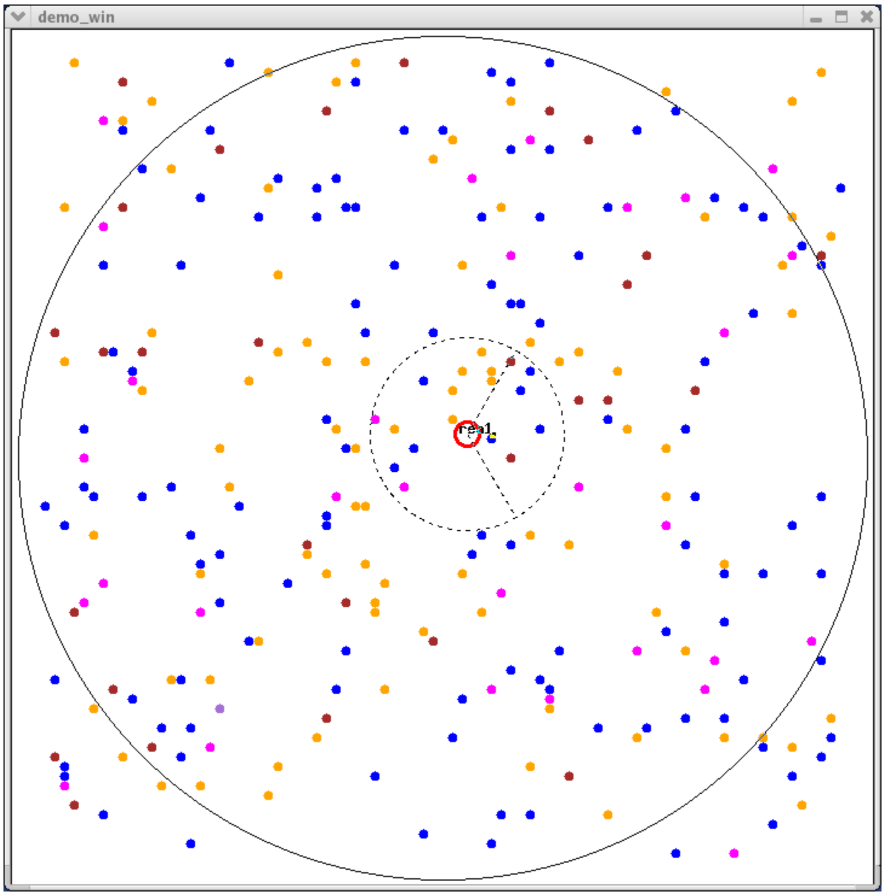


Figure A.1: A screenshot of the system demonstration window, with a reactive agent.

A.2 How to create a new environment

The aim of the procedures defined in file `setup-world.p` is to generate the object instances that populate the agent's environment, and to state the laws according to which these object instances are grouped into causal classes. Such causal classes can be viewed as the ontology of the designer. Each object instance is represented as a conjunction of attribute values. This information is accessible to the agent through its sensory module, and can become part of the input data to its learning module.

Creating a new environment involves the following steps:

- Choose the attributes that will characterise the objects, for example taste, colour and size; and choose, for each attribute, its possible values, for example, sweet and sour for the taste; red, blue and yellow for the colour, and small, medium and large for the size. In this case, there would be $2 * 3 * 3 = 18$ possible object types.

- Distinguish between relevant and irrelevant attributes, for example establish that only taste and size are relevant for characterising the causal classes. Following our previous example, this means that the most specific object types reduces to 6 (sweet and small, sweet and medium, etc., sour and small, sour and medium, etc.) and that the possible variants for each object type will depend on the irrelevant attributes; for example, if the signature of the objects of type1 is 'sweet and small' then all the possible variants of type1 objects are sweet, small and red; sweet, small and blue; sweet, small and yellow.

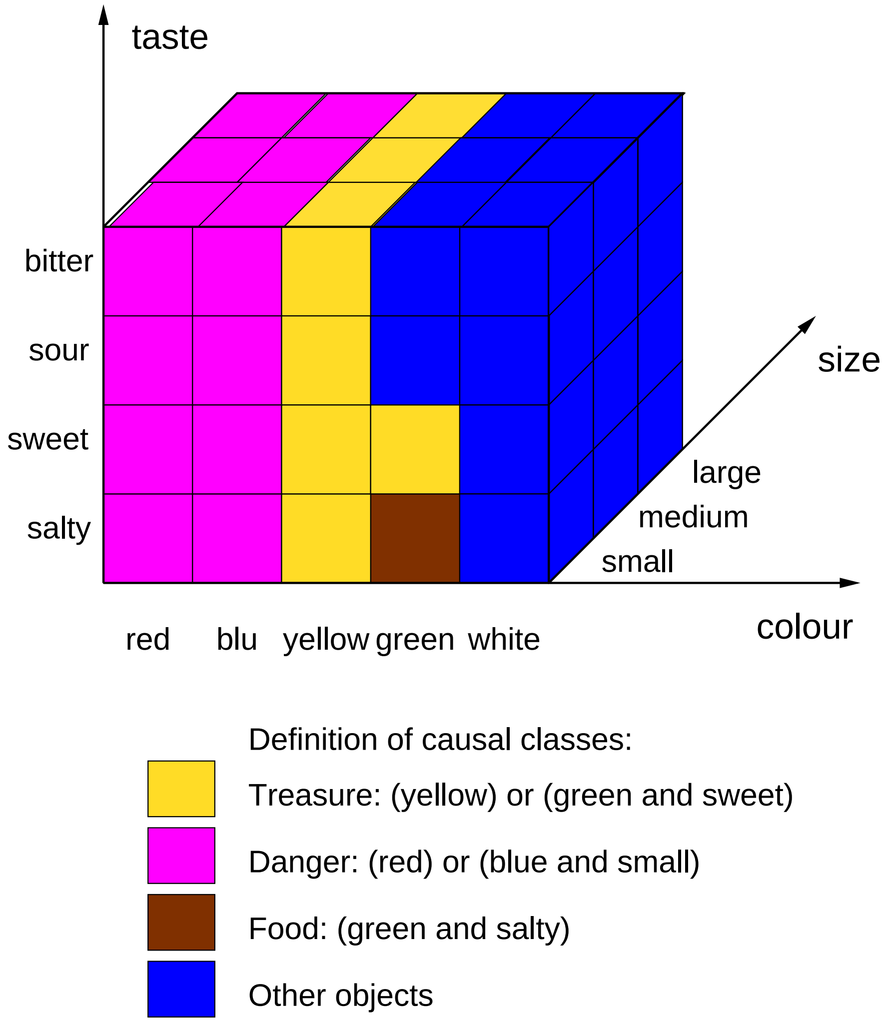


Figure A.2: The designer model specified in file `test_env.p`

- Establish which object type falls under which causal class, by choosing which values of the relevant attributes are admissible for each class; for example establish that food items are sweet and small or sour and small. Then choose in which proportion each causal class is to be represented.

- Establish how many objects will be created and assign a pair of spatial coordinates to each of them.

The environment specified in file `test_env.p` (see Figure A.2) has the following characteristics: the relevant object attributes are colour, size and taste, and there are no irrelevant attributes; the colour attribute has 5 possible values, namely red, blue, yellow, green and white; the size attribute has 3 possible values, namely small, medium and large; and the taste attribute has 4 possible values, namely salty, sweet, sour and bitter. There are in total 240 objects in the environment, of which 80 are treasure items, 60 are danger items and 50 are food items; the remaining 50 object are irrelevant to the agent. Treasure items are either yellow or green and sweet, danger items are either red or blue and small, and food items are green and salty.

If the user is willing to do so, different environments, with newly specified characteristics, can be created using some of the procedures defined in file `setup-world.p`. As an example of how an environment can be created, we give, in the following paragraphs, the full details of the commands that have been executed in order to create file `test_env.p`.

Once `setup-world.p` has been loaded and compiled, execute

```
setup_with_types ([relevant [colour Re Bl Ye Gr Wh]
                            [size S M L]
                            [taste salty sweet sour bitter]
                  irrelevant],
                 "test_env_aux1");
```

The procedure will generate the 60 possible object types determined by the attribute values given as input. The user will then be asked to choose which object type falls under which category. At the first prompt, enter the following types for the category Treasure:

```
 t25 t26 t27 t28 t29 t30 t31 t32 t33 t34 t35 t36 t38 t42 t46
```

This will define treasure items to be either yellow or green and sweet.At the second prompt, enter the following types for the category Danger:

```
 t1 t2 t3 t4 t5 t6 t7 t8 t9 t10 t11 t12 t13 t14 t15 t16
```

This will define danger items to be either red or blue and small. Finally, at the third prompt, enter the following types for the category Food:

```
 t37 t41 t45
```

This will define food items to be green and salty.

The output of `setup_with_types` is a file called `test_env_aux1.p` which will constitute the input to the next command:

```
do_instances_with_info('test_env_aux1.p',240,"test_env_aux2");
```

The output of `do_instances_with_info` is the file `test_env_aux2.p` containing the signatures of all the object items that will constitute the environment. 240 is the total number of object items that are created; at the prompt enter the following fractionals to specify in which proportions objects will be treasure items, danger items and food items respectively:

```
80_/240 60_/240 50_/240
```

where for example `80_/240` means 80 items out of 240.

The last command takes as input file `test_env_aux2.p` and formats it according to what is needed to the procedures `demo_rea` and `demo_adapt` defined in file `learning4.p`; the output is file `test_env.p`:

```
insert_names_with_info('test_env_aux2.p',"test_env");
```

A.3 The main program file

```
;;; -------------------------------------------------------------------------
;;; Section 0 - Tests

/*
;;; Setting the value of ranseed:

12 -> ranseed;

;;; Running a demo with the reactive agent:

demo_rea ('test_env.p', 1500);
demo_rea ('test_env.p', false);

;;; Running a demo with the adaptive agent:

demo_adap ('test_env.p', 1500);
demo_adap ('test_env.p', false);

*/

;;; -------------------------------------------------------------------------
;;; Section 1 - Libraries

uses newkit
uses sim_agent
uses sim_geom

;;; Graphical facilities in the RCLIB library
uses rclib

;;; See help rc_control_panel
uses rc_buttons
uses rc_linepic
uses rc_control_panel


;;; Extend sim_agent with graphical classes and methods
uses sim_picagent;
uses sim_harness;


compilehere
./isc.p                  ;;; the learning algorithm
;

;;; -------------------------------------------------------------------------
;;; Section 2 - Global variables

;;; Variables for the demo

vars my_agent; ;;; if 1 then reactive agent, if 2 then adaptive agent

vars rea_obj_met = 0,
```

```
     rea_times_hurt = 0,
     rea_tried_recharge = 0,
     rea_succ_recharge = 0,
     rea_tried_collect = 0,
     rea_succ_collect = 0;

vars dead_status = false;

define reset_demo_counters ();
    0 -> rea_obj_met;
    0 -> rea_times_hurt;
    0 -> rea_tried_recharge;
    0 -> rea_succ_recharge;
    0 -> rea_tried_collect;
    0 -> rea_succ_collect;
enddefine;

define myprint_rea_end_demo ();
    if dead_status then
      pr ('\n Agent is dead. \n');
    else
      pr ('\n Agent is alive. \n');
    endif;
    pr ('\n Number of objects met is ' >< rea_obj_met >< ' \n');
    pr ('\n Number of times hurt is ' >< rea_times_hurt >< '\n');
    pr ('\n Attempts to recharge ' >< rea_tried_recharge >< ' \n');
    pr ('\n Successful recharging ' >< rea_succ_recharge >< ' \n');
    pr ('\n Attempts to collect treasure ' >< rea_tried_collect >< ' \n');
    pr ('\n Successful collecting ' >< rea_succ_collect >< ' \n');
enddefine;

;;; Other variables

vars g_color_demo_static = 'black',
     g_color_rea_agent = 'red',
     g_color_adapt_agent = 'green',
     g_rea_vision_range = 80,
     g_rea_vision_degree = 60,
     g_rea_max_dist = 350,
     g_rea_speed = 5,
     g_rea_max_speed = 20,
     g_rea_init_head = 0,
     g_rea_init_happ = 10,
     g_rea_init_pain = 0, ;;; must be 0
     g_rea_init_en = 15000,

     g_out_of_energy = 0,
     g_pain_threshold = 0,
     g_hunger_threshold = 9000,

     g_rea_boredom_threshold = 1,
     g_adapt1_boredom_threshold = 1,
     ;

;;; the number of instances retained in memory by isc
vars g_isc_parameter = 7;
```

```
vars demo_win;

;;; the two agents
vars manu, lola;

;;; the things in the environment
vars all_objs;

;;; Control of the tracing messages via keywords.
/*
[body_state
 int_exp
 know
 reason
 gen_trace]
*/
[] -> prb_sayif_trace;

;;; setting the variables that control the tracing
vars
    sim_pause_demo = false,
    demo_check_popready = false,
    demo_cycle_pause = false,
    sim_delay = 5;

vars prb_chatty =
false;
;;; 5;
;;; 5*17;
;;; true;
/*
constants controlling prb_chatty
lconstant
    INSTANCES = 2,
    WHERETESTS = 3,
    DATABASE = 5,
    APPLICABILITY = 7,
    APPLICABLE = 11,
    DATABASE_CHANGE = 13,
    SHOWRULES = 17,
    TRACE_WEIGHTS = 19,
*/

;;; -----------------------------------------------------------------------
;;; Section 3 - Classes

;;; Spatial  things

define :mixin spatial_thing;
        is sim_movable,
           rc_rotatable_picsonly;
    ;;; sim_x and sim_y inherited from sim_movable in LIB sim_picagent
    slot min_dist = undef;
    slot rc_pic_lines = undef;
enddefine;
```

```
/*
A spatial thing can be  approximated to a circular body  with
radius equal to min_dist. Whenever something is as close  as
min_dist to a spatial thing, then it is in contact with  the
surface of that thing.
*/

;;; Demo statics -  spatial things with  no internal  processing
;;; mechanism
;;; Demo statics have features, which determine how they will interact
;;; with an agent if the agent performs an action upon them
;;; Demo statics represent objects as conjunctions of features

define :class demo_static;
    is spatial_thing;
    slot sim_name = nullstring >< gensym("s");  ;;; s for static
    ;;; slot sim_rulesystem == [];
    ;;; slot sim_sensors == [];
    slot features = undef;
    ;;; the only other features inherited from spatial_thing are
    ;;; - a location (sim_x and sim_y)
    ;;; - a body dimension (min_dist)
    slot object_type = undef;
    slot bodycolor = g_color_demo_static;
enddefine;

;;; Simple agents - spatial things with a simple internal
;;; processing mechanism, defined in the rulesystem

define :rulesystem rea_rulesystem;  ;;; defined in the next section
enddefine;

define :class reactive_agent;
    is spatial_thing, sim_object;
    slot sim_name = nullstring >< gensym("rea");    ;;; rea for reactive
    slot sim_rulesystem = rea_rulesystem;
    ;;; slot sim_sensors = [];
    slot sim_status = undef;    ;;; can be "OK","bored","dead","hungry"
                                ;;; or "hurt"
    slot local_cycle = 0;       ;;; to count the agent internal time

    ;;; external sensor slots
    slot vision_range = g_rea_vision_range;      ;;; set to 80
    slot vision_degree = g_rea_vision_degree;    ;;; in degrees - it is half
                                                 ;;; of the
                                                 ;;; angle of the visual field
                                                 ;;; set to 60
    ;;; the agent home
    slot orig_x = 0;
    slot orig_y = 0;
    slot max_dist = g_rea_max_dist;    ;;; how far from home it can go,
                                       ;;; set to 350

    ;;; motion related slots
    slot speed = g_rea_speed;          ;;; set to 5
```

```
    slot max_speed = g_rea_max_speed;  ;;; set to 20
    slot heading = g_rea_init_head;    ;;; set to 0

    ;;; internal sensors
    slot happiness = g_rea_init_happ;       ;;; increase when collecting treasure
    slot pain = g_rea_init_pain;            ;;; increase when acting upon dangerous
                                            ;;; objects
    slot energy = g_rea_init_en;            ;;; increase when collecting energy
                                            ;;; decrease when moving
    slot no_motion = 0;                     ;;; number of cycles the agent
                                            ;;; has not been moving
    ;;; slot updated = false;

    slot learning_db = prb_newdatabase (sim_dbsize, []);
    ;;; this is the internal database where the data used by the
    ;;; classification algorithm are stored
    ;;; the data have form
    ;;; [<category name> <positive/negative> [<feature list>]]

    slot model_db = prb_newdatabase (sim_dbsize, []);
    ;;; this is the internal db where the
    ;;; [isa PREDICATE [<feature list>]]
    ;;; data are stored; this db can be viewed as the backbone
    ;;; ontology of the agent's world model

    slot bodycolor = g_color_rea_agent;
enddefine;

define :rulesystem adaptive1_rulesystem;  ;;; defined in the next section
enddefine;

define :class adaptive_agent;
    is spatial_thing, sim_object;
    slot sim_name = nullstring >< gensym("adapt");    ;;; adapt for adaptive
    slot sim_rulesystem = adaptive1_rulesystem;
    ;;; slot sim_sensors = [];
    slot sim_status = undef;    ;;; can be "OK","bored","dead","hungry"
                                ;;; or "hurt"
    slot local_cycle = 0;       ;;; to count the agent internal time

    ;;; external sensor slots
    slot vision_range = g_rea_vision_range;      ;;; set to 80
    slot vision_degree = g_rea_vision_degree;    ;;; in degrees - it is half
                                                 ;;; of the
                                                 ;;; angle of the visual field
                                                 ;;; set to 60
    ;;; the agent home
    slot orig_x = 0;
    slot orig_y = 0;
    slot max_dist = g_rea_max_dist;    ;;; how far from home it can go,
                                       ;;; set to 350

    ;;; motion related slots
    slot speed = g_rea_speed;          ;;; set to 5
    slot max_speed = g_rea_max_speed;  ;;; set to 20
    slot heading = g_rea_init_head;    ;;; set to 0
```

```
    ;;; internal sensors
    slot happiness = g_rea_init_happ;       ;;; increase when collecting
                                            ;;; treasure items
    slot pain = g_rea_init_pain;            ;;; increase when acting upon
                                            ;;; danger items
    slot energy = g_rea_init_en;            ;;; increase when collecting energy
                                            ;;; decrease when moving
    slot no_motion = 0;                     ;;; number of cycles the agent
                                            ;;; has not been moving
    slot just_turned = false;

    slot learning_db = prb_newdatabase (sim_dbsize, []);
    ;;; this is the internal database where the data used by the
    ;;; classification algorithm are stored

    slot model_db = prb_newdatabase (sim_dbsize, []);
    slot old_model_db = prb_newdatabase (sim_dbsize, []);
    ;;; this is the internal db where the
    ;;; [isa PREDICATE [<feature list>]]
    ;;; data are stored; this db can be viewed as the backbone
    ;;; ontology of the agent's world model

    slot bodycolor = g_color_adapt_agent;
enddefine;

/*
An agent has something like a home, the place with
coordinates orig_x and orig_y, and at most will go as
further away as max_dist from its home.
*/

;;; ----------------------------------------------------------------------------
;;; Section 4 - Methods and utilities

;;; Creation methods

define new_demo_static (x, y, r, feature_list, t) -> obj;
    instance demo_static;
        sim_x = x;
        sim_y = y;
        min_dist = r;
        features = feature_list;
        object_type = t;
    endinstance -> obj;
    lvars the_colour;
    if object_type(obj) = 'Trea' then 'orange' -> the_colour;
    elseif object_type(obj) = 'Dan' then 'magenta' -> the_colour;
    elseif object_type(obj) = 'En' then 'brown' -> the_colour;
    else 'blue' -> the_colour;
    endif;
    ;;; see help xcolours
    ;;; see help rc_linepic
    [WIDTH 2 [rc_draw_blob {0 0 ^(min_dist(obj)) ^the_colour}]]
        -> rc_pic_lines(obj);
enddefine;
```

```
define new_reactive_agent (x, y, r) -> ag;
    instance reactive_agent;
        sim_x = x;
        sim_y = y;
        min_dist = r;
    endinstance -> ag;

    conspair([WIDTH 1 COLOUR 'black' STYLE ^LineOnOffDash
            [ANGLE 60 {0 0} {80 0}]
            [ANGLE 300 {0 0} {80 0}]
            [CIRCLE {0 0 80}]],
        [[WIDTH 3 COLOUR ^(bodycolor(ag))
                [CIRCLE {0 0 ^(min_dist(ag))}]
                [COLOUR 'black' [SQUARE {20 0 2}]]]]]) -> rc_pic_lines(ag);
    [{-6 -0 ^(sim_name(ag))}] -> rc_pic_strings(ag);
enddefine;

define new_adaptive_agent (x, y, r) -> ag;
    instance adaptive_agent;
        sim_x = x;
        sim_y = y;
        min_dist = r;
    endinstance -> ag;

    conspair([WIDTH 1 COLOUR 'black' STYLE ^LineOnOffDash
            [ANGLE 60 {0 0} {80 0}]
            [ANGLE 300 {0 0} {80 0}]
            [CIRCLE {0 0 80}]],
        [[WIDTH 3 COLOUR ^(bodycolor(ag))
                [CIRCLE {0 0 ^(min_dist(ag))}]
                [COLOUR 'black' [SQUARE {20 0 2}]]]]]) -> rc_pic_lines(ag);
    [{-6 -0 ^(sim_name(ag))}] -> rc_pic_strings(ag);
enddefine;

;;; Redefined printing method for sim_object instances

define :method print_instance (item:sim_object);
    printf(
        '<object name:%P x_coord:%P y_coord:%P>',
        [% sim_name(item), sim_x(item), sim_y(item) %])
enddefine;

;;; Redefined tracing methods

define sim_scheduler_pausing_trace (objects, cycle);
    if my_agent == 1 then
    ;;; if reactive agent then do not print anything
    else pr('\n====================== end of cycle ' >< cycle ><
            ' ==================\n');
    endif;
enddefine;

define :method sim_agent_rulefamily_trace(object:sim_object, rulefamily);
    ;;; do nothing
enddefine;
```

```
define :method sim_agent_rulefamily_trace(object:reactive_agent, rulefamily);
    ;;; do nothing
enddefine;

define :method sim_agent_rulefamily_trace(object:adaptive_agent, rulefamily);
    ;;; do nothing
enddefine;

define :method sim_agent_endrun_trace(object:sim_object);
    ;;; do nothing
enddefine;

define :method sim_agent_endrun_trace(object:reactive_agent);
    ;;; do nothing
enddefine;

define :method sim_agent_endrun_trace(object:adaptive_agent);
    prb_print_table( sim_get_data(object) );
enddefine;

define :method sim_agent_actions_out_trace(object:sim_object);
    ;;; do nothing
enddefine;

define :method sim_agent_actions_out_trace(object:demo_static);
    ;;; do nothing
enddefine;

define :method sim_agent_action_trace(object:sim_object);
    ;;; do nothing
enddefine;

;;; Perception utilities

;;; Compute and return the distance between two objects

define :method sim_distance(o1:sim_object, o2:sim_object);
    sqrt((o1.sim_x - o2.sim_x)**2 + (o1.sim_y - o2.sim_y)**2)
enddefine;

;;; Check if something is in visual range - returns a truth
;;; value

;;; dist is the distance between the agent and the obj
;;; range is the agent's vision range
;;; dir1 is the agent's heading, in degree
;;; dir2 is sim_heading_from applied to the agent and the object
;;; dev is the agent's vision degree

define in_range(dist,range,dir1,dir2,dev);
    if dir1 < dev and dist <= range then
        dir2 <= dir1 + dev or 360 - (dev - dir1) <= dir2
    elseif dir1 > 360 - dev and dist <= range then
        dir1 - dev <= dir2 or dir2 <= dir1 + dev - 360
    else
```

```
        dist <= range and dir1 - dev <= dir2 and dir2 <= dir1 + dev
    endif;
enddefine;

;;; Find the nearest thing in the visual field
;;; to be used by the reactive agent

define rea_find_nearest (agent) -> (dir, dist);
    dlocal prb_database = sim_data(agent);
    lvars vlist, info, item, best = vision_range(agent), new_dist;

    ;;; get a list of things in the visual field
    prb_collect_values(![?info [thing ??info]]) -> vlist;
    ;;; vlist is a list of form [[dir1 dist1 type1 features1]
    ;;;                          [dir2 dist2 type2 features2] ...]
    for item in vlist do
        item(2) -> new_dist;
        if new_dist < best then
            new_dist -> best;
            new_dist -> dist;
            item(1) -> dir;
        endif;
    endfor;
enddefine;

;;; Find the nearest relevant thing in the visual field
;;; to be used by the adaptive agent

define find_nearest (agent) -> (nearest, obstacles);
    dlocal prb_database = sim_data(agent);
    lvars glist, info1, vlist, info2, item,
        best = vision_range(agent), new_dist, partial, partial1;

    ;;; get a list of relevant things in visual field
    ;;; relevant things are potential targets
    ;;; they are represented as [good_one <dir> <dist> <type> <features>]
    ;;; and were produced by the procedure find_best
    prb_collect_values(![?info1 [good_one ??info1]]) -> glist;
    ;;; glist is a list of form [[dir1 dist1 type1 features1]
    ;;;                          [dir2 dist2 type2 features2] ...]

    if length(glist) = 1 then [%[target]<>hd(glist)%] -> nearest;
        [TRACE set the only good one as target] ==>

    else
        ;;; get the nearest one in glist
        for item in glist do
            item(2) -> new_dist;
            if new_dist < best then
                new_dist -> best;
                item -> partial1;
            endif;
        endfor;
        [% [target]<>partial1 %] -> nearest;
        [TRACE set the nearest good one as target] ==>
    endif;
```

```
    [TRACE]<>nearest ==>

    ;;; get a list of things in the visual field
    prb_collect_values(![?info2 [thing ??info2]]) -> vlist;
    ;;; vlist is a list of form [[dir1 dist1 type1 features1]
    ;;;                          [dir2 dist2 type2 features2] ...]

    ;;; remove nearest from vlist, and label all the other things as
    ;;; obstacles

    delete (tl(hd(nearest)), vlist) -> partial;
    [% for item in partial do
            [obst]<>item;
        endfor;
    %] -> obstacles;
    ;;; [TRACE obstacles is] ==>
    ;;; obstacles ==>

enddefine;

define help_find_best1 (thing_list, dan_constr) -> best;
    lvars item, features;
    [% for item in thing_list do
            item(4) -> features;
            ;;; the procedure sublist is defined in isc.p
            ;;; sublist (list1, list2) check whether list1 is a
            ;;; sublist of list2
            if not(sublist (dan_constr, features)) then item; endif;
    endfor; %] -> best;
enddefine;

define help_find_best2 (thing_list, constr) -> best;
    lvars item, features, partial;
    [% for item in thing_list do
            item(4) -> features;
            ;;; the procedure sublist is defined in isc.p
            ;;; sublist (list1, list2) check whether list1 is a
            ;;; sublist of list2
            if sublist (constr, features) then item; endif;
    endfor; %] -> best;
enddefine;


define find_best (agent, constr, dan_constr) -> (best);
    dlocal prb_database = sim_data(agent);
    lvars thing_list, item, new_dist, info,
        best_dist = vision_range(agent), partial, partial1,partial2,
        it1, it2;

    ;;; get a list of things in the visual field
    prb_collect_values(![?info [thing ??info]]) -> thing_list;
    ;;; vlist is a list of form [[dir1 dist1 type1 features1]
    ;;;                          [dir2 dist2 type2 features2] ...]

    ;;; Case1: the agent has no concept for the DANGER category
    ;;; and the agent has no concept for the relevant category (the kind
```

```
;;; of things it is looking for.)

if constr == [] and dan_constr == [] then
    for item in thing_list do
        item(2) -> new_dist;
        if new_dist < best_dist then
            new_dist -> best_dist;
            item -> it1;
        endif;
    endfor;
    [% [target]<>it1 %] -> best;
    [TRACE no danger concept no relevant concept] ==>
    [TRACE set nearest object as target] ==>
    [TRACE]<>best ==>

else
    ;;; Case2: the agent has a concept of DANGER but no concept of the
    ;;; relevant category
    if constr == [] then
        [TRACE danger concept but no relevant concept] ==>

        ;;; consider a potential target everything which is not a DANGER
        help_find_best1 (thing_list, dan_constr) -> partial1;

        ;;; Case3: the agent has no concept of DANGER but it has
        ;;; a concept of the relevant category
    elseif dan_constr == [] then
        [TRACE relevant concept but no danger concept] ==>

        ;;; consider a potential target everything which belongs to
    ;;; the relevant category

        help_find_best2 (thing_list, constr) -> partial1;

        ;;; Case4: the agent has both a concept of DANGER and a concept
        ;;; of the relevant category; hence, consider a potential target
        ;;; everything which belongs to the relevant category and is not
    ;;; a DANGER

    else help_find_best1 (thing_list, dan_constr) -> partial;
        if partial == [] then
            ;;; [] -> partial1;
            help_find_best2(thing_list, constr) -> partial1;
        else
            help_find_best2 (partial, constr) -> partial2;
            if partial2 == [] then partial -> partial1;
            else partial2 -> partial1;
            endif;
        endif;
    endif;
    if partial1 == [] then
        for item in thing_list do
            item(2) -> new_dist;
            if new_dist < best_dist then
                new_dist -> best_dist;
                item -> it2;
```

```
                endif;
            endfor;
            [% [target]<>it2 %] -> best;
            [TRACE no good one was found] ==>
            [TRACE set nearest object as target] ==>
            [TRACE]<>best ==>
        else
            [% for item in partial1 do
                    [good_one]<>item;
                endfor; %] -> best;
            [TRACE good ones are] ==>
            [TRACE]<>best ==>
        endif;
    endif;
enddefine;

;;; Redefined sim_run_sensors method for demo_static

define :method sim_run_sensors (object:demo_static, entities)
        -> sensor_data;
    ;;; do nothing
    [] -> sensor_data;
enddefine;

;;; Redefined sim_run_sensors method for simple_moving_agent

define :method sim_run_sensors (agent:spatial_thing, entities)
;;; define :method sim_run_sensors (agent:adaptive_agent, entities)
        -> sensor_data;
[] -> sensor_data;
returnif (sim_status(agent) == "dead");
    ;;; lvars sensor_data = [];
    ;;; false -> updated(agent);

    lvars entity,
        vision_rng = vision_range(agent),
        vision_dg = vision_degree(agent),
        vision_items = [],
        ag_x = sim_x(agent),
        ag_y = sim_y(agent),
        ag_head = heading(agent);

    for entity in entities do
        lvars dist1 = sim_distance (agent, entity),
              dist2 = dist1 - min_dist(entity);
        unless (entity == agent or dist2 > vision_range (agent)) then
            lvars ent_x = sim_x(entity),
                ent_y = sim_y(entity),
                dir = sim_heading_from (ag_x, ag_y, ent_x, ent_y),
                ent_features = features (entity),
                ent_type = object_type (entity);

            if in_range (dist2, vision_rng, ag_head, dir, vision_dg) then
                ;;; this will determine which data are in the
                ;;; [thing ...] items in the agent database (which
                ;;; informations that agent gets about the thing)
```

```
                ;;; one possibility is: direction and distance

                ;;; this will make the agent stop at the border of the
                ;;; thing, but then I have problems because nothing prevents
                ;;; the agent walk through the thing
                ;;; so right now it is better to assume that agent and things
                ;;; are more or less of the same size, and can be approximated
                ;;; to points

                conspair ([thing %dir% %dist2% %ent_type% %ent_features%],
                         vision_items)
                    -> vision_items;
                ;;; 'd from object is'=>
                ;;; dist2=>
            endif;
        endunless;
    endfor;
    ;;; store data in agent
    [[actual_level energy %energy(agent)%]
     [actual_level happiness %happiness(agent)%]
     [actual_level pain %pain(agent)%]
     [heading %heading(agent)%]
     [speed %speed(agent)%]
    ]
    <> vision_items -> sensor_data;
enddefine;


;;; Utilities for the rulesystem

;;; This can be used in the learning_data_ruleset, to build the
;;; learning database of the agent: it checks that the item
;;; is not already present in the database, before adding it.
;;; This means that the database containing the learning data
;;; does not contains duplicates.

define sim_testadd_data (item, prb_database);
    ;;; make this locally global for the procedures used below:
    dlocal prb_database;

    unless prb_instance_present(item) then
        prb_add(item);
    endunless;

enddefine;

;;; These are used in the choose_action_ruleset

define :method choose_new_speed (ag:spatial_thing, dist);
;;; define :method choose_new_speed (ag:adaptive_agent, dist);
    if dist == false then 30;
;;; to be sure that it will be on the other side of the object

    elseif dist > 30 then ag.max_speed;
;;;        'accelerate' => ;
    else 5;
```

```
;;; this means that 10 <= dist <= 30

;;;        'approaching target, reduce speed' => ;
    endif;
enddefine;

define new_position (ag_x, ag_y, heading, speed) -> (new_x, new_y);
    (speed * cos(heading) + ag_x) -> new_x;
    (speed * sin(heading) + ag_y) -> new_y;
enddefine;

define :method choose_new_heading (ag:spatial_thing) -> result;
;;; define :method choose_new_heading (ag:adaptive_agent) -> result;
    lvars new_x, new_y, new_sign, to_add, d_from_home;
    new_position (ag.sim_x, ag.sim_y, ag.heading, ag.speed) -> (new_x, new_y);
;;;    'ag speed is'=>
;;;    ag.speed =>
;;;    'ag new x will be'=>
;;;    new_x=>
;;;    'ag new y will be'=>
;;;    new_y=>
    sim_distance_from (new_x, new_y, ag.orig_x, ag.orig_y) -> d_from_home;
;;;    'distance from home is' =>
;;;    d_from_home =>
    if d_from_home < ag.max_dist then
        ag.heading -> result;
        false -> ag.just_turned;
    else
        if ag.just_turned = true then ag.heading -> result;
        else
            true -> ag.just_turned;
            sim_normalise_angle (ag.heading + 180) -> result;
            ;;; random(2) -> new_sign;
            ;;; random(180) -> to_add;
            ;;; 0 -> to_add;
            ;;; if new_sign = 1 then
            ;;;     sim_normalise_angle (ag.heading + 180 + to_add);
            ;;; else sim_normalise_angle (ag.heading - (180 + to_add));
;;;           'too far from home, go back' => ;
        endif;
    endif;
enddefine;


define :method choose_new_heading1 (ag:reactive_agent) -> result;
    lvars new_x, new_y, new_sign, to_add, d_from_home;
    new_position (ag.sim_x, ag.sim_y, ag.heading, ag.speed) -> (new_x, new_y);
    sim_distance_from (new_x, new_y, ag.orig_x, ag.orig_y) -> d_from_home;
    if d_from_home < ag.max_dist then
        ag.heading -> result;
    else sim_heading_from (ag.sim_x, ag.sim_y, ag.orig_x, ag.orig_y) -> result;
    ;;; if too far from home, go back towards home
    ;;;    [TRACE too far from home, go back towards home] =>;
    endif;
enddefine;
```

```
define :method choose_new_heading1 (ag:adaptive_agent) -> result;
    lvars new_x, new_y, new_sign, to_add, d_from_home;
    new_position (ag.sim_x, ag.sim_y, ag.heading, ag.speed) -> (new_x, new_y);
    sim_distance_from (new_x, new_y, ag.orig_x, ag.orig_y) -> d_from_home;
    if d_from_home < ag.max_dist then
        ag.heading -> result;
    else sim_heading_from (ag.sim_x, ag.sim_y, ag.orig_x, ag.orig_y) -> result;
    ;;; if too far from home, go back towards home
    ;;;    [TRACE too far from home, go back towards home] =>;
    endif;
enddefine;

;;; These are used in the do_action_ruleset

define :method move_agent (ag:spatial_thing, ag_heading, ag_speed);
;;; define :method move_agent (ag:adaptive_agent, ag_heading, ag_speed);
    new_position(ag.sim_x, ag.sim_y, ag_heading, ag_speed);
enddefine;

define :method update_energy (ag:spatial_thing, ag_speed);
;;; define :method update_energy (ag:adaptive_agent, ag_speed);
    ag.energy - ag_speed**2/5 - 3 -> ag.energy;
    ;;; ag.energy - ag_speed*3 - 8 -> ag.energy;
enddefine;

define :method recharge (ag:spatial_thing, obj_type);
    ;;; define :method recharge (ag:adaptive_agent, obj_type);
    rea_obj_met +1 -> rea_obj_met;
    rea_tried_recharge +1 -> rea_tried_recharge;
    if obj_type = 'En' then ag.energy + 6000 -> ag.energy;
    rea_succ_recharge +1 -> rea_succ_recharge;
    elseif obj_type = 'Dan' then
        10 -> ag.pain;
        ;;; make the agent beep when hurt
        vedscreenbell();
        rea_times_hurt +1 -> rea_times_hurt;
    endif;
enddefine;

define :method get_treasure (ag:spatial_thing, obj_type);
    ;;; define :method get_treasure (ag:adaptive_agent, obj_type);
    rea_obj_met +1 -> rea_obj_met;
    rea_tried_collect +1 -> rea_tried_collect;
    if obj_type = 'Trea' then ag.happiness + 10 -> ag.happiness;
    rea_succ_collect +1 -> rea_succ_collect;
    elseif obj_type = 'Dan' then
        10 -> ag.pain;
        ;;; make the agent beep when hurt
        vedscreenbell();
        rea_times_hurt +1 -> rea_times_hurt;
    endif;
enddefine;


;;; ------------------------------------------------------------------------
```

```
;;; Section 5A - Reactive agent architecture (rea_rulesystem)


define :rulesystem rea_rulesystem;
    cycle_limit = 1;
    debug = true;
    with_interval = 1;

include: rea_evaluate_internal_status_ruleset
include: rea_evaluate_expectation_ruleset
include: rea_learning_data_ruleset
include: rea_plan_action_ruleset
include: rea_select_target_ruleset
include: rea_choose_action_ruleset
   ;;; set [action_link happiness collect_treasure]
   ;;; set [action_link energy collect_energy]
include: rea_internal_language_ruleset1
   ;;; set [pred_link <sensor_name> <pred_symbol>]
include: rea_internal_language_ruleset2
   ;;; set [pred_link2 <action_name> <pred_symbol>]
include: rea_do_action_ruleset
include: rea_memory_ruleset

enddefine;


define :ruleset rea_evaluate_internal_status_ruleset;
    [DLOCAL [prb_allrules = true] [prb_sortrules = false]];
    [LVARS [ag_name = sim_name(sim_myself)]];

RULE my_trace
  ==>
  [SAYIF gen_trace "evaluating internal state for agent ?ag_name "]

/*
RULE clear_old_model
==>
[POP11 clearproperty(old_model_db(sim_myself));]
*/

/* stopagent if agent is dead */

RULE dead
    [WHERE energy(sim_myself) <= g_out_of_energy]
        ==>
    [POP11 "dead" -> sim_status(sim_myself);]
    [status dead]
    [SAYIF body_state "agent is dead"]
    ;;; this lets the other agents (if any) run before exiting the scheduler
    [POP11 True -> sim_stopping_scheduler;]
    [POP11 True -> dead_status;]
    [STOPAGENT]

/*
Another possibility is:
[POP11 'message' =>
```

```
        sim_stop_scheduler ();]

This would exit the scheduler immediately (other agents will not run).
*/


/*
;;; monitor changes in internal sensors; this must be done before
;;; the evaluation of the internal state, otherwise they might not fire
;;; since the rules evaluating the internal state might contain a
;;; [STOP] action
*/

RULE constant_internal_values
    [actual_level ?sensor_name ?actual]
    [old_level ?sensor_name ?old]
    [WHERE actual = old]
        ==>
    [constant ?sensor_name]
    [SAYIF body_state "?sensor_name is constant"]

RULE increased_internal_values
    [actual_level ?sensor_name ?actual]
    [old_level ?sensor_name ?old]
    [WHERE actual > old]
        ==>
    [increased ?sensor_name]
    [SAYIF body_state "?sensor_name has increased"]

RULE decreased_internal_values
    [actual_level ?sensor_name ?actual]
    [old_level ?sensor_name ?old]
    [WHERE actual < old]
        ==>
    [decreased ?sensor_name]
    [SAYIF body_state "?sensor_name has decreased"]

/* ;;; evaluate the internal state */

RULE hurt
    [WHERE pain(sim_myself) > g_pain_threshold]
        ==>
    [POP11 "hurt" -> sim_status(sim_myself);]
    [status hurt]
    [SAYIF body_state "agent is hurt"]
    [STOP]

RULE hungry
    [WHERE energy(sim_myself) <= g_hunger_threshold]
        ==>
    [POP11 "hungry" -> sim_status(sim_myself);]
    [status hungry]
    [SAYIF body_state "agent is hungry"]
    [STOP]

RULE bored
```

```
    [WHERE no_motion(sim_myself) > g_rea_boredom_threshold]
        ==>
    [POP11 "bored" -> sim_status(sim_myself);]
    [status bored]
    [SAYIF body_state "agent is bored"]
    [STOP]

RULE ok
        ==>
    [POP11 "ok" -> sim_status(sim_myself);]
    [status ok]
    [SAYIF body_state "agent is ok"]

enddefine;


define :ruleset rea_evaluate_expectation_ruleset;
    [DLOCAL [prb_allrules = true] [prb_sortrules = false]];
/*
If in the previous cycle the agent was performing some action(s) it
has expectations on the results of its action(s). Evaluate these
expectations and store information on the object acted upon.
If the expectations are deceived the internal state changes to "bored"
so that the agent will move away.
*/

RULE negative_danger
    [expect no pain]
    [pred_link pain ?pred]
    [NOT status hurt]
        ==>
    [negative ?pred]
    [NOT expect no pain]
    [SAYIF know "thing is not ?pred (not dangerous)"]

RULE positive_danger
    [expect no pain]
    [pred_link pain ?pred]
    [status hurt]
        ==>
    [positive ?pred]
    [NOT expect no pain]
    [SAYIF know "thing is ?pred (dangerous)"]

RULE negative_instance
    [pred_link ?sensor_name ?predicate]
    [expect increase ?sensor_name]
    [OR [decreased ?sensor_name] [constant ?sensor_name]]
        ==>
    [negative ?predicate]
    [SAYIF know "thing is not ?predicate "]
    [NOT expect increase ?sensor_name]
    [NOT decreased ?sensor_name]
    [NOT constant ?sensor_name]
    [POP11 "bored" -> sim_status(sim_myself);]
    [NOT status ==]
```

```
    [status bored]
    [SAYIF body_state "internal state changed to bored"]
    [STOP]

RULE positive_instance
    [pred_link ?sensor_name ?predicate]
    [expect increase ?sensor_name]
    [increased ?sensor_name]
        ==>
    [positive ?predicate]
    [SAYIF know "thing is ?predicate "]
    [NOT expect increase ?sensor_name]
    [NOT increased ?sensor_name]
    [STOP]

enddefine;

define :ruleset rea_learning_data_ruleset;
    [DLOCAL [prb_allrules = true] [prb_sortrules = false]];

/*
If at target, actually store data (in the learning db) on the
object you are at, learn something, and then store you hypotheses
about how to recognise relevant objects in the model db.
*/

RULE forget_expectations
        ==>
    [NOT expect ==]

RULE positive_item
    [positive ?cat]
    [at_target = ?features]
        ==>
    [POP11 sim_add_data ([^cat positive ^features],
        learning_db(sim_myself));
        sim_add_data ([irrelevant negative ^features],
        learning_db(sim_myself));]
/*
    [POP11 sim_testadd_data ([^cat positive ^features],
        learning_db(sim_myself));
        sim_testadd_data ([irrelevant negative ^features],
        learning_db(sim_myself));]
*/
    [NOT positive ?cat]

RULE negative_item
    [negative ?cat]
    [at_target = ?features]
        ==>
    [POP11 sim_add_data ([^cat negative ^features],
        learning_db(sim_myself));]
/*
    [POP11 sim_testadd_data ([^cat negative ^features],
        learning_db(sim_myself));]
*/
```

```
    [NOT negative ?cat]

RULE build_model
        ==>
    [POP11 sim_add_list_to_db (mdl(exp_analyse(learning_db(sim_myself)),
        g_isc_parameter),
        model_db(sim_myself));

        ;;; I would like to put what follows into a [SAYIF ...] action
        if datalength(model_db(sim_myself)) = 0
        then sim_add_data ([model empty], sim_data(sim_myself));
        ;;; ['No model'] ==>
        else
        ;;; ['Data in learning db'] ==>
        ;;; prb_print_table (learning_db(sim_myself));
        ['Data in model db'] ==>
        prb_print_table (model_db(sim_myself));
        endif;]
enddefine;


define :ruleset rea_plan_action_ruleset;
    [DLOCAL [prb_allrules = true] [prb_sortrules = false]];

;;; the internal state triggers a plan to perform a certain action
;;; the default plan is collecting treasures

RULE avoid_being_hurt
    [status hurt]
        ==>
    [plan run_away]
    [SAYIF int_exp "plan is to run away"]

RULE avoid_being_hungry
    [status hungry]
        ==>
    [plan collect_energy]
    [SAYIF int_exp "plan is to collect energy"]
;;; if at target, forget being there
    [NOT at_target ==]

RULE avoid_being_bored
    [status bored]
        ==>
    [plan move_away]
    [SAYIF int_exp "plan is to move away"]

RULE default_plan
    [status ok]
        ==>
    [plan collect_treasure]
    [SAYIF int_exp "plan is to collect treasure"]
;;; if at target, forget being there
    [NOT at_target ==]

;;; here it should forget the internal state
```

```
RULE forget_internal_state
   ==>
    [NOT status ==]

enddefine;

define :ruleset rea_select_target_ruleset;
    [DLOCAL [prb_allrules = true] [prb_sortrules = false]];

RULE nothing_in_visual_field
    [OR [NOT thing ==] [at_target ==]]
        ==>
    [SAYIF reason [nothing in visual field]]
    [STOP]

RULE select_target
    [NOT target ==]
    [NOT at_target ==]
    [LVARS [ [dir dist] = rea_find_nearest(sim_myself)]]
    [thing ?dir ?dist ?type ?features]
        ==>
    [target ?dir ?dist ?type ?features]
    [SAYIF reason  target is ^features]

RULE arrived_at_target
    [target = ?dist_target ?type ?features]
    [WHERE dist_target <= 10 ] ;;; for minimal distance of that object
        ==>
    [NOT target ==]
    [at_target ?type ?features]
    [SAYIF reason [arrived at target]]

enddefine;


define: ruleset rea_choose_action_ruleset;
    [DLOCAL [prb_allrules = true] [prb_sortrules = false]];
    [LVARS new_heading new_speed];

;;; here, set the symbolic link between the expectation and the action
;;; that causes such expectation

RULE move_further_on
    [NOT thing ==]
        ==>
    [POP11 choose_new_heading1(sim_myself) -> new_heading;
        new_heading -> heading(sim_myself);]
    [to_be_moved_with ?new_heading 20]
    [SAYIF reason "nothing in visual field, moving forward"]

RULE head_towards_target
    [NOT at_target ==]
    [target ?dir_target ?dist_target ==]
;;; if there are obstacles, they should be taken into account
        ==>
    [POP11
```

```
        choose_new_speed (sim_myself, dist_target)-> new_speed;
        new_speed -> speed(sim_myself);
        dir_target -> heading(sim_myself);
        choose_new_heading1 (sim_myself) -> new_heading;
        new_heading -> heading(sim_myself);]
    [to_be_moved_with ?new_heading ?new_speed]
    [SAYIF reason "target in view hence approaching target"]

RULE run_from_target
    [at_target ==]
    [plan run_away]
        ==>
    [NOT at_target ==]
    [NOT to_stay]
    [POP11
        choose_new_speed(sim_myself, false) -> new_speed;
        new_speed -> speed(sim_myself);
        choose_new_heading1(sim_myself) -> new_heading;
        new_heading -> heading(sim_myself);]
    [to_be_moved_with ?new_heading ?new_speed]
    [SAYIF reason "running away from target"]

RULE move_from_target
    [at_target ==]
    [plan move_away]
        ==>
    [NOT at_target ==]
    [NOT to_stay]
    [POP11
        choose_new_speed(sim_myself, false) -> new_speed;
        new_speed -> speed(sim_myself);
        choose_new_heading1(sim_myself) -> new_heading;
        new_heading -> heading(sim_myself);]
    [to_be_moved_with ?new_heading ?new_speed]
    [SAYIF reason "moving away from target"]

RULE stay_and_collect_energy
    [at_target ==]
    [plan collect_energy]
        ==>
    [TESTADD to_stay]
    [action collect_energy]
    [expect increase energy]
    [expect no pain]
    [TESTADD action_link energy collect_energy]
    [SAYIF reason "collecting energy"]

RULE stay_and_collect_treasure
    [at_target ==]
    [plan collect_treasure]
        ==>
    [TESTADD to_stay]
    [action collect_treasure]
    [expect increase happiness]
    [expect no pain]
    [TESTADD action_link happiness collect_treasure]
```

```
    [SAYIF reason "collecting treasures"]

enddefine;


define :ruleset rea_internal_language_ruleset1;
    [DLOCAL [prb_allrules = true] [prb_sortrules = false]];

;;; set a predicate a symbol for each relevant category; relevant
;;; categories are determined by the agent's expectations

RULE generate_pred_symbols
    [LVARS pred_symbol]
    [expect = ?sensor_name]
    [NOT pred_link ?sensor_name ==]
        ==>
    [POP11 sim_name(sim_myself) >< gensym("_predicate") -> pred_symbol;]
    [TESTADD pred_link ?sensor_name ?pred_symbol]
    [SAYIF know "?pred_symbol refers to relevant objects
                 with respect to ?sensor_name"]

enddefine;

define :ruleset rea_internal_language_ruleset2;
    [DLOCAL [prb_allrules = true] [prb_sortrules = false]];

;;; link the predicate a symbol of each relevant category to
;;; the appropriate action; this is done using the link between expectation
;;; and predicate symbol and the link between expectation and action

RULE link_pred_symbols_with_actions
    [pred_link ?sensor_name ?pred_symbol]
    [action_link ?sensor_name ?action_name]
        ==>
    [TESTADD pred_link2 ?action_name ?pred_symbol]
    ;;; [SAYIF know "?pred_symbol refers to relevant object for action ?action_name"]

enddefine;


define :ruleset rea_do_action_ruleset;
    [DLOCAL [prb_allrules = true] [prb_sortrules = false]];

RULE do_not_move
    [LVARS [cycles = no_motion(sim_myself)]]
    [to_stay]
        ==>
    [NOT to_stay]
    [POP11 (cycles + 1) -> no_motion(sim_myself);
        0 -> speed(sim_myself);
;;;         'speed set to 0' =>
        update_energy(sim_myself, speed(sim_myself))]

RULE recharge
    [at_target ?type =]
    [action collect_energy]
```

```
        ==>
    [NOT action collect_energy]
    [POP11 recharge(sim_myself, type)]

RULE get_treasure
    [at_target ?type =]
    [action collect_treasure]
        ==>
    [NOT action collect_treasure]
    [POP11 get_treasure(sim_myself, type)]

RULE do_move
    [to_be_moved_with ?ag_heading ?ag_speed]
        ==>
    [NOT to_be_moved_with ==]
    [POP11
        lvars new_x, new_y;
        move_agent(sim_myself, ag_heading, ag_speed) -> (new_x, new_y);
        (new_x, new_y) -> sim_coords(sim_myself);
        ag_speed -> speed(sim_myself);
        ag_heading -> heading(sim_myself);
        ag_heading -> rc_axis(sim_myself);
        update_energy(sim_myself, ag_speed);
        0 -> no_motion(sim_myself);
        0 -> pain(sim_myself)]

enddefine;

define :ruleset rea_memory_ruleset;
    [DLOCAL [prb_allrules = true] [prb_sortrules = false]];

RULE remember_internal_values
    [actual_level ?sensor_name ?value]
        ==>
    [NOT old_level ?sensor_name ==]
    [old_level ?sensor_name ?value]
    [NOT actual_level ?sensor_name ==]

RULE forget_data
        ==>
    ;;; [NOT status ==]
    [NOT target ==]
    [NOT heading ==]
    [NOT speed ==]
    [NOT thing ==]
    ;;; [NOT to_stay]
    [NOT action ==]
    [NOT constant ==]
    [NOT increased ==]
    [NOT decreased ==]
    [NOT plan ==]
    [NOT model empty]
    [NOT avoid ==]
    [NOT obst ==]
    [NOT good_one ==]
```

```
    ;;; clear the model db, so that in the next cycle
    ;;; it is updated
    [POP11
        clearproperty(model_db(sim_myself));]

    ;;; The same effect is obtained by setting true
    ;;; sim_cycle_pause
    ;;; [PAUSE]

enddefine;


;;; -----------------------------------------------------------------------
;;; Section 5B - Adaptive agent architecture (adaptive1_rulesystem)

define :rulesystem adaptive1_rulesystem;
    cycle_limit = 1;
    debug = true;
    with_interval = 1;

include: evaluate_internal_state_ruleset
include: evaluate_expectation_ruleset
   ;;; use [pred_link ...] to decide whether positive case of pred_symbol
   ;;; or negative case of pred_symbol
include: learning_data_ruleset
include: plan_action
   ;;; set [plan <action_name>]
include: select_target_ruleset1
   ;;; use [pred_link2 <action_name> <pred_symbol>] to select appropriate
   ;;; target
include: select_target_ruleset2
include: set_target_ruleset
include: evaluate_position_ruleset

include: choose_action_ruleset
   ;;; set [action_link happiness collect_treasure]
   ;;; set [action_link energy collect_energy]
include: internal_language_ruleset1
   ;;; set [pred_link <sensor_name> <pred_symbol>]
include: internal_language_ruleset2
   ;;; set [pred_link2 <action_name> <pred_symbol>]
include: do_action_ruleset
;;; The previous two should be merged ...

include: memory_ruleset

enddefine;


define :ruleset evaluate_internal_state_ruleset;
    [DLOCAL [prb_allrules = true] [prb_sortrules = false]];
    [LVARS [ag_name = sim_name(sim_myself)]];

RULE my_trace
  ==>
  [SAYIF gen_trace "evaluating internal state for agent ?ag_name "]
```

```
RULE clear_old_model
==>
[POP11 clearproperty(old_model_db(sim_myself));]

/* stopagent if agent is dead */

RULE dead
    [WHERE energy(sim_myself) <= g_out_of_energy]
        ==>
    [POP11 "dead" -> sim_status(sim_myself);]
    [status dead]
    [SAYIF body_state "agent is dead"]
    ;;; this lets the other agents (if any) run before exiting the scheduler
    [POP11 True -> sim_stopping_scheduler;]
    [POP11 True -> dead_status;]
    [STOPAGENT]

/*
Another possibility is:
[POP11 'message' =>
        sim_stop_scheduler ();]

This would exit the scheduler immediately (other agents will not run).
*/


/*
;;; monitor changes in internal sensors; this must be done before
;;; the evaluation of the internal state, otherwise they might not fire
;;; since the rules evaluating the internal state might contain a
;;; [STOP] action
*/

RULE constant_internal_values
    [actual_level ?sensor_name ?actual]
    [old_level ?sensor_name ?old]
    [WHERE actual = old]
        ==>
    [constant ?sensor_name]
    [SAYIF body_state "?sensor_name is constant"]

RULE increased_internal_values
    [actual_level ?sensor_name ?actual]
    [old_level ?sensor_name ?old]
    [WHERE actual > old]
        ==>
    [increased ?sensor_name]
    [SAYIF body_state "?sensor_name has increased"]

RULE decreased_internal_values
    [actual_level ?sensor_name ?actual]
    [old_level ?sensor_name ?old]
    [WHERE actual < old]
        ==>
    [decreased ?sensor_name]
```

```
    [SAYIF body_state "?sensor_name has decreased"]

/* ;;; evaluate the internal state */

RULE hurt
    [WHERE pain(sim_myself) > g_pain_threshold]
        ==>
    [POP11 "hurt" -> sim_status(sim_myself);]
    [status hurt]
    [SAYIF body_state "agent is hurt"]
    [STOP]

RULE hungry
    [WHERE energy(sim_myself) <= g_hunger_threshold]
        ==>
    [POP11 "hungry" -> sim_status(sim_myself);]
    [status hungry]
    [SAYIF body_state "agent is hungry"]
    [STOP]

RULE bored
    [WHERE no_motion(sim_myself) > g_adapt1_boredom_threshold]
        ==>
    [POP11 "bored" -> sim_status(sim_myself);]
    [status bored]
    [SAYIF body_state "agent is bored"]
    [STOP]

RULE ok
        ==>
    [POP11 "ok" -> sim_status(sim_myself);]
    [status ok]
    [SAYIF body_state "agent is ok"]


enddefine;

define :ruleset evaluate_expectation_ruleset;
    [DLOCAL [prb_allrules = true] [prb_sortrules = false]];

/*
If in the previous cycle the agent was performing some action(s) it
has expectations on the results of its action(s). Evaluate these
expectations and store information on the object acted upon.
If the expectations are deceived the internal state changes to "bored"
so that the agent will move away.
*/

RULE negative_danger
    [expect no pain]
    [pred_link pain ?pred]
    [NOT status hurt]
        ==>
    [negative ?pred]
    [NOT expect no pain]
    [SAYIF know "thing is not ?pred (not dangerous)"]
```

```
RULE positive_danger
    [expect no pain]
    [pred_link pain ?pred]
    [status hurt]
        ==>
    [positive ?pred]
    [NOT expect no pain]
    [SAYIF know "thing is ?pred (dangerous)"]

RULE negative_instance
    [pred_link ?sensor_name ?predicate]
    [expect increase ?sensor_name]
    [OR [decreased ?sensor_name] [constant ?sensor_name]]
        ==>
    [negative ?predicate]
    [SAYIF know "thing is not ?predicate "]
    [NOT expect increase ?sensor_name]
    [NOT decreased ?sensor_name]
    [NOT constant ?sensor_name]
    [POP11 "bored" -> sim_status(sim_myself);]
    [NOT status ==]
    [status bored]
    [SAYIF body_state "internal state changed to bored"]
    [STOP]

RULE positive_instance
    [pred_link ?sensor_name ?predicate]
    [expect increase ?sensor_name]
    [increased ?sensor_name]
        ==>
    [positive ?predicate]
    [SAYIF know "thing is ?predicate "]
    [NOT expect increase ?sensor_name]
    [NOT increased ?sensor_name]
    [STOP]

enddefine;


define :ruleset learning_data_ruleset;
    [DLOCAL [prb_allrules = true] [prb_sortrules = false]];

/*
If at target, actually store data (in the learning db) on the
object you are at, learn something, and then store you hypotheses
about how to recognise relevant objects in the model db.
*/

RULE forget_expectations
        ==>
    [NOT expect ==]

RULE positive_item
    [positive ?cat]
    [at_target = ?features]
```

```
        ==>
    [POP11 sim_add_data ([^cat positive ^features],
        learning_db(sim_myself));
        sim_add_data ([irrelevant negative ^features],
        learning_db(sim_myself));]
    [NOT positive ?cat]

RULE negative_item
    [negative ?cat]
    [at_target = ?features]
        ==>
    [POP11 sim_add_data ([^cat negative ^features],
        learning_db(sim_myself));]
    [NOT negative ?cat]

RULE build_model
        ==>
    [POP11 sim_add_list_to_db (mdl(exp_analyse(learning_db(sim_myself)),
        g_isc_parameter),
        model_db(sim_myself));

        ;;; I would like to put what follows into a [SAYIF ...] action
        if datalength(model_db(sim_myself)) = 0
        then sim_add_data ([model empty], sim_data(sim_myself));
        ;;; ['No model'] ==>
        else
        ;;; ['Data in learning db'] ==>
        ;;; prb_print_table (learning_db(sim_myself));
        ['Data in model db'] ==>
        prb_print_table (model_db(sim_myself));
        endif;]

/*
;;; forgetting being at_target should depend on the plan
RULE forget_being_at_target
        ==>
    [NOT at_target ==]
    */

enddefine;


define :ruleset plan_action;
    [DLOCAL [prb_allrules = true] [prb_sortrules = false]];

;;; the internal state triggers a plan to perform a certain action
;;; the default plan is collecting treasures

RULE avoid_being_hurt
    [status hurt]
        ==>
    [plan run_away]
    [SAYIF int_exp "plan is to run away"]

RULE avoid_being_hungry
    [status hungry]
```

```
        ==>
    [plan collect_energy]
    [SAYIF int_exp "plan is to collect energy"]
;;; if at target, forget being there
    [NOT at_target ==]

RULE avoid_being_bored
    [status bored]
        ==>
    [plan move_away]
    [SAYIF int_exp "plan is to move away"]

RULE default_plan
    [status ok]
        ==>
    [plan collect_treasure]
    [SAYIF int_exp "plan is to collect treasure"]
;;; if at target, forget being there
    [NOT at_target ==]

;;; here it should forget the internal state
RULE forget_internal_state
   ==>
    [NOT status ==]

enddefine;

;;; NB the condition [NOT thing ==] will prevent interpreting
;;; (but for the first rule) the next 3 rulesets, concerning the
;;; selection of a target

define :ruleset select_target_ruleset1;
    [DLOCAL [prb_allrules = true] [prb_sortrules = false]];
    [LVARS [model_data = model_db(sim_myself)]];

;;; If nothing in visual field then cannot select target
;;; hence stop; if already at target then no need to select target

RULE no_thing
[OR [NOT thing ==] [at_target ==]]
==>
[STOP]

;;; Here the architecture discriminate between
;;; - no model at all
;;; - a model providing a concept of dangerous thing
;;; - a model, but no concept of dangerous thing
;;; In the next ruleset the cases <no model> and <a model>
;;; but no concept of dangerous thing and no concept of
;;; the relevant thing in question will be reduced to the same
;;; behaviour, that is: set the nearest thing in visual field
;;; as your target.

RULE no_model
    ;;; [NOT target ==]
    ;;; [NOT at_target ==]
```

```
    ;;; [thing ==]
    [model empty]
    [plan ?planned_action]
        ==>
    [select_target nearest]
    [SAYIF reason "model is empty hence make nearest thing my target"]
    [STOP]
    ;;; whatever was your plan

;;; Model is not empty

RULE with_model
    ;;; [NOT target ==]
    ;;; [NOT at_target ==]
    ;;; [NOT model empty]
    ;;; [thing ==]
    [plan ?planned_action]
    [pred_link2 ?planned_action ?pred_symbol]
        ==>
    [select_target ?pred_symbol]
    [SAYIF reason "make ?pred_symbol thing my target"]

RULE avoid_dangers
    ;;; [NOT target ==]
    ;;; [NOT at_target ==]
    ;;; [NOT model empty]
    [pred_link pain ?danger_symbol]
    [INDATA ?model_data [isa ?danger_symbol ?danger_features]]
    ;;; [thing ==]
        ==>
    [avoid ?danger_features]
    [SAYIF know "?danger_symbol are dangerous"]
    [SAYIF reason "avoid things with features ?danger_features"]

enddefine;


define :ruleset select_target_ruleset2;
    [DLOCAL [prb_allrules = true] [prb_sortrules = false]];
    [LVARS [model_data = model_db(sim_myself)]
        [agent_data = sim_data(sim_myself)]];

;;; If nothing in visual field then cannot select target
;;; hence stop

RULE no_thing
[OR [NOT thing ==] [at_target ==]]
==>
[STOP]

RULE set_nearest
    [select_target nearest]
    [NOT avoid ==]
    [LVARS [ [best] = find_best(sim_myself, [], [])]]
        ==>
    [LVARS item]
```

```
    [POP11 for item in best do prb_add_to_db(item, agent_data); endfor;]
    [STOP]

RULE set_recognised_item_with_danger_concept
    [select_target ?pred_symbol]
    [avoid ?danger_features]
    [INDATA ?model_data [isa ?pred_symbol ?constraints]]
    [LVARS [ [best] = find_best(sim_myself, constraints, danger_features)]]
        ==>
    [LVARS item]
    [POP11 for item in best do prb_add_to_db(item, agent_data); endfor;]
    [STOP]

RULE set_recognised_item_without_danger_concept
    [select_target ?pred_symbol]
    [NOT avoid ==]
    [INDATA ?model_data [isa ?pred_symbol ?constraints]]
    [LVARS [ [best] = find_best(sim_myself, constraints, [])]]
        ==>
    [LVARS item]
    [POP11 for item in best do prb_add_to_db(item, agent_data); endfor;]
    [STOP]

RULE just_avoid_dangers
    [select_target ==]
    [avoid ?danger_features]
    [LVARS [ [best] = find_best(sim_myself, [], danger_features)]]
        ==>
    [LVARS item]
    [POP11 for item in best do prb_add_to_db(item, agent_data); endfor;]
    [STOP]

RULE no_relevant_concept
    [select_target ==]
    [NOT avoid ==]
    [LVARS [ [best] = find_best(sim_myself, [], [])]]
        ==>
    [LVARS item]
    [POP11 for item in best do prb_add_to_db(item, agent_data); endfor;]
    [STOP]

RULE no_select_target
    [NOT select_target ==]
    [avoid ==]
    [LVARS [ [best] = find_best(sim_myself, [], [])]]
        ==>
    [LVARS item]
    [POP11 for item in best do prb_add_to_db(item, agent_data); endfor;]
    [STOP]

RULE check
        ==>
    ;;; it should not arrive here
    [POP11 ['ERROR Best has not been chosen!'] ==>]

enddefine;
```

```
define :ruleset set_target_ruleset;
    [DLOCAL [prb_allrules = true] [prb_sortrules = false]];
    [LVARS [model_data = model_db(sim_myself)]
        [agent_data = sim_data(sim_myself)]];

;;; If nothing in visual field then cannot select target
;;; hence stop

RULE no_thing
[OR [NOT thing ==] [at_target ==]]
==>
[STOP]

RULE forget_select_target
        ==>
    [NOT select_target ==]

RULE set_target
    [NOT target ==]
    ;;; [NOT at_target ==]
    [LVARS [ [nearest obstacles] = find_nearest(sim_myself)]]
    ;;; [good_one ==]
        ==>
    [LVARS item]
    [POP11 for item in nearest do prb_add_to_db(item, agent_data); endfor;
        for item in obstacles do prb_add_to_db(item, agent_data);endfor;]
enddefine;

/*
Now that the target is set, it should evaluate its own position
and check whether it is arrived at target.
*/

define: ruleset evaluate_position_ruleset;
    [DLOCAL [prb_allrules = true] [prb_sortrules = false]];

;;; If nothing in visual field then cannot evaluate position with
;;; respect to target, hence stop

RULE no_thing
[OR [NOT thing ==] [at_target ==]]
==>
[STOP]

RULE arrived_at_target
    [target = ?dist_target ?type ?features]
    [WHERE dist_target <= 10 ] ;;; for minimal distance of that object
        ==>
    [NOT target ==]
    [at_target ?type ?features]
    [SAYIF reason "arrived at target"]

enddefine;
```

```
define: ruleset choose_action_ruleset;
    [DLOCAL [prb_allrules = true] [prb_sortrules = false]];
    [LVARS new_heading new_speed];

;;; here, set the symbolic link between the expectation and the action
;;; that causes such expectation

RULE move_further_on
    [NOT thing ==]
        ==>
    ;;;    [POP11 'move on' => ]
    [POP11 choose_new_heading1(sim_myself) -> new_heading;
        new_heading -> heading(sim_myself);]
    [to_be_moved_with ?new_heading 20]
    [SAYIF reason "nothing in visual field, moving forward"]

RULE head_towards_target
    [NOT at_target ==]
    [target ?dir_target ?dist_target ==]
;;; if there are obstacles, they should be taken into account
        ==>
    [POP11
        ;;; 'head towards target' =>
        choose_new_speed (sim_myself, dist_target)-> new_speed;
        new_speed -> speed(sim_myself);
        dir_target -> heading(sim_myself);
        choose_new_heading1 (sim_myself) -> new_heading;
        new_heading -> heading(sim_myself);]
    [to_be_moved_with ?new_heading ?new_speed]
    [SAYIF reason "target in view hence approaching target"]

RULE run_from_target
    [at_target ==]
    ;;; [status hurt]
    [plan run_away]
        ==>
    [NOT at_target ==]
    ;;; [NOT status hurt]
    [NOT to_stay]
    [POP11
        choose_new_speed(sim_myself, false) -> new_speed;
        new_speed -> speed(sim_myself);
        choose_new_heading1(sim_myself) -> new_heading;
        new_heading -> heading(sim_myself);
        ; ]
    [to_be_moved_with ?new_heading ?new_speed]
    [SAYIF reason "running away from target"]

RULE move_from_target
    [at_target ==]
    ;;; [status bored]
    [plan move_away]
        ==>
    [NOT at_target ==]
    ;;; [NOT status bored]
    [NOT to_stay]
```

```
    [POP11
        choose_new_speed(sim_myself, false) -> new_speed;
        new_speed -> speed(sim_myself);
        choose_new_heading1(sim_myself) -> new_heading;
        new_heading -> heading(sim_myself);]
    [to_be_moved_with ?new_heading ?new_speed]
    [SAYIF reason "moving away from target"]

RULE stay_and_collect_energy
    [at_target ==]
    ;;; [OR [at_target ==] [to_stay]]
    ;;; [status hungry]
    [plan collect_energy]
        ==>
    ;;; [NOT status hungry]
    [TESTADD to_stay]
    [action collect_energy]
    [expect increase energy]
    [expect no pain]
    [TESTADD action_link energy collect_energy]
    [SAYIF reason "collecting energy"]

RULE stay_and_collect_treasure
    [at_target ==]
    ;;; [OR [at_target ==] [to_stay]]
    ;;; [status ok]
    [plan collect_treasure]
        ==>
    ;;; [NOT status ok]
    [TESTADD to_stay]
    [action collect_treasure]
    [expect increase happiness]
    [expect no pain]
    [TESTADD action_link happiness collect_treasure]
    [SAYIF reason "collecting treasures"]

enddefine;


define :ruleset internal_language_ruleset1;
    [DLOCAL [prb_allrules = true] [prb_sortrules = false]];

;;; set a predicate a symbol for each relevant category; relevant
;;; categories are determined by the agent's expectations

RULE generate_pred_symbols
    [LVARS pred_symbol]
    [expect = ?sensor_name]
    [NOT pred_link ?sensor_name ==]
        ==>
    [POP11 sim_name(sim_myself) >< gensym("_predicate") -> pred_symbol;]
    [TESTADD pred_link ?sensor_name ?pred_symbol]
    [SAYIF know "?pred_symbol refers to relevant objects
             with respect to ?sensor_name"]

enddefine;
```

```
define :ruleset internal_language_ruleset2;
    [DLOCAL [prb_allrules = true] [prb_sortrules = false]];

;;; link the predicate a symbol of each relevant category to
;;; the appropriate action; this is done using the link between expectation
;;; and predicate symbol and the link between expectation and action

RULE link_pred_symbols_with_actions
    [pred_link ?sensor_name ?pred_symbol]
    [action_link ?sensor_name ?action_name]
        ==>
    [TESTADD pred_link2 ?action_name ?pred_symbol]
    ;;; [SAYIF know "?pred_symbol refers to relevant object for action ?action_name"]

enddefine;


define :ruleset do_action_ruleset;
    [DLOCAL [prb_allrules = true] [prb_sortrules = false]];

RULE do_not_move
    [LVARS [cycles = no_motion(sim_myself)]]
    [to_stay]
        ==>
    [NOT to_stay]
    [POP11 (cycles + 1) -> no_motion(sim_myself);
        0 -> speed(sim_myself);
;;;        'speed set to 0' =>
        update_energy(sim_myself, speed(sim_myself))]

RULE recharge
    [at_target ?type =]
    [action collect_energy]
        ==>
    [NOT action collect_energy]
    [POP11 recharge(sim_myself, type)]

RULE get_treasure
    [at_target ?type =]
    [action collect_treasure]
        ==>
    [NOT action collect_treasure]
    [POP11 get_treasure(sim_myself, type)]

RULE do_move
    [to_be_moved_with ?ag_heading ?ag_speed]
        ==>
    [NOT to_be_moved_with ==]
    [POP11
        lvars new_x, new_y;
        move_agent(sim_myself, ag_heading, ag_speed) -> (new_x, new_y);
        (new_x, new_y) -> sim_coords(sim_myself);
        ag_speed -> speed(sim_myself);
        ag_heading -> heading(sim_myself);
        ag_heading -> rc_axis(sim_myself);
```

```
        update_energy(sim_myself, ag_speed);
        0 -> no_motion(sim_myself);
        0 -> pain(sim_myself)]

enddefine;

define :ruleset memory_ruleset;
    [DLOCAL [prb_allrules = true] [prb_sortrules = false]];

RULE remember_internal_values
    [actual_level ?sensor_name ?value]
        ==>
    [NOT old_level ?sensor_name ==]
    [old_level ?sensor_name ?value]
    [NOT actual_level ?sensor_name ==]

RULE forget_data
        ==>
    ;;; [NOT status ==]
    [NOT target ==]
    [NOT heading ==]
    [NOT speed ==]
    [NOT thing ==]
    ;;; [NOT to_stay]
    [NOT action ==]
    [NOT constant ==]
    [NOT increased ==]
    [NOT decreased ==]
    [NOT plan ==]
    [NOT model empty]
    [NOT avoid ==]
    [NOT obst ==]
    [NOT good_one ==]

    ;;; clear the model db, so that in the next cycle
    ;;; it is updated
    [POP11
        prb_add_db_to_db(model_db(sim_myself), old_model_db(sim_myself), true);
        clearproperty(model_db(sim_myself))]

    ;;; The same effect is obtained by setting true
    ;;; sim_cycle_pause
    ;;; [PAUSE]

enddefine;


define insert_names (input_file, info_list, output_file);
    ;;; input_file is a list L of lists
    ;;; info_list is something like
    ;;; [[150 'Irr'] [50 'En'] [50 'Dan'] [50 'Trea']]
    ;;; it means that the first 150 items of L will be called Irr
    ;;;               the next 50 items will be called En
    ;;; and so on and so forth
    lvars list_of_fea, item, number, name, result, counter;
    compile (input_file) -> list_of_fea;
```

```
    dlocal pop_pr_quotes;
    true -> pop_pr_quotes;
    [% for item in info_list do
            item(1) -> number;
            item(2) -> name;
            1 -> counter;
            until counter = number do
                [^name] <>[% hd(list_of_fea) %];
                tl(list_of_fea) -> list_of_fea;
                counter + 1 -> counter;
            enduntil;
        endfor; %] -> result;
    result ==>
    save (output_file, pr(% result %));
enddefine;

;;; In what follows we use the code in setup-world.p

new_reactive_agent (20, 20, 10) -> manu;
new_adaptive_agent (0, 0, 10) -> lola;

define create_objects ();
enddefine;

define ad_hoc_slot_values ();
enddefine;

define create_slot_values ();
enddefine;

define generate_locations1 ();
enddefine;

define setup_with_manu (r, d, input_file);
    ;;; input_file should be a file like ex_with_names.p
    ;;; [ [ <obj-name> [ <obj-features> ]] ... ]
    lvars list_with_names, num, list_of_coords, r_win, list_of_values, a;
    compile (input_file) -> list_with_names;
    length(list_with_names) -> num;

    ;;; generate_locations (num, r, d, prec) -> (r_win, list_of_coords);

    generate_locations1 (num, 2*r, d) -> (r_win, list_of_coords);

    create_slot_values (list_of_coords, r, list_with_names) -> list_of_values;
    ;;; [check for procedure create_slot_values] =>
    ;;; [the result is] =>
    ;;; ksslist_of_values ==>
    ;;; manu::(lola::(create_objects (new_demo_static, list_of_values))) ->
    ;;;        all_objs;
    manu::(create_objects (new_demo_static, list_of_values)) -> all_objs;


    rc_new_window_object("right", "top", 710, 710,
              true, 'demo_win', newsim_picagent_window) -> demo_win;
```

```
    ;;; rc_new_window_object("right", "top", ceiling(0.85*r_win),
    ;;;     ceiling(0.85*r_win), true, 'demo_win',
    ;;;     newsim_picagent_window) -> demo_win;

    ;;; draw the territory of the agent
    demo_win -> rc_current_window_object;
    rc_draw_circle (0, 0, g_rea_max_dist);

    /*
    lvars territory = new_rc_linepic();
    [WIDTH 2 COLOUR 'black'
        [CIRCLE {0 0 ^g_rea_max_dist}]] -> rc_pic_lines(territory);
    [^demo_win] -> rc_pic_containers(territory);
    rc_draw_linepic(territory);
    */

    for a in all_objs do

        [^demo_win] -> rc_pic_containers(a);
        rc_draw_linepic(a);
        ;;; make object mouse manipulable
        rc_add_pic_to_window(a, demo_win, true);
    endfor;
enddefine;

define demo_rea (input_file, num_cycles);
    1 -> my_agent;
    setup_with_manu (4, 25, input_file);
    reset_demo_counters ();
    sim_scheduler (all_objs, num_cycles);
    myprint_rea_end_demo ();
enddefine;

define setup_with_lola (r, d, input_file);
    ;;; input_file should be a file like ex_with_names.p
    ;;; [ [ <obj-name> [ <obj-features> ]] ... ]
    lvars list_with_names, num, list_of_coords, r_win, list_of_values, a;
    compile (input_file) -> list_with_names;
    length(list_with_names) -> num;

    ;;; generate_locations (num, r, d, prec) -> (r_win, list_of_coords);

    generate_locations1 (num, 2*r, d) -> (r_win, list_of_coords);

    create_slot_values (list_of_coords, r, list_with_names) -> list_of_values;
    ;;; [check for procedure create_slot_values] =>
    ;;; [the result is] =>
    ;;; ksslist_of_values ==>
    ;;; manu::(lola::(create_objects (new_demo_static, list_of_values))) ->
    ;;;        all_objs;
    lola::(create_objects (new_demo_static, list_of_values)) -> all_objs;


    rc_new_window_object("right", "top", 710, 710,
            true, 'demo_win', newsim_picagent_window) -> demo_win;
```

```
    ;;; rc_new_window_object("right", "top", ceiling(0.85*r_win),
    ;;;     ceiling(0.85*r_win), true, 'demo_win',
    ;;;     newsim_picagent_window) -> demo_win;

    ;;; draw the territory of the agent
    demo_win -> rc_current_window_object;
    rc_draw_circle (0, 0, g_rea_max_dist);

    /*
    lvars territory = new_rc_linepic();
    [WIDTH 2 COLOUR 'black'
        [CIRCLE {0 0 ^g_rea_max_dist}]] -> rc_pic_lines(territory);
    [^demo_win] -> rc_pic_containers(territory);
    rc_draw_linepic(territory);
    */

    for a in all_objs do

        [^demo_win] -> rc_pic_containers(a);
        rc_draw_linepic(a);
        ;;; make object mouse manipulable
        rc_add_pic_to_window(a, demo_win, true);
    endfor;
enddefine;

define demo_adap (input_file, num_cycles);
    2 -> my_agent;
    setup_with_lola (4, 25, input_file);
    reset_demo_counters ();
    sim_scheduler (all_objs, num_cycles);
    myprint_rea_end_demo ();
enddefine;

define create_slot_values (list_of_locs, r, fea_with_names) -> result;
    lvars item, name, fea_list;
    [% for item in list_of_locs do
            explode(hd(fea_with_names)) -> (name, fea_list);
            item <> [^r] <> [^fea_list] <> [^name];
            tl(fea_with_names) -> fea_with_names;
        endfor; %] -> result;
enddefine;

define create_objects (new_obj, list) -> all_objs;
    lvars item;
    [% for item in list do
            new_obj (explode(item));
        endfor; %] -> all_objs;
enddefine;

;;; a procedure to generate the locations of n square objects
;;; with side r
;;; objects are located randomly, they do not overlap, and have a density
;;; of d
;;; the input will be n (number of objs), r (side of the square)
;;; and d (density)
;;; the output will be the size of the window (a square of side r_win, centered
```

```
;;; on the origin of the coordinate system)
;;; and a list of n locations for the n objects (pairs of x and y coordinates)

;;; prec is the precision you want for the location; should be a multiple of 10
;;; if prec = 10 then coordinates have 1 decimal
;;; if prec = 100 then coordinates have 2 decimals, and so on

define ceiling () -> result;
enddefine;

define random_n_m (n, m) -> result_list;
    ;;; choose randomly n numbers in range 1 < numbers <= m
    ;;; return the list of the n numbers

    lvars counter = 0;
    [% until counter = n do random (m);
            counter +1 -> counter; enduntil; %] -> result_list;
enddefine;

define my_rem (divid, divis) -> num;
    if (divid rem divis) = 0 then divis -> num
    else (divid rem divis) -> num; endif;
enddefine;

define gen_coords (r, module, places_list) -> result;

    lvars sq1, y_gen, x_gens, item;

    ;;; first of all find the coordinates of the center of
    ;;; square number 1 (uppermost left)

    [% ((-0.5)*module+0.5)*r, (0.5*module-0.5)*r %] -> sq1;

    ;;; then find the x_generators and the y_generator

    sq1(2) -> y_gen;
    lvars counter = 1, keeper = sq1(1);
    [% sq1(1), until counter = module do
            keeper + r;
            keeper + r -> keeper;
            counter + 1 -> counter;
        enduntil; %] -> x_gens;

    ;;; then find the coordinates for each item in places_list

    [% for item in places_list do
            if item <= module then [% x_gens(item), y_gen %]
            else
                [% x_gens(my_rem (item, module)),
                    (y_gen - (item div module)*r) %];
            endif;
        endfor; %] -> result;

enddefine;

define generate_locations1 (n, r, d) -> (r_win, list_of_coords);
```

```
    ;;; this procedure will generate the locations for n objects
    ;;; (approximated to squares of size r) without the objects
    ;;; overlapping; objects will have a density 1/d

    lvars module, list_of_places;

    ceiling (sqrt(n*d)) -> module;
    ;;; [TRACE module is ^module]=>

    module*r -> r_win;
    random_n_m (n, (module*module)) -> list_of_places;
    ;;; [TRACE list of places is ^list_of_places] =>
    gen_coords (r, module, list_of_places) -> list_of_coords;
enddefine;

define ceiling n -> result;
    if round (n) < n then round (n) + 1 else round (n);
    endif; -> result;
enddefine;

;;; NOTES

;;; Procedure find_best has been modified, to prevent the agent
;;; approaching danger items when no good items are found

;;; It is the code of learning3.p modified so that there are two
;;; agents in the world, each acquiring a model of the environment.
;;; The reactive agent does not use the acquired model to change its
;;; behaviour, while the adaptive agent does so.
```

A.4 The learning algorithm

```
/*
The main procedures defined in this file are:

- exp_analyse
- mdl
- isc

After compiling the file, you can try the following tests.

TESTS for procedure isc:

The variable to control the tracing of isc is my_trace; it's default
value is false; make it true to have some tracing messages.

vars my_trace = true;

vars test = [[positive [thin light two_t one_n]]
             [positive [thin light one_t two_n]]
             [negative [thin light one_t one_n]]
             [positive [thick dark two_t two_n]]
             [positive [thin dark one_t two_n]]
```

```
             [positive [thin light one_t two_n]]
            ];

vars second_test = [[positive [thin light two_t one_n]]
                    [positive [thin light one_t two_n]]
                    [negative [thin light one_t one_n]]
                    [positive [thick dark two_t two_n]]
                    [positive [thin light one_t two_n]]
                    [positive [thin dark one_t two_n]]
                   ];

test ==>
second_test ==>

Notice the different behaviour on different parameter k:

isc (test, 4) =>
isc (test, 6) =>

isc (second_test, 4) =>
isc (second_test, 6) =>
*/

;;; -------------------- EXP_ANALYSE ------------------------------
/*
The procedure EXP_ANALYSE analyses the learning database of the agent
(that can be viewed as the agent's internal representation of experiences),
and makes them suitable for the procedure MDL.

Inputs: ESET a set of experiences, represented as a prb_database

e.g.
[[treasure negative [large bitter shaval1 smeval2 colval4]]
 [treasure positive [large bitter shaval1 smeval3 colval3]]
]
[[danger negative [small sour shaval2 smeval1 colval1]]
 [danger positive [small sweet shaval1 smeval2 colval5]]
]

Outputs:
PREDS a list of predicate symbols;
CSET a set of class names associated with positive and
negative instances of the class.

e.g.
[[treasure [[negative ...] [positive ...] [positive ...]]]
 [danger [positive ...] [negative ...] [negative ...]]]
]

NB in learning3.p class names have the following structure:
   <agent-name>_predicate<number>
   for example manu_predicate34
   meaning that predicate34 is a predicate symbol of the internal
   language of the agent manu
*/
```

```
;;; make poprulebase available
uses newkit;
uses poprulebase;

define exp_analyse (eset) -> (preds, cset);
    lvars partial, item1;
    if eset == [] then [] -> preds; [] -> cset;
    else
        [% appproperty(eset,
                procedure(item,val);
                    lvars item;
                    item;
                endprocedure
            ); %] -> partial;
        partial -> preds;
        ;;;dlocal popmatchvars = [];   ;;; reset popmatchvars on exit
        [% for item1 in partial do
                [^item1 [% prb_match_apply (eset, [^item1 ==], tl); %]];
            endfor;
        %] -> cset;
    endif;
enddefine;


;;; -------------------- MDL ------------------------------
/*
Procedure MDL to generate a multi-class decision list.
Input: CSET a set of class names
       ISET a set of classified instances
       K is the parameter to be given to isc.p
Output: RULE-SET a decision list composed of single-region
        descriptions

Procedure MDL (CSET, ISET, K)
Let RULE-SET be the empty set.
For each CLASS in CSET,
    let IISET be positive and negative instances in ISET labeled
    with CLASS;
    let DNF be ISC (IISET, K);
    for each term D in DNF,
        add the rule "If D then Class" to RULE-SET.
Return RULE-SET.

NB The possible conflicts among rules in RULE-SET will be solved
   at the level of the agent architecture. In learning3.p the conflict
   resolution strategy is just attempted, and is not working properly yet.
*/

;;; See below
define isc ();
enddefine;

define mdl (preds, cset, k) -> rule_set;
    lvars pred_symbol, item, dnf_exp, disjunct;
    [% for pred_symbol in preds do
            for item in cset do
```

```
                if hd(item) = pred_symbol then
                    if my_trace then
                        [TRACE the arguments to isc are] ==>
                        item(2) ==>
                        k ==>
                    endif;
                    isc(item(2), k) -> dnf_exp;
                    if my_trace then
                        [TRACE the output from isc is] ==>
                        dnf_exp ==>
                    endif;
                    if dnf_exp == [] then [isa ^pred_symbol []] else
                        for disjunct in dnf_exp do
                            [isa ^pred_symbol ^disjunct];
                        endfor;
                    endif;
                endif;
            endfor;
        endfor; %] -> rule_set;
enddefine;


;;; -------------------- ISC ------------------------------
/*
The procedure ISC implements an algorithm for the incremental induction of
disjunctive concepts using separate and conquer.

ISC retains a single hypothesis in memory, which is a disjunctive
set of logical or threshold terms, and is initialised to a single term
at the outset. ISC retains up to the most recent k training instances
for use in evaluating alternative hypotheses. ISC revises its hypothesis
only when it makes an error in classification.

ISC takes as input a list of classified instances of one category and
produces as output a disjunction of single-region descriptions to be used
as a recognition test for the category. The input is represented as a list
of list with the following form:
[<positive/negative> [<list of features]]

For example, something like:
[[negative [large bitter shaval1 smeval2 colval4]]
 [negative [large bitter shaval1 smeval3 colval3]]
 [negative [small sour shaval2 smeval1 colval1]]
 [negative [medium bitter shaval2 smeval2 colval3]]
 [positive [small sweet shaval1 smeval2 colval5]]
]


Procedure ISC to learn incrementally a DNF ecpression using
separate and conquer.
Input: IISET a set of classified training instances.
Output: A disjunction of single-region descriptions.
Parameters: K is the number of instances retained in memory.

Procedure ISC (IISET)
Let H be a set containing a maximally specific single-region
```

```
    description based on the first positive instance in IISET.
For each misclassified instance I in IISET,
    Let KSET be the previous K (or fewer) instances.
    If I is positive,
    then for each term C in H,
             let S be the minimal revision of C that matches I.
             Generate a revision of H in which S replaces C.
         Let BEST be the best scoring of these revisions.
         Let NEW be ADD_TERM (H, I)
    Else if I is negative,
         then let BEST be a copy of H.
         For each term C in BEST that matches I,
             Let G be a minimal revision of C that does not match I.
             Replace C with G in BEST.
         Let NEW be REMOVE_TERM (H, C)
    If SCORE (NEW, KSET) > SCORE (BEST, KSET)
    then let BEST be NEW.
    If SCORE (BEST, KSET) > SCORE (H, KSET)
    then let H be BEST.
Return H.

*/

/*
vars test = [[positive [thin light two_t one_n]]
             [positive [thin light one_t two_n]]
             [negative [thin light one_t one_n]]
             [positive [thick dark two_t two_n]]
             [positive [thin dark one_t two_n]]
             [positive [thin light one_t two_n]]
            ];



vars second_test = [[positive [thin light two_t one_n]]
                    [positive [thin light one_t two_n]]
                    [negative [thin light one_t one_n]]
                    [positive [thick dark two_t two_n]]
                    [positive [thin light one_t two_n]]
                    [positive [thin dark one_t two_n]]
                   ];
*/

;;; my_take makes a list out of the first n elements
;;; of the input list

define my_take1 (n, list) -> list1;
    lvars counter, item, partial;
    if length(list)<= n then list -> list1;
    else
        0 -> counter;
        [] -> partial;
        until counter = n do
            partial <> [%hd(list)%] -> partial;
            tl(list) -> list;
            counter + 1 -> counter;
```

```
        enduntil;
        partial -> list1;
    endif;
enddefine;

;;; another one ...

define my_take (n, list) -> list1;
    lvars counter, item;
    if n = 0 then [] -> list1 else
        [% 0 -> counter;
            for item in list do
                item;
                counter + 1 -> counter;
                if counter = n then quitloop; endif;
            endfor; %] -> list1;
    endif;
enddefine;

/*
my_take (0, [a b c d]) =>
my_take (2, [a b c d]) =>
my_take (7, [a b c d]) =>
*/

define subset (list1, list2) -> boolean;
    ;;; check whether list1 is a subset of list2
    lvars partial, item;
    if length(list1)>length(list2) then false -> boolean;
    elseif list1 == [] or list2 == [] then false -> boolean;
    else
        true -> boolean;
        maplist (list1, member(%list2%)) -> partial;
        for item in partial do
            if item = false then false -> boolean; quitloop; endif;
        endfor;
    endif;
enddefine;


define sublist1 (list1, list2) -> boolean;
    ;;; check whether list1 is a sublist of list2 with no gaps
    ;;; e.g. [b c] is a sublist1 of [a b c d]
    ;;; but [b d] is not
    lvars num1, num2;
    length(list1) -> num1;
    length(list2) -> num2;
    if num1>num2 or list1 == [] or list2 == [] then false -> boolean;
    elseif my_take (num1, list2) = list1 then true -> boolean;
    else sublist1 (list1, tl(list2)) -> boolean;
    endif;
enddefine;


define sublist (list1, list2) -> boolean;
    if list1 == [] then true -> boolean;
```

```
    elseif list2 == [] then false -> boolean;
    elseif hd(list1) = hd(list2) then sublist(tl(list1), tl(list2)) -> boolean;
    else sublist(list1, tl(list2)) -> boolean;
    endif;
enddefine;

/*
sublist ([], [1 2 3]) =>
sublist ([1 2 3], []) =>
sublist ([1 2 3], [1 2]) =>
sublist ([1 2 3], [1 2 3]) =>
sublist ([a b], [a b c d]) =>
sublist ([a c d], [a b c d]) =>
sublist ([c d], [a b c d]) =>
sublist ([a f g], [a b c d]) =>
sublist ([a [b c]], [[b c] a f g]) =>
*/

define classified (inst, hyp) -> boolean;
    ;;; check whether instance is classified by hyp
    ;;; hyp is a list of lists, representing a disjunction of
    ;;; conjunction of object features
    lvars conjunct;
    false -> boolean;
    for conjunct in hyp do
        if sublist(conjunct, inst(2)) then true -> boolean;
        endif;
    endfor;
enddefine;

/*
classified ([positive [dark thin]], [[dark][light thin]])=>
classified ([negative [dark thin one_t]], [[]])=>
classified ([negative [dark thick]], [[light][dark one_t]])=>
*/

define isc_general (conjunct, inst_fea) -> result;
    ;;; generalise conjunct just enough to cover inst_fea
    lvars item;
    [% for item in conjunct do
            if member(item, inst_fea) then item;
            endif;
        endfor; %] -> result
enddefine;

/*
isc_general ([dark thin], [light thin red cheap]) =>
isc_general ([dark thin], [light thick green]) =>
isc_general ([dark thin], [dark thin green]) =>
isc_general ([dark thin red], [light thin blue cheap]) =>
*/

;;; Now the pocedure to generate the minimal revision in case
;;; of misclassified positive instance

define help_prova (n, hyp, inst) -> result;
```

```
lvars partial = copylist(hyp);
    isc_general (hyp(n), inst(2)) -> partial(n);
    partial -> result;
enddefine;

define prova (hyp, inst) -> result;
    lvars counter;
    [% 1 -> counter;
        until counter = length(hyp)+1 do
            help_prova (counter, hyp, inst);
            counter+1 -> counter;
        enduntil;
    %] -> result;
enddefine;

;;; help_prova and prova can be combined so

define help_isc_class1 (hyp, inst) -> result;
    lvars counter, hyp1;
    [% 1 -> counter;
        until counter = length(hyp)+1 do
            ;;; NB copylist MUST be used to prevent hyp being changed
            ;;; together with hyp1!!!
            copylist(hyp) -> hyp1;
            isc_general (hyp(counter), inst(2)) -> hyp1(counter);
            hyp1;
            counter + 1 -> counter;
        enduntil;
    %] -> result;
enddefine;

/*
help_isc_class1 ([[dark][light thin][red]], [positive [dark thin blue]]) ==>
*/


define isc_add_term (hyp, inst) -> new_hyp;
    inst(2)::hyp -> new_hyp;
enddefine;

/*
isc_add_term ([[dark][light thin][red]], [positive [dark thin blue]]) =>
*/


define get_fea_values (inst, vset) -> new_vset;
    ;;; extract feature values from inst
    ;;; the possible values of each object feature are needed to be able
    ;;; to specialise an overly general hypothesis

    ;;; I assume here that sensors are run ALWAYS in the same order
    ;;; and if a value is not present, a 0 (zero) is put in, to mark
    ;;; the place. I am relying on the fact that, in a list of feature
    ;;; values, the position of each item represents the sensorial quality
    ;;; of the value: for example, the first item is always a reading from
    ;;; sensor1 (assume, sight), the second item is always a reading from
```

```
    ;;; sensor2 (assume, touch), etc. etc.

    lvars n, value, item, num, val_list;
    if vset == [] then
        1 -> n;
        [% for value in inst(2) do
                [^n ^value];
                n+1 -> n;
            endfor;
        %] -> new_vset;
    else
        [% for item in vset do
                hd(item) -> num;
                tl(item) -> val_list;
                if member ((inst(2))(num), val_list) then item
                else num::((inst(2))(num)::val_list);
                endif;
            endfor;
        %] -> new_vset;
    endif;
enddefine;

/*
get_fea_values ([positive [red cold sweet hard]], []) =>
** [[1 red] [2 cold] [3 sweet] [4 hard]]

get_fea_values ([positive [green cold sweet soft]],
                [[1 red] [2 cold] [3 sweet] [4 hard]]) =>
** [[1 green red] [2 cold] [3 sweet] [4 soft hard]]
*/

define take_best1 ();
enddefine;

define present (num, vset) -> bool;
    lvars item;
    false -> bool;
    for item in vset do
        if num = hd(item) then true -> bool;
            quitloop;
        endif;
    endfor;
enddefine;

define my_order (item, list) -> result;
    if list==[] then [^^list ^item] -> result;
    elseif hd(item)<(hd(list))(1) then item::list -> result;
    else hd(list)::(my_order(item, tl(list))) -> result;
    endif;
enddefine;

/*
my_order ([3 sweet bitter], [[1 red]]) =>
my_order ([3 sweet bitter], [[2 dark] [4 hot]]) =>
my_order ([3 sweet bitter], [[5 dark]]) =>
*/
```

```
define remove_num (list) -> result;
    lvars item;
    [% for item in list do item(2);
        endfor; %] -> result;
enddefine;

/*
remove_num (my_order ([3 sweet], [[2 dark] [4 hot]])) =>
*/

define my_combine (list1, analysed_conj, neg_fea) -> result;
    lvars item;
    [% for item in list1 do
            if not(member(item(2), neg_fea)) then
                remove_num (my_order (item, analysed_conj));
            endif;
        endfor; %] -> result;
enddefine;

define isc_special (conj, inst, vset, kset) -> result;
    ;;; specialises conj (using vset -- the possible feature values)
    ;;; so that it does not match inst anymore
    ;;; All the possible specialisaitons are found, then the most accurate
    ;;; one is chosen and returned. The choice is made using the fucntion
    ;;; accuracy (positive covered + negative not covered divided by total
    ;;; number of examined instances)

    lvars neg_fea = inst(2), item, item1, analysed_conj, analysed_vset,
        list1, item2, partial;

    [% for item in conj do
            for item1 in vset do
                if member (item, tl(item1)) then hd(item1)::[^item];
                endif;
            endfor;
        endfor; %] -> analysed_conj;

    [% for item in vset do
            if not(present(hd(item), analysed_conj)) then item;
            endif;
        endfor; %] -> analysed_vset;

    [% for item in analysed_vset do
            [% for item1 in tl(item) do hd(item)::[^item1]; endfor%] -> list1;
            explode(my_combine (list1, analysed_conj, neg_fea));
        endfor; %] -> partial;

    if null(partial)
        ;;; this means there are no further ways of specialising conj
        then conj -> result;
    else
        take_best1 (partial, kset) -> result;
    endif;

enddefine;
```

```
/*
isc_special ([red sweet], [negative [red cold sweet smooth]],
             [[1 red green blue] [2 hot cold] [3 sweet bitter] [4 soft hard]],
             []) ==>

all the possible specialisations are [[red hot sweet] [red cold sweet]
                                      [red sweet soft] [red sweet hard]]

those that do not match inst are [[red hot sweet] [red sweet soft]]

the best one depends on kset
*/

define help_isc_class2 (hyp, inst, vset, kset) -> result;
    ;;; replace each term in hyp that matches inst with a minimal
    ;;; revision (a more specialised term) that do not match inst
    ;;; anymore
    lvars item;
    [% for item in hyp do
            if sublist (item, inst(2)) then isc_special (item, inst, vset, kset);
            else item;
            endif;
        endfor;
    %] -> result;
enddefine;

/*
trace take_best1;
help_isc_class2 ([[thin light]],
                 [negative [thin light one_t one_n]],
                 [[1 thin] [2 light] [3 one_t two_t] [4 two_n one_n]],
                 [[negative [thin light one_t one_n]]
                  [positive [thin light one_t two_n]]
                  [positive [thin light two_t one_n]]]) =>
** [[thin light two_t]]
*/


define isc_remove_term (hyp, inst) -> new_hyp;
    lvars disjunct, partial;
    [% for disjunct in hyp do
            if not(sublist(disjunct, inst(2))) then disjunct; endif;
        endfor; %] -> partial;
    if partial == [] then [[]] -> new_hyp else
        partial -> new_hyp; endif;
enddefine;

/*
isc_remove_term ([[red][green light][soft][]], [negative [red light hard]]) =>

isc_remove_term ([[thin light]],
                 [negative [thin light one_t one_n]]) =>
** []

*/
```

```
define accuracy (hyp, k, kset) -> result;
    lvars n, item;
    0 -> n;
    for item in kset do
        if ((item(1)="positive") and (classified (item, hyp)))
        or ((item(1)="negative") and not(classified (item, hyp))) then
            n+1 -> n;
        endif;
    endfor;
    n/k -> result;
enddefine;

define isc_score(hyp, k, kset) -> result;
    ;;; k is the length of kset
    lvars l = length(hyp);
    if l = 0 then 0 -> result;
        [ERROR hypothesis was the empty list] =>
    elseif l = 1 and hd(hyp) == [] then
        1/k + accuracy (hyp, k, kset) -> result;
    else 1/l + accuracy (hyp, k, kset) -> result;
    endif;
enddefine;


/*
define isc_score(hyp, k, kset) -> result;
    ;;; k is the length of kset
    lvars l = length(hyp);
    1/l + accuracy (hyp, k, kset) -> result;
enddefine;
*/

/*
isc_score ([[Pippo][Pluto]], 9, test) =>
*/


define take_best1 (revisions, kset) -> best;
    lvars revision, partial, new, k = length(kset);
    0 -> partial;
    for revision in revisions do
        accuracy ((revision::[]), k, kset) -> new;
        if new > partial then
            new -> partial;
            revision -> best;
        endif;
    endfor;
enddefine;


define take_best (revisions, kset) -> best;
    lvars revision, partial, new, k = length(kset);
    0 -> partial;
    for revision in revisions do
        isc_score (revision, k, kset) -> new;
```

```
        if new > partial then
            new -> partial;
            revision -> best;
        endif;
    endfor;
enddefine;

/*
take_best (help_isc_class1 ([[dark][light thin][red]],
[positive [dark thin blue]]), test)  =>
*/

define isc_classify (inst, k, kset, hyp, vset)
        -> (new_kset, new_hyp, new_vset);
    ;;; NB the score function is isc_score
    lvars k1, partial1, best, new;

    ;;; [TRACE A kset is ^kset] =>
    my_take (k, inst::kset) -> kset;
    ;;; [TRACE B kset is ^kset] =>
    length(kset) -> k1;

    ;;; [TRACE A vset is ^vset] =>
    get_fea_values (inst, vset) -> vset;
    ;;; [TRACE B vset is ^vset] =>

    ;;; Case1: the instance is correctly classified
    if ((inst(1)="positive") and (classified (inst, hyp)))
    or ((inst(1)="negative") and not(classified (inst, hyp))) then
        if my_trace then
            [TRACE Case1: correctly classified instance] =>
        endif;
        ;;; my_take (k, inst::kset) -> new_kset;
        hyp -> new_hyp;
    else
        ;;; Case2: the misclassified instance is positive
        if inst(1)="positive" then
            ;;; for each conjunct in hyp generate a revision of hyp
            ;;; (more general) that cover inst
            if my_trace then
                [TRACE Case2: misclassified positive instance] =>
            endif;
            help_isc_class1 (hyp, inst) -> partial1;

            ;;; take the best of these revision, according to
            ;;; score function isc_score
            take_best (partial1, kset) -> best;
            if my_trace then
                [TRACE best is ^best] =>
            endif;
            ;;; generate a new hypothesis by simply adding a new disjunct
            ;;; based on the misclassified instance
            isc_add_term (hyp, inst) -> new;
            if my_trace then
                [TRACE new is ^new] =>
            endif;
```

```
        else
            ;;; Case3: the misclassified instance is negative
            if my_trace then
                [TRACE Case3: misclassified negative instance] =>
            endif;
            help_isc_class2 (hyp, inst, vset, kset) -> best;
            if my_trace then
                [TRACE best is ^best] =>
            endif;
            isc_remove_term (hyp, inst) -> new;
            if my_trace then
                [TRACE new is ^new] =>
            endif;
        endif;
        if isc_score (new, k1, kset) > isc_score (best, k1, kset) then
            new -> best;
        endif;
        if isc_score (best, k1, kset) >= isc_score (hyp, k1, kset) then
            ;;; CHECK!  hyp -> new_hyp else best -> new_hyp;
            best -> new_hyp else hyp -> new_hyp;
        endif;
        ;;; my_take (k, inst::kset) -> new_kset;
    endif;

    if my_trace then
        [TRACE examined instance ^inst with
            training set ^kset
            working hypothesis ^hyp
            known feature values ^vset] ==>
    endif;

    kset -> new_kset;
    ;;; [TRACE C new_kset is ^new_kset] =>

    vset -> new_vset;
    ;;; [TRACE C new_vset is ^new_vset] =>
    if my_trace then
        [TRACE new_hyp is ^new_hyp] =>
    endif;
enddefine;


define my_test (hyp, best, new, k1, kset) -> result;
lvars new_hyp;
    if isc_score (new, k1, kset) > isc_score (best, k1, kset) then
        new -> best;
    endif;
    if isc_score (best, k1, kset) > isc_score (hyp, k1, kset) then
        hyp -> best;
    endif;
    best -> result;
enddefine;

/*
trace isc_score;
```

```
my_test ([[thin light]],
         [[thin light two_t]],
         [],
         3,
         [[negative [thin light one_t one_n]]
          [positive [thin light one_t two_n]]
          [positive [thin light two_t one_n]]]) =>

accuracy ([[]],
           3,
           [[negative [thin light one_t one_n]]
            [positive [thin light one_t two_n]]
            [positive [thin light two_t one_n]]]) =>

*/


define isc_classify2 (iset, k, kset, hyp, vset) -> result;
    lvars nk, nh, nv;
    isc_classify(hd(iset), k, kset, hyp, vset) -> (nk, nh, nv);
    if tl(iset) == [] then nh -> result;
    else
        isc_classify2(tl(iset), k, nk, nh, nv) -> result;
    endif;
enddefine;


define isc (iset, k) -> hyp;
    lvars item, in_hyp, features, kset, val_set;
    ;;; initialise hyp to be:
    ;;; the empty list if iset is empty, or contains only negative instances
    ;;; a maximally specific single-region
    ;;; description based on the first positive instance in iset
    if iset == [] then [] -> hyp else
        for item in iset do
            if item matches ![positive ?features]
            then [^features] -> in_hyp; quitloop;
            else [] -> in_hyp;
            endif;
        endfor;

        if my_trace then
            [TRACE initial hypothesis is ^in_hyp] =>
        endif;

        if in_hyp == [] then in_hyp -> hyp;
            if my_trace then
                [TRACE return because initial hypothesis is empty] =>
            endif;
            return;
        else
            [] -> kset;
            ;;; get_fea_values(hd(iset), []) -> val_set;
            [] -> val_set;

            isc_classify2 (iset, k, kset, in_hyp, val_set) -> hyp;
```

```
        endif;
    endif;
enddefine;
```

A.5 Setting the environment

```
/*
This file contains the main procedures for the setup of the environment.

Commands used to generate test_env.p (the environment used for the demo
of the system):

setup_with_types ([relevant [colour Re Bl Ye Gr Wh]
                            [size S M L]
                            [taste salty sweet sour bitter]
                   irrelevant],
                  "test_env_aux1");

do_instances_with_info('test_env_aux1.p',240,"test_env_aux2");

insert_names_with_info('test_env_aux2.p',"test_env");

*/

vars procedure show_possible, choose_cat_with_types;

define setup_with_types (attributes,file);
    lvars partial1, causal_classes = [Treasure Danger Food];
    [% show_possible(attributes) %] -> partial1;
    save (file, pr(% [% choose_cat_with_types(partial1,causal_classes) %] %));
enddefine;

;;; the next procedure is used to establish how many objects should be
;;; in the world, and in which proportions
;;; input_file_name should be the output of setup_with_types
;;; num is the number of objects in the world
;;; the procedure will ask the proportions of the causal classes
;;; output_file_name is the file to be used as inpuit for the next
;;; setup procedure, namely insert_names_with_info


vars procedure make_randomly, make_proportions;
;;; trace make_randomly, make_proportions;

define sum_list (aList) -> number;
    if aList == [] then 0 -> number;
    elseif length(aList) = 1 then hd(aList) -> number;
    else hd(aList)+(sum_list(tl(aList))) -> number;
    endif;
enddefine;

/*
sum_list([])=>
```

```
sum_list([3])=>
sum_list([1 2 3])=>
sum_list([0.3 0.3 0.3])=>
*/

define mk_pairs (list1,list2) -> result;
    lvars n = 1;
    if list1 == [] or list2 == [] then []
    else
        [% until n = length(list1)+1 do
                [% list1(n), list2(n) %];
                n+1 -> n;
            enduntil; %]
    endif -> result;
enddefine;

/*
mk_pairs ([],[])=>
mk_pairs ([],[a b c]) =>
mk_pairs ([a b c],[]) =>
mk_pairs ([a b c],[1 2 3 4])=>
*/

define do_instances_with_info (input_file_name, num, output_file_name);
    lvars list1, types, cat_types, list_of_cat, cat_names, props,
        item, partial, partial2, list_of_inst, info_list;
    compile (input_file_name) -> list1;
    list1(1) -> types;
    ;;; something like
    ;;; [[[type1 [small sweet]] [type2 [small sour]] [type3 [small bitter]] ... ]
    ;;; rev(list1(2)) -> cat_types;
    list1(2) -> cat_types;
    ;;; something like
    ;;; [[OTHER [type3 [small bitter]] [type6 [medium bitter]] ... ]
    ;;;  [EN [type4 [medium sweet]]]
    ;;;  [DAN [type1 [small sweet]]]
    ;;;  [TREA [type2 [small sour]] [type5 [medium sour]] [type8 [large sour]]] ]
    list1(3) -> list_of_cat;
    ;;; something like
    ;;; [ [OTHER [[obj1 [small bitter shaval1 smeval1 colval1]]
    ;;;           [obj2 [small bitter shaval1 smeval1 colval2]]
    ;;;           ...
    ;;;          ]
    ;;;          [[obj1 [medium bitter shaval1 smeval1 colval1]]
    ;;;           [obj2 [medium bitter shaval1 smeval1 colval2]]
    ;;;           ...
    ;;;          ]
    ;;;      ...
    ;;;   ]
    ;;;   [EN [[obj1 [medium sweet shaval1 smeval1 colval1]]
    ;;;        [obj2 [medium sweet shaval1 smeval1 colval2]]
    ;;;    ...
    ;;;   ]
    ;;;  ...
    ;;; ]
    ;;; [% for item in allbutlast(1,cat_types) do item(1); endfor %] -> cat_names;
```

```
    ;;; [% for item in allbutfirst(1,cat_types) do item(1); endfor %] -> cat_names;
    [% for item in cat_types do item(1); endfor %] -> cat_names;

    [type in proportions for %allbutfirst(1,cat_names)% (total is ^num)] =>
    readline() -> props;
    lvars n = sum_list(props);
    if n > 1 then
        [the sum of proportions should not exceed 1]=>
        [type in proportions]=>
        readline() -> props;
        (1 - sum_list(props))::props -> props;
    else
        (1 - n)::props -> props;
    endif;
    mk_pairs(cat_names,props) -> partial;

    [% for item in partial do
            [% item(1), round(num*item(2)), item(2) %];
        endfor; %] -> partial2;

    make_randomly (list_of_cat, partial2) -> list_of_inst;

    ;;; remember, partial2 is something like
    ;;; [[OTHER 150 1_/2] [Treasure 50 1_/6] [Danger 50 1_/6] [Food 50 1_/6]]

    /*
    [% for item in partial2 do
            [% item(2), item(1) %]
        endfor; %] -> info_list;
    */

    true -> pop_pr_quotes;
    [ [%partial2(1)(2), 'Irr'%] [%partial2(2)(2), 'Trea'%]
        [%partial2(3)(2), 'Dan'%] [%partial2(4)(2), 'En'%] ] -> info_list;

    ;;; [info_list is] =>
    ;;; info_list ==>

    save (output_file_name, pr(% [^info_list ^^list_of_inst]
        %));

enddefine;

define insert_names_with_info (input_file, output_file);
    ;;; input_file is a list L of lists
    ;;; info_list is something like
    ;;; [[150 'Irr'] [50 'En'] [50 'Dan'] [50 'Trea']]
    ;;; it means that the first 150 items of L will be called Irr
    ;;;              the next 50 items will be called En
    ;;; and so on and so forth
    lvars list_of_fea, item, number, name, result, counter,
        info_list, partial;
    compile (input_file) -> partial;
    hd(partial) -> info_list;
    tl(partial) -> list_of_fea;
    true -> pop_pr_quotes;
```

```
    [% for item in info_list do
            item(1) -> number;
            item(2) -> name;
            0 -> counter;
            until counter = number do
                [^name] <>[% hd(list_of_fea) %];
                tl(list_of_fea) -> list_of_fea;
                counter + 1 -> counter;
            enduntil;
        endfor; %] -> result;
    ;;; result ==>
    save (output_file, pr(% result %));
enddefine;

vars product, multiply, generate_id;

define show_possible (feature_list) ->
        (num_types, num_variants, type_signatures, rel_list, irrel_list);
    ;;; feature_list has form
    ;;; [relevant [...] ... [...] irrelevant [...] ... [...]]
    feature_list --> ![relevant ??rel_list irrelevant ??irrel_list];
    ;;; rel_list and irrel_list have form
    ;;; [<feature> <value>  ... <value>]
    [relevant features with values are] =>
    rel_list ==>
    [the number of possible object types is] =>
    if null(rel_list) then 0 -> num_types; num_types =>
    else product(maplist(rel_list, procedure(x); length(tl(x)) endprocedure))
      ->num_types;
        num_types =>
        [their signatures are] =>
        generate_id(multiply(maplist(rel_list, tl)), "t") -> type_signatures;
        type_signatures ==>
    endif;

    [irrelevant features with values are] =>
    irrel_list ==>
    [the number of possible variants for each object type is] =>
    if null(irrel_list) then 0
    else product(maplist(irrel_list, procedure(x); length(tl(x)) endprocedure))
    endif; -> num_variants;
    num_variants =>
enddefine;

/*
show_possible ([relevant [colour red green yellow] [shape square circle]
   irrelevant]) =>
*/

;;; A new definition of the procedure to choose the object types that
;;; fall under one category. Instead of entering the feature values,
;;; the actual object type is entered. This allows the specification
;;; of an environment where the concepts to be learnt are DISJUNCTIVE.

vars procedure make_inst_with_types, populate_with_types, make_obj2;
```

```
define choose_cat_with_types (list,cat_names)
        -> (obj_type_signatures, inst_list, populate_list);
    lvars n, answer, rel, irrel,
        categories, overlapping, item1, item2;
    list(3) -> obj_type_signatures;
    list(4) -> rel;
    list(5) -> irrel;
    [for each category choose the object types that fall under it]=>
    [% for item1 in cat_names do
            [choose types for category ^item1]=>
            obj_type_signatures ==>
            [% item1; readline(); %];
        endfor; %] -> categories;
    ;;; categories is something like
    ;;; [[ Trea [type1 type3 type11]]
    ;;;  [ En   [type23 type5]]]
    make_inst_with_types (categories, obj_type_signatures) -> inst_list;
    [TRACE the following categories have been defined] =>
    inst_list ==>
    populate_with_types (inst_list, obj_type_signatures, irrel)
      -> populate_list;
enddefine;


define make_inst_with_types (categories, obj_type_signatures) -> inst_list;
    lvars item1, item2, item3, partial, list1, list2;
    [% for item1 in categories do
            [% hd(item1);
                [% for item2 in hd(tl(item1)) do
                    for item3 in obj_type_signatures do
                        if item2 matches hd(item3) then item3;
                        endif;
                    endfor;
                endfor; %] %]
        endfor; %] -> partial;
    [% for item1 in partial do
            for item2 in hd(tl(item1)) do item2(1); endfor;
        endfor; %] -> list1;
    [% for item2 in obj_type_signatures do
            if not(member(hd(item2), list1)) then item2;
            endif;
        endfor %] -> list2;
    if list2 == [] then partial -> inst_list else
        [OTHER ^list2]::partial -> inst_list;
    endif;
enddefine;

define populate_with_types (inst_list, obj_type_signatures, irrel) -> result;
    lvars item1, item2, partial1, partial2, partial3, aaa;
    ;;; inst_list is something like [[OTHER [[obj2 [...]] [obj3 [...]]]]
    ;;; [TREA [[obj1 [...]] [obj4 [...]]]] ... ]
    [% for item1 in irrel do tl(item1); endfor; %] -> partial1;
    ;;; [partial1 is] =>
    ;;; partial1 =>
    [% for item1 in inst_list do
            item1(1)::[% for item2 in item1(2) do item2(2); endfor %];
```

```
        endfor; %] -> partial2;
    ;;; [partial2 is] =>
    ;;; partial2 =>
    [% for item1 in partial2 do
            item1(1)::make_obj2 (tl(item1), partial1);
        endfor; %] -> partial3;
    partial3 -> result;
enddefine;

define make_obj2 (list1, list2) -> result;
    lvars item1, item2;
    [% for item1 in list1 do
            generate_id(multiply([% for item2 in item1 do [^item2] endfor;
                    %]<>list2), "obj");
        endfor; %] -> result;
enddefine;


define make_proportions (cat_name, types_num, types_list, partial2) -> result;
    lvars item, tot, n, counter, obj_name, prop, num1, partial;
    for item in partial2 do
        if item(1)=cat_name then item(2) -> tot; endif;
    endfor;
    if tot = 0 then 0 -> num1 else div(tot,types_num) -> num1; endif;
    [%  1 -> n;
        0 -> counter;
        until n = types_num do
            ;;; hd(types_list(n)) -> obj_name;
            ;;; [^cat_name has ^types_num types; type in proportion
            ;;; (total is ^tot)] =>
            ;;; hd(readline()) -> prop;
            ;;; round(tot*prop) -> num1;
            ;;; [there will be ^num1 examples] =>
            num1::types_list(n);
            n + 1 -> n;
            counter + num1 -> counter;
        enduntil;
    %] -> partial;
    partial<>[% (abs(tot - counter))::types_list(n) %] ->
    result;
    ;;; [result of makeProportions is ^result] =>
    ;;; result ==>
enddefine;


define help_make_randomly (list_of_cat, partial2) -> result_list;
    lvars item, cat_name, types_num, item1;
    [% for item in list_of_cat do
            item(1) -> cat_name;
            length(tl(item)) -> types_num;
            if not(types_num = 1) then
                ;;; explode(ask_proportions (cat_name, types_num, tl(item),
                ;;; partial2))
                explode(make_proportions (cat_name, types_num, tl(item),
                        partial2))
            else
```

```
                for item1 in partial2 do
                    if item1(1)=cat_name then item1(2)::hd(tl(item)); endif;
                endfor;
            endif;
        endfor; %] -> result_list;
enddefine;

define get_number (item) -> result;
    ;;; item is like [obj30 [large sour shaval2 smeval3 colval5]]
    ;;; result is like [30 [large sour shaval2 smeval3 colval5]]
    strnumber(allbutfirst(3, hd(item))) -> item(1);
    item -> result;
enddefine;

/*
vars prova = [1 2 4]; 3 -> prova(3); prova =>
maplist([ [obj30 [large sour shaval2 smeval3 colval5]]
          [obj30 [large sour shaval2 smeval3 colval5]]], get_number) =>
*/

define make_randomly (list_of_cat, partial2) -> result;
    lvars help_list1, item, help_list2, repetition_n, max_n, value, item1;
    ;;; [starting]==>
    help_make_randomly(list_of_cat, partial2) -> help_list1;
    ;;; [help_list1 is]=>
    ;;; help_list1 ==>
    [% for item in help_list1 do
            ;;; item =>
            hd(item)::maplist(tl(item), get_number)
        endfor; %] -> help_list2;
    ;;; [help_list2 is]=>
    ;;; help_list2 ==>
    ;;; now, help_list2 is something like
    ;;; [ [50 [1 [...]] [2 [...]] ... [n [...]] ] [30 [1 [...]] [2 [...]] ] ]
    ;;; 50 means I need to produce 50 instances
    ;;; choosing from the 1 .. n possible choices that follows
    [% for item in help_list2 do
            hd(item) -> repetition_n;
            hd(last(item)) -> max_n;
            repeat repetition_n times random(max_n)->value;
                for item1 in tl(item) do
                    if item1(1)=value then item1(2); endif;
                endfor;
            endrepeat;
        endfor; %] -> result;
enddefine;

/*
TESTS

vars a = [[OTHER [[obj1 [red L sweet]]] [[obj1 [red L sour]]] [[obj1 [yellow
    S sweet]]] [[obj1 [yellow S sour]]] [[obj1 [yellow M sweet]]] [[obj1
    [yellow L sour]]] [[obj1 [blue S sweet]]] [[obj1 [blue S sour]]] [[obj1
    [blue M sweet]]] [[obj1 [blue M sour]]]] [Trea [[obj1 [blue L sweet]]]
    [[obj1 [blue L sour]]]] [Dan [[obj1 [red S sweet]]] [[obj1 [red S sour]]]
    [[obj1 [red M sweet]]] [[obj1 [red M sour]]]] [Food [[obj1 [yellow
```

```
    M sour]]] [[obj1 [yellow L sweet]]]]];

vars b = [[OTHER 20 0.2] [Trea 20 0.2] [Dan 30 0.3] [Food 30 0.3]];

help_make_randomly (a,b) ==>

make_randomly (a,b) =>
*/

;;; Related procedures

define product(x);
    if null(x) then 1 else hd(x)*product(tl(x));
    endif;
enddefine;

;;; product([2 27 5 10])=>
;;; ** 2700

/*
PROCEDURE: addfronts (list1, list2) -> newlist
INPUTS   : list1, list2
  Where  :
    list1 is a list
    list2 is a list of lists
OUTPUTS  : newlist is a list of lists
USED IN  : multiply
CREATED  : 13 Jun 2000
PURPOSE  : Given a list and a list of lists as its inputs addfronts
           produces as a result a new list of lists where the items in the
           first list have been added in front of every sublist.

TESTS:

addfronts ([a b c], [[1 2] [cat dog fish] []]) ==>
** [[a 1 2]
    [a cat dog fish]
    [a]
    [b 1 2]
    [b cat dog fish]
    [b]
    [c 1 2]
    [c cat dog fish]
    [c]]

addfronts ([[a] [b]], [[1 2] []]) ==>
** [[[a] 1 2] [[a]] [[b] 1 2] [[b]]]

addfronts ([], [[a b c]]) =>
** []

addfronts ([[]], [[a b][c]]) =>
** [[[] a b] [[] c]]

addfronts ([a b c], [[]]) =>
** [[a] [b] [c]]
```

```
addfronts ([a b c], []) =>
** []

*/

define addfronts (list1, list2) -> newlist;
    lvars item1, item2;
        [% for item1 in list1 do
                for item2 in list2 do
                    [^item1 ^^item2]
                endfor;
            endfor;
        %] -> newlist;
enddefine;


/*
PROCEDURE: multiply (lists) -> result
INPUTS   : lists is a list of lists
OUTPUTS  : result is a list of lists
USED IN  : setup_objects
CREATED  : 13 Jun 2000
PURPOSE  : Given a list of sublist as its input multiply produces as a
           result the cartesian product of the sublists.

TESTS:

multiply ([]) =>
** [[]]

multiply ([[]]) =>
** []

multiply ([[][[]][]]) =>
** []

multiply ([[2 3] [cats dogs fishes]]) ==>
** [[2 cats] [2 dogs] [2 fishes] [3 cats] [3 dogs] [3 fishes]]

multiply([[a b c] [1 2] [pp qq]]) ==>
** [[a 1 pp]
    [a 1 qq]
    [a 2 pp]
    [a 2 qq]
    [b 1 pp]
    [b 1 qq]
    [b 2 pp]
    [b 2 qq]
    [c 1 pp]
    [c 1 qq]
    [c 2 pp]
    [c 2 qq]]

multiply ([ [[1 a][1 b][1 c]] [[2 a][2 b]] ]) ==>
** [[[1 a] [2 a]]
```

```
    [[1 a] [2 b]]
    [[1 b] [2 a]]
    [[1 b] [2 b]]
    [[1 c] [2 a]]
    [[1 c] [2 b]]]

multiply ([[red blue green] [Tshape Lshape] [hard soft]]) ==>
** [[red Tshape hard]
    [red Tshape soft]
    [red Lshape hard]
    [red Lshape soft]
    [blue Tshape hard]
    [blue Tshape soft]
    [blue Lshape hard]
    [blue Lshape soft]
    [green Tshape hard]
    [green Tshape soft]
    [green Lshape hard]
    [green Lshape soft]]

*/

define multiply (lists) -> result;
    lvars list;
        [[]] -> result;
        for list in rev(lists) do
            addfronts(list, result) -> result;
        endfor;
enddefine;


;;; redefine generate_id so that the string used as argument for
;;; gensym is given as a parameter

define generate_id (list, string) -> result;
    lvars item, name;
    clearproperty(gensym_property);
    [% for item in list do
            gensym(string) -> name;
            [^name ^item];
        endfor;
    %] -> result;
enddefine;

/*
generate_id ([a b c], "pippo") =>
** [[pippo1 a] [pippo2 b] [pippo3 c]]
*/
```

Appendix B

A detailed worked example and some experiments

B.1 A detailed worked example

In order to make clear how the mechanism we proposed and described in Sections 4.1.3, 5.1.3 and 5.2 can form affordance concepts, we give a detailed worked example of our system[1] We also discuss some limitations of the system and suggest a few preliminary ideas on how to overcome them.

Let's assume an environment has been set up with an adaptive agent inhabiting it and let's assume the adaptive agent is run. At the beginning the agent has no knowledge of its environment, so it has neither a concept of danger nor any other relevant concept. The nearest object in its perceptual field (which we assume to be not empty at start) is therefore set as its first target. Let's assume this first target object is neither a danger item, nor a treasure item, nor a food item, and let's also assume the object has the following attribute values `[Red Sour Small fea1val2 fea2val2 fea3val1 fea4val2 fea5val2]`. Depending on the initial setting of the internal energy parameter, the agent will choose its first action to perform on the target object, once in contact with it. So, let's assume that at start the agent is not hungry, and hence will perform the action `collect_treasure` on the target. Because of the built-in associations between actions and expectations on action results, the agent will expect no increase of its internal level of pain and an increase of its internal level of happiness. However, since the object is not a treasure item, the second of the agent's expectations will fail. A representation of the target object (namely, a list containing the object's attribute values) will therefore be stored in the learning database of the agent both as a negative example of the concept of Danger and as a negative example of the concept of Treasure.

Since the learning algorithm that we used is initialised to the first positive instance met, after this first experience the agent's concepts of Danger and Treasure will still be empty. Again, the agent will set as its target the nearest object in its perceptual field and will perform the action `collect_treasure` on it. Let's assume the new target object is indeed a treasure item and has the following attribute values `[Blue Sweet Medium fea1val3 fea2val2 fea3val2 fea4val2 fea5val1]`. Following this second experience, the agent will develop its first concept of Treasure as things that look like the first treasure item it encountered. Afterwards, if an object with these same attribute values happens

[1] The example is extracted from an actual experiment that has been performed with the implemented system.

to be in the agent's perceptual field and the agent is still motivated to perform the action `collect_treasure`, such an object will be selected as the new target, even though other objects may be closer to the agent.

It may happen however that no object with such characteristics is in the agent's perceptual field: in this case, again the nearest object will be approached. Let's assume this to be the case, and that the new target is indeed a danger item with the following attribute values `[Red Sweet Large fea1val3 fea2val3 fea3val1 fea4val1 fea5val2]`. After performing the action `collect_treasure` on the target, the agent's expectation on its internal level of pain not to increase will fail, hence a representation of the target object will be stored as a positive instance of the concept of danger, and the concept of danger itself will be initialised to this same representation. Afterwards the agent will avoid approaching any object that looks like this first danger item that has been encountered.

Now the agent's concepts of danger and treasure are not empty any more, so in the next lifecycle, while examining its perceptual field, the agent will partition the perceived objects in three classes: those that look like danger items to it and that will be avoided, those that are relevant to it because they look like treasure items (we are assuming that the agent continues to be motivated to perform the action `collect_treasure`), and all the others. If the relevant objects class is not empty, the agent will approach the nearest of those objects, otherwise it will approach the nearest object which is not subsumed by its concept of danger. Let's assume this second one to be the case and let's assume the chosen target to be again a danger item but with the following attribute values `[Red Sweet Medium fea1val3 fea2val3 fea3val2 fea4val2 fea5val3]`. After performing the action `collect_treasure` on the target, the agent's expectation on its internal level of pain not to increase will fail, hence a representation of the target object will be stored as a positive instance of the concept of danger which has been misclassified. As a consequence, the agent will revise its concept of danger and more precisely will try to make it slightly more general in order to cover the new positive instance, wihtout subsuming any of the negative instances that are still in memory. The revised concept of danger will be `[Red Sweet fea1val3 fea2val3]`.

After a while, the agent's internal level of energy will drop below a given threshold and the agent's motivation to act will therefore change to `collect_energy`. The agent's expectations on the results of its action will also change: though the agent will still expect not to be hurt, it will also expect its internal level of energy to increase, and will therefore start forming a concept of food. As time goes by, the agent will revise its concepts of danger, treasure and food (either generalising or specialising them) to accommodate as many as possible of the experiences kept in memory, and eventually discover the actual causal classes set by the designer of its environment.

As we already explained in Section 4.1.3, the agent's architecture that we propose to learn affordance concepts cannot form concepts relating to learnt, and not innate, expectations, that is affordance concepts for which the functional aspect is not already built in. This problem has been discussed at length in Section 6.2.1, where some preliminary ideas have been proposed to weaken the innateness assumption. We refer the reader to that Section for more details.

We would like however to mention one further difficulty, which is mainly due to the particular learning algorithm that we have chosen. Since the incremental separate and conquer algorithm that we have been using is very sensitive to the history of presentation of the training instances, the agent may face problems when handling misclassified negative instances: more precisely, when the algorithm specialises because a negative instance has been misclassified and no positive instance is in memory (the algorithm retains up to the most recent k training instances), it may happen that a specialised hypothesis is generated which is groundless. For example, after being presented with a set of training instances containing, among others, `[red small sour]` and `[red medium bitter]` as positives, the algorithm could have produced the over general hypothesis `[red]` rather than the disjunct `[red sour] or [red bitter]`. After a while, when presented with the negative instance `[red small sweet]` (and assuming no positive instance is among the last k training instances), the algorithm may produce `[red large]` or `[red salty]` as plausible, more special hypotheses for the concept to be learned, even though such hypotheses would not cover many positive examples of that concept. In order to prevent this problem, either a very large k parameter is used (thus slowing down the processing) or it is ensured that at least some of the positive training instances encountered are also kept in memory and used to guide the specialisation process. Another possibility is to carefully select the training examples and the sequence in which they are presented.

B.2 Experiments with the implemented system

Even though the system presented in AppendixA has not been tested extensively, we did run two sets of experiments in order to assess:

- the better performance of the adaptive agent, with respect to the reactive one; and
- the quality of the model learned by the adaptive agent.

For the first set of experiments, we used a relatively "friendly" environment, with the following characteristics: the relevant object features are colour (with red, blue, yellow, green and white as possible values), size (with small, medium and large as possible values) and taste (with salty, sour, sweet and bitter as possible values); treasure items are set to be either green and large or white and large; danger items are set to be either red and small or red and medium; and food items are set to be either yellow and medium or yellow and large (see file `env-exp1.p`). The environment contains 240 objects and is said to be "friendly" because it contains a high percentage of food items, namely 50%; the remaining part of objects is composed for 25% of treasure items and for 25% of danger items, and there are no irrelevant objects.

Since there are a lot of food items, the agent could start with a low initial energy, allowing it to meet just a few objects, and then it could get again just a little bit of energy when recharging. However, if this is the case, then the agent will be recharging most of the time. And when it is looking for treasure, since treasure items are not abundant, it will often meet food items or danger items instead. In the long run, the adaptive agent will probably discover also what treasure items and danger items look like, but the actual discovery will depend on whether the energy parameters allow the agent to live long enough.

To compare the performance of the two agents, we first determined empirically (using the reactive agent) the energy units that are used, on average, during one life cycle, and the number of cycles that is needed, on average, to meet one object. This allows us to calculate the energy units that the agent will need as initial energy and as recharging energy, given the number of objects that we would like it to visit. We then made some initial tests to assess the behaviour of the two agents, given the setting of the relevant initial parameters, namely the initial energy that the agent has, its hunger threshold and the energy units that are acquired when recharging. In Table B.1 we present the data obtained from 10 simulations done using very low initial energy and recharging energy parameters (400 units and 450 units respectively, that is less than what is needed to visit 2 objects) and a hunger threshold of 800 energy units. For each run and for each agent, we limited the scheduler to 900 cycles and we counted the number of lifecycles that the agent survived. It is interesting to see that also with such "rough" energy parameters, the adaptive agent outperforms the reactive one and lives much longer.

simulation run	n. of cycle survived by reactive agent	n. of cycle survived by adaptive agent
1	276	307
2	372	still alive
3	375	still alive
4	386	637
5	319	still alive
6	312	788
7	424	still alive
8	330	327
9	306	313
10	459	still alive
average	356	687

Table B.1: Data results for experiment 1.

To assess the quality of the model learned by the adaptive agent we used an environment with the following characteristics (see Figure B.1 and file `env-exp2.p`): the environment contains 240 objects of which 30% are treasure items, 20% are danger items, 20% are food items, and 30% are irrelevant to the agent. Moreover, treasure items are either (blue and medium) or (blue and large) or (white and sweet); danger items are either (red and sweet) or (blue and small); and food items are either (red and sour and large) or (yellow and sour and large) or (green and medium).

The model to be learned by the agent is a collection of disjuncts, so in order to compare the model set by the designer and the model acquired by the agent, we assigned:

- 3 points for any correct disjunct (with respect to the designer model) produced by the agent;
- 1 point for each correct conjunct present within any disjunct produced by the agent at the right level of generality (with respect to the designer model);
- 0 points for any disjunct produced by the agent which is too special (with respect to the designer model).

The designer model is worth 24 points, namely 3 points for each disjunct. As an example, we calculate the points associated to the following model (that might have been produced by the agent):

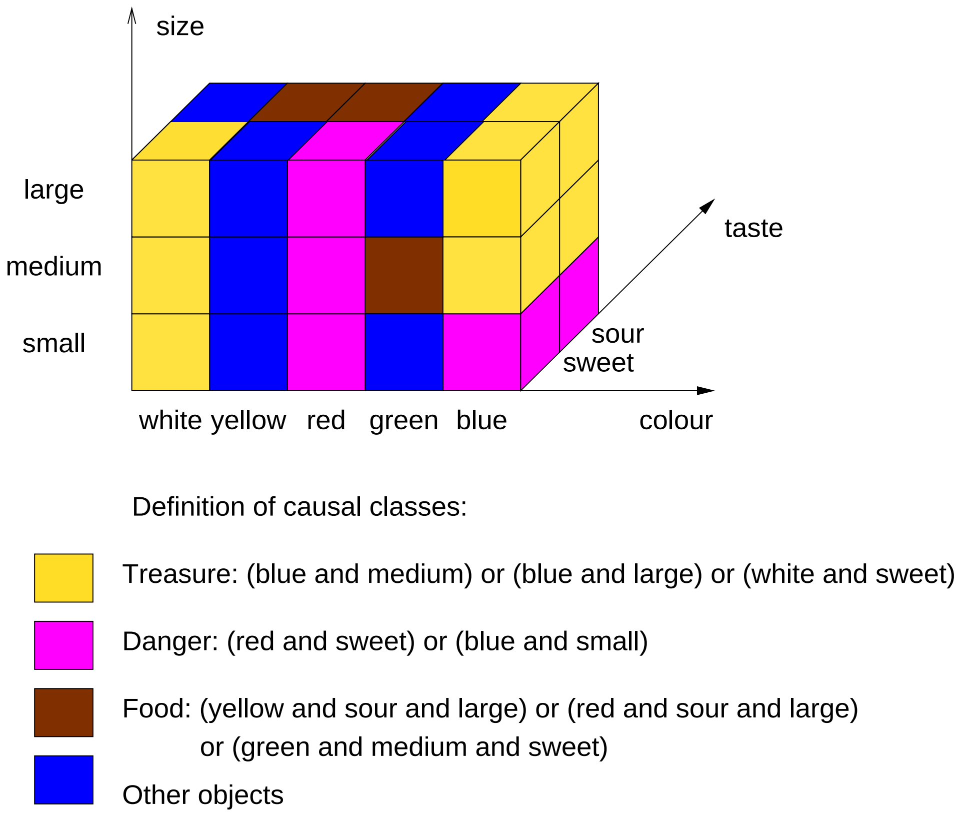


Figure B.1: Designer model to assess the quality of the model learned by the adaptive agent.

cycle number	medium value of model learned
100	7
200	5
300	7
400	4
500	9
600	9
700	17
800	17
900	17
1000	18
1100	18
1200	18
1300	19
1400	18
1500	19

Table B.2: Aggregated data results for experiment 2.

```
[[isa danger_item [Bl sweet]]
 [isa food_item [Gr sour]]
 [isa treasure_item [Wh sweet]]
 [isa treasure_item [Bl sour]]]
```

The agent's concept of Danger has only one disjunct, at the right level of generality, but with only one correct conjunct (blue), hence we count 1 point. The agent's concept of Food has only one disjunct, at the right level of generality, but with only one correct conjunct, hence we count 1 more point. The agent's concept of Treasure has two disjuncts, both at the right level of generality: (white and sweet) is correct, hence we count 3 points; (blue and sour) has only one correct conjunct, namely blue, hence we count 1 point. The whole agent's model is therefore worth 6 points.

Table B.2 shows the aggregated results for 10 random runs of the adaptive agent, each of 1500 cycles, with the environment specified by file `env-exp2.p`. More precisely, the table shows the medium value of the model formed by the agent every 100 lifecycles: even though the quality of the model learned does not increase constantly, as time goes by the agent tends to discover the actual model set by the designer.