Ethogenetics and the Evolutionary Design of Agent Behaviors

Sébastien PICAULT and Samuel LANDAU
LIP6, Université Pierre et Marie Curie
75 252 Paris CEDEX 05, FRANCE

ABSTRACT

This paper introduces the concept of *Ethogenetics* which addresses the issue of the design of evolvable agent behaviors. We first emphasize the need of a new approach towards existing evolutive approaches for this field, and outline the principles of Ethogenetics. We then describe a model, *ATNoSFERES*, that we use to experimentally investigate this concept. Finally, we discuss the properties and features of the ATNoSFERES model and propose further extensions.

Keywords

Ethogenetics, Evolutionary Computing, Agent Behavior, ATN, Control Architectures

1 INTRODUCTION : LIMITATIONS OF EXISTING APPROACHES

The purpose of Ethogenetics (EG) is the design of evolvable agent behaviors, possibly with agents operating at different organization levels. The behavior of the agents of each level may either be given, or be *built* from an hereditary substrate (a bit string called "chromosome"). This new approach is an attempt to overcome problems raised by classical evolutionary paradigms in the design of *behaviors*.

Genetic Algorithms [1, 2, 3] and Evolutionary Strategies [4, 5] allow non-semantic manipulations of the genotypes (strings of bits or arrays of floats), inducing in most cases gradual modifications in the phenotypes. Thus they ensure that individuals produce children with quite the same level of adaptation to the environment than their own. However, these evolutionary algorithms have a very poor expressive power, since their purpose is the optimization of a set of parameters [6] in behaviors which *have to be given a priori*.

On the contrary, the Genetic Programming paradigm [7] is based on the evolution of *programs* (i.e. instructions trees), thus their expressive power is much higher. But the genetic operators associated with trees operate in a syntactic way on semantic structures, so they induce strong variations in the effects of the resulting program ; in addition, the impact of the genetic operators tightly depends on the level at which they operate (a modification near the root is likely to have a deeper influence than one on a leaf). Moreover, the behavior of an agent cannot be reduced to a program.

Thus, for designing evolvable agent behaviors, there is a need for an approach that would be able to combine the advantages of Genetic Algorithms and Genetic Programming. The Ethogenetics approach has been designed in order to address this issue.

2 ETHOGENETICS

Ethogenetics provides general principles to the design of evolutive agent behaviors – the ability to *build* agent behaviors (with a large expressive power) from a meaningless genetic substrate. Since Darwinian evolution is a blind process, its use to produce agent behaviors and collective behaviors in multi-agent systems implies several properties, mainly consequences of two principles : continuity and expressive behavioral power.

2.1 Continuity

The environmental selection pressure acts on the whole system, on its ability to react, to perform a task, to reach collective goals and so on. Adapted systems are selected to produce "offspring" : other systems, the genotype of which is a mixture of those of their "parents". The adaptation degree of the offspring systems should be close to their parents ones (if not, the adaptive effect of natural selection gets lost). Thus, the behavior building process has to be :

- *robust towards mutations* : small variations in the genotype should induce in most cases only small variations in the phenotype;
- *independent from the structure of the genetic substrate* : unlike Genetic Programming (where the hierarchical tree structure has heavy consequences), distant parts of the genotype should have few effects on each other. Thus it is useful to dissociate the semantic structure (that produces the behavior) from the "syntactic" one (the genetic substrate). When this requirement is not fulfilled, semantics tightly constrains syntax, so that syntactic manipulations (resulting from "blind" genetic operations) often destroys the semantic structure.

2.2 Expressive behavioral power

The second major requirement in order to produce *agent behaviors* is the ability to design complex behaviors. Thus the semantic structure used for that goal should have at least the expressive power of a program tree (a tree provides more interesting features as a semantic structure rather than as a syntactic structure). But these behaviors, even if complex, should meet some requirements :

- *Behaviors should be understandable*. It may be useful to provide the agents with *understandable behaviors* : some control architectures such as artificial neural networks might be very efficient, but the resulting behavior cannot be clearly described. The ability to easily interpret the behaviors would allow on the one hand to understand *what* has been selected, and on the other hand, to explicitly specify some of the behaviors using this same structure, allowing to set *a priori* the behaviors of some agents.

- *Behaviors should be able to adapt*. Since the system will have to operate in a given environment, it should be able to adapt itself, to reconfigure according to environmental constraints. Thus the semantic structure representing behaviors should avoid using explicit parameters : parameters are a kind of shortcut, they reflect prior knowledge about the environment. The building of *situated behaviors* has to be independent from any parameters, in order to keep more flexibility.

In the next section, we present ATNoSFERES, a model aimed at investigating Ethogenetics properties.

3 DESCRIPTION OF THE ATNoSFERES MODEL

3.1 General principles

Ethogenetics [8] are part of a larger project, ATNoSFERES [9], which uses the SFERES [10] framework as a tool for modelling the agents classes, integrating those classes to the system, designing an environmental simulator and providing classical evolutionary techniques.

In particular, ATNoSFERES provides a general class, the `ATNAgent`, the behavior of which is produced through the following steps:

1. a translator produces tokens from the string,
2. an interpreter uses these tokens as instructions to build a graph (an ATN[1]),
3. finally, the graph determines the behavior of the agent.

The translator and the interpreter themselves are agents; in the following lines, we will consider that their behavior is given, but it could evolve as well to provide the system with higher autonomy.

[1] ATN stands for Augmented Transition Network

3.2 The ATN Graph and the `ATNAgent`

The `ATNAgent` class is intended to behave according to an ATN graph [11]. ATN have previously been used by [12] for designing agent behaviors. Each subclass of `ATNAgent` is associated with two collections of tokens: condition ones and action ones. The actions are behavioral "primitives" that can be performed by the agent, the conditions are perceptions or stimuli that induce action selection. The edges of the graph are labeled with a set of conditions and a sequence of actions (see figure 1).

The ATN built by adding nodes and edges to a basic structure containing two nodes: a "Real Start Node" and a "Real End Node". At each time step, the agent (initially in the "Real Start Node" state) randomly chooses an edge among those having either *no condition* in their label, or *all conditions simultaneously true*. It performs the actions associated with this edge and jumps to the destination node. It stops working when its state is the "Real End Node".

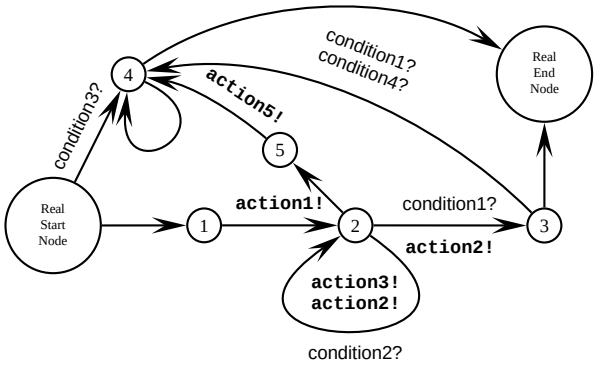

Figure 1: An example of ATN.

3.3 The Interpreter

The purpose of the interpreter is to build an ATN from tokens. Some of these tokens will be action or condition ones that are used to label edges between nodes in the ATN. The other ones are interpreted as instructions, either to create nodes or connect them, or to manipulate the structure under construction.

As we mentioned in section 1, the structure built by the tokens sequence has to be robust towards mutations. For instance, the replacement of one token by another, or its deletion, should have only a *local impact*, rather than transforming the whole graph. Therefore, we use a "top-level" programming language operating on a list (see table 1).

If an instruction cannot execute successfully, it is simply ignored, except instructions operating on nodes (i.e. *connect* and *dupObject*) which are "pushed" in the list until new nodes are produced; then they try to execute again with the new data. Finally, when the interpreter does not receive tokens any more, it terminates the ATN: actions and conditions tokens still present between nodes are treated as *implicit connections* (so that new edges are created) and the consistency of the ATN is checked ("Real Start Node" is linked to

token (initial list state)	→	(resulting list)
condition? $(x \ ...)$	→	(condition? x ...)
`action!` $(x \ ...)$	→	(**`action!`** x ...)
node $(x \ ...)$	→	$(N_i \ x \ ...)^a$
startNode $(x \ ...)$	→	$(N_i \ x \ ...)^b$
endNode $(x \ ...)$	→	$(N_i \ x \ ...)^c$
connect (c2? $x \ N_i \ y$ - - c1? z **`a2! a1!`** $t \ N_j \ u \ ...$)	→	$(x \ N_i \ y \ z \ t \ N_j \ u \ ...)^d$
dup $(x \ y \ ...)$	→	$(x \ x \ y \ ...)$
dupObject $(x \ y \ N_i \ z \ ...)$	→	$(N_i \ x \ y \ N_i \ z \ ...)$
popRoll $(x \ y \ ... \ z)$	→	$(y \ ... \ z \ x)$
pushRoll $(x \ ... \ y \ z)$	→	$(z \ x \ ... \ y)$
swap $(x \ y \ ...)$	→	$(y \ x \ ...)$
forget $(x \ y \ ...) + [z \ ...]$	→	$(y \ ...) + [x \ z \ ...]^e$
recall $(x \ ...) + [y \ z \ ...]$	→	$(y \ x \ ...) + [z \ ...]^e$

[a] creates a node N_i
[b] creates a node N_i and connects "RealStartNode" to it
[c] creates a node N_i and connects it to "RealEndNode"
[d] creates an edge between N_j and N_i, with (c1?& c2?) as condition label and the list {**`a1!`**,**`a2!`**} as action label
[e] with an auxiliary stack

Table 1: The ATN-building language.

nodes having no incoming edges, except from themselves; in the same way, nodes having no outgoing edges are linked to "Real End Node").

3.4 The Translator

The translator has a very simple behavior. It reads the genotype (a string of bits) and decodes it into a sequence of tokens. It uses a *genetic code*, i.e. a function

$$\mathcal{G} : \{0,1\}^n \longrightarrow \mathcal{T} \qquad (|\mathcal{T}| \leq 2^n)$$

where $\mathcal{T}$ is a set of tokens, which includes both action and condition ones (specific to the agent to build) and those understood by the interpreter (see table 1).

Depending on the number of tokens available, the genetic code might be more or less redundant. If necessary, it can be designed in order to resist mutations, but we will not discuss this issue in this paper.

4 EXPERIMENTS

The purpose of this section is both to demonstrate how the system works by using it on a simple example (with a single class of agents operating at the same level), and to provide some results regarding the behaviors that evolved in the following experiments.

4.1 Experimental setup

To illustrate the evolution of simple behaviors, let us consider an experiment with a discrete environment containing a color light bulb and a single agent (instance of a subclass `ATNAnimat` of the general `ATNAgent` class) with the action and perception abilities described in table 2. We want the agent to go to the right when the light is green and to the left when it is red. To make the agent behavior evolve, we apply the rules of darwinian selection over a population of 100 homogeneous agents.

Actions		**Conditions**	
`N!`	no action		
`R!`	move to the right	g?	true when the light is green
`L!`	move to the left	r?	true when the light is red
`U!`	move up	rand?	true with probability $p = 0.5$
`D!`	move down		

Table 2: Action and condition tokens of the `ATNAnimat` agent.

The genetic code for these agents contains the 11 interpreter tokens and the 8 action/condition tokens; thus it needs at least $32=2^5$ codons. In the following experiments, a 32-codon genetic code has been automatically built from a circular list containing the interpreter and action/condition tokens. The chromosome of the agents is initially a *random* bit string with a *random length* (from 200 to 300 bits).

We evaluate the fitness of each agent by making it run during 100 time steps in its environment. The color of the light bulb randomly flips (with probability 0.05 at each time step) from green to red and vice-versa. The rewarding rules in the fitness function are: +1 point if the move is correct, -1 point if it is erroneous (e.g. left when green), 0 in the other cases (e.g. move up). Only the first move performed during the current time step is rewarded ("do nothing" is not considered as a move).

At each generation, the agents are evaluated through their average fitness (calculated over 10 runs in the above conditions) and selected to produce 30 new agents (by crossing over chromosomes), thus replacing 30 agents removed from the old population (depending on their fitness, too).

We have experimented with various mutation strategies, among them:

1. before their evaluation, *all agents* are subject to punctual random mutations of their chromosome with rate r (r % of the bits are randomly flipped);

2. same situation, but p % of the mutations are random insertions or deletions of codons in the chromosome, instead of punctual mutations.

4.2 Results

As agents are initialized in their "Real Start Node" state, the first time step is used to jump to one of the available nodes.

Then, during the 99 other time steps, the behavior of the agents only depends from their ATN structure and its ability to respond to environmental changes. Thus, the maximum fitness in these experiments is 99 (correct answers at each time step after the first one).

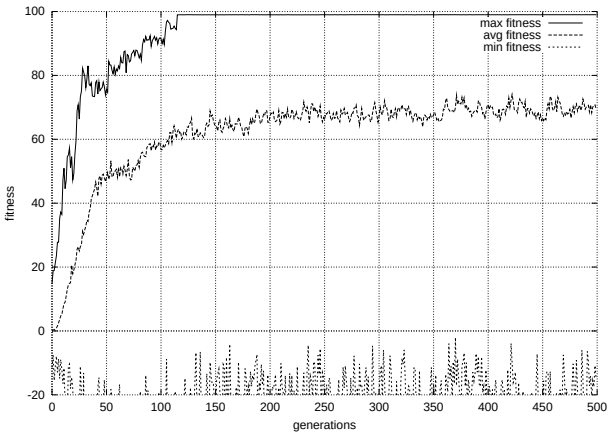

Figure 2: Evolution of the fitness (punctual mutations only, r = 1 %).

Figure 2 shows the average evolution of the fitness with the first mutation strategy. It has been calculated from 10 experiments. Figure 3 shows the average evolution of the fitness with the second mutation strategy, and the evolution of the chromosome length (10 experiments too). In a few cases (almost in the first strategy), no good solution is found before 500 generations.

5 DISCUSSIONS

5.1 Results analysis

Though the problem to solve is very simple, these results validate the ATNoSFERES model for evolving agent behaviors from a fine-grain substrate. But we would like to focus on the specificity of this model and give a qualitative analysis of the behaviors produced by natural selection.

The ATN described on figure 4 is the "optimal" solution to this problem (99 points with the simplest ATN structure): when the light is red, go to the left; when it is green, go to the right. To produce this ATN, only 35 bits are theoretically required, for example to encode the following tokens:

node, g?, **R!**, *dupObject*, **L!**, r?, *dupObject*

(with a maximal use of implicit connections). But in this solution the order and nature of tokens is crucial, thus it is highly vulnerable to mutations. In addition to this, the agents have much more bits in their chromosome than necessary – this can be a source of inadequate behavior.

The experimental results show that two strategies are used to produce an adequate behavior (99 points). The first one

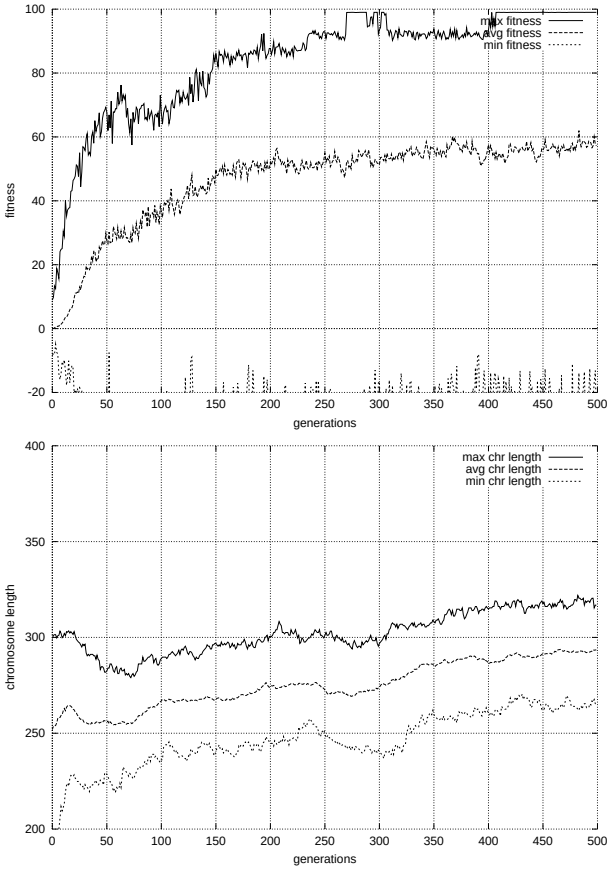

Figure 3: Evolution of the fitness and chromosome length (random insertions and deletions, r = 1 %, p = 20 %)

consists in building a simple ATN (very close or same than the "optimal" ATN on figure 4), by delaying the node creation and thus using tokens that have no effect. For instance, a 207-bit chromosome encoding the following tokens:

L!, *dup*, *swap*, *popRoll*, *popRoll*, *forget*, *popRoll*, *swap*, *forget*, *recall*, *recall*, **R!**, **L!**, *recall*, rand?, **L!**, *forget*, *startNode*, **R!**, *dup*, g?, *dupObject*, *popRoll*, *pushRoll*, **L!**, *forget*, *pushRoll*, **L!**, r?, *startNode*, *forget*, *dup*, *dupObject*, r?, **R!**, *forget*, *dup*, **R!**, *dup*, g?, *popRoll*

produces an ATN very close to figure 4, with labels (r?, **L!L!R!R!**) on one edge and (g?, **R!R!**) on the other. This is a good example of using the properties of the ATN-building language.

The agents using the second strategy build a complex ATN in which a small subset only is used, for instance like the ATN on figure 5. It leads to exactly the same behavior than the "optimal" solution, since a large part of it cannot be reached from the other nodes.

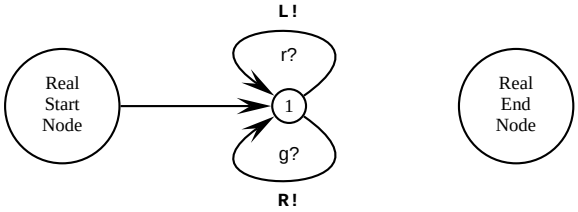

Figure 4: The "optimal" ATN (providing the highest fitness with the simplest structure).

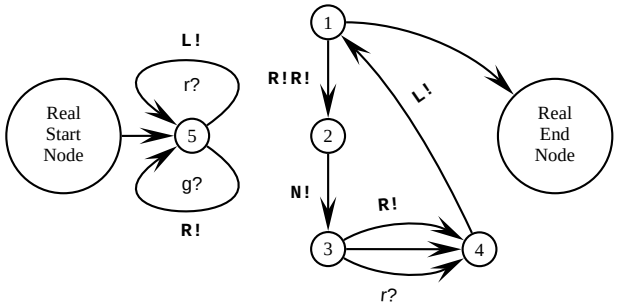

Figure 5: An ATN built by natural selection, implementing the best behavioral strategy.

5.2 The ATNoSFERES model with regard to Ethogenetics

The ATNoSFERES model fulfills the Ethogenetics requirements expressed in section 2. Preliminary experiments [9] [13] have validated the use of ATNoSFERES regarding the following aspects:

- the ability to evolve *adequate* agents behaviors in a simple situation, from *random graphs*;
- the consistency of the ATN-building evolutionary language.

The experimental results have also confirmed that the generation of behaviors do not rely on a precise structure in the genotype: various adequate solutions have been found, based either on different graph-building strategies, or on the use of properties of the graphs (more details can be found in [9]).

5.3 Advantages of the ATNoSFERES model

As an evolutive approach, the ATNoSFERES model provides three main features.

First, it separates the genetic information structure (plain bit string, the lexical structure) from its interpretation (ATN, the semantic structure). Thus, the semantic structure built is *always correct*. The behavior described by the ATN always has a meaning – even if it is not adequate to the environment. Not having to worry about the syntactic correctness of the automatically designed semantic structure is a good point over many other evolutive approaches.

The second main evolutive features are related to the genetic operators. The level of influence of the classical genetic operators – mutation and crossover – does not depend on the parts of the bit string they involve (neither on their location in the bit string nor on their size). This is also a main advantage over many evolutive approaches. As a matter of fact, mutations only have a local impact in the expression of the genetic information, and crossovers involve bit substrings which carry locally functional genetic code. We might also consider more exotic genetic operators, such as deletions/insertions in the bit string. These operators in particular permit to smoothly manage string resizing, since they only have a local impact in the ATNoSFERES model.

The third feature is that the model does not use any parameter to build behaviors. The behaviors execution only depends on environmental conditions, thus hard-coded genetic parameters are not even needed. Apart from behaviors design, parameters encoding is a problem in many evolutive approaches, (see for example discussion on epistasis in [14]), but as long as building behaviors is concerned, we think it should be considered not to rely on fixed parameters in order to produce situated, adaptive behaviors.

As a model for designing multi-agent systems, the ATNoSFERES model does not set any restriction on the agent level specification. Furthermore, agents can be introduced later on at a lower organization level (for instance inside an agent), keeping the latter structure, if a finer-grain agent specification is needed. A `CompositeAgent`, derived from the `ATNAgent` class, was introduced in ATNoSFERES to encapsulate finer-grain agent specifications. The description of the `CompositeAgent` and the agent level specification issues in ATNoSFERES have been developed and discussed in [13].

As a model for automatic behavior design (as part of ATNoSFERES), ATNs use a simplified framework, where only the conditions and actions of the agents have to be specified. Then, with an ATN as the structure for behavior description, it is possible to directly describe and explain the behavior of any agent.

6 CONCLUSIONS AND PERSPECTIVES

We have presented Ethogenetics, an approach for evolutive agents behaviors, and discussed its specific features. To summarize, interesting agents behaviors can be built through an evolutionary approach that is able to ensure *continuity* between the genetic substrate and the phenotypic behavior, and a *high expressive power* in the behavior produced. We propose therefore a two-step building that leads to graph-based behaviors (the ATNoSFERES model).

We have then presented the ATNoSFERES model, and a simple problem solved with ATNoSFERES. We made an analysis of its results in order to exhibit the model specific

features and to emphasize that it validates the main principles of Ethogenetics as an evolutive behaviors design approach.

The ATNoSFERES model is currently under experimentation for more complex behavior design, especially in the case where some agents are made up of finer-grain agents.

In further works, we plan to introduce a metabolic regulatory mechanism, associated with the actions within the ATN. It will act as an environmental constraint, allowing or disabling some edge transitions at a given time.

REFERENCES

[1] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. Ann Arbor: University of Michigan Press, 1975.

[2] K. A. De Jong, *An analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, Dept. of Computer and Communication Sciences, University of Michigan, 1975.

[3] D. E. Goldberg, *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley Pub. Co., 1989.

[4] I. Rechenberg, *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzi pien der biologischen Evolution*. Stuttgart: Frommann–Holzboog, 1973.

[5] H.-P. Schwefel, *Evolutionsstrategie und numerische Optimierung*. Dr.-Ing. Thesis, Technical University of Berlin, Department of Process Engineering, 1975.

[6] T. Bäck and H.-P. Schwefel, "An overview of evolutionary algorithms for parameter optimization," *Evolutionary Computation*, vol. 1, no. 1, pp. 1–23, 1993.

[7] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, Massachusetts: MIT Press, 1992.

[8] S. Picault and S. Landau, "Ethogenetics - A Darwinian Approach towards Individual and Collective Agents Behavior," technical report (in press), LIP6, Paris, 2001.

[9] S. Landau, S. Picault, and A. Drogoul, "ATNoSFERES: a Model for Evolutive Agent Behaviors," in *Proc. of AISB'01 symposium on Adaptive Agents and Multi-agent systems*, 2001.

[10] S. Landau, S. Doncieux, A. Drogoul, and J.-A. Meyer, "SFERES, a Framework for Designing Adaptive Multi-Agent Systems," technical report, LIP6, Paris, 2001.

[11] W. A. Woods, "Transition networks grammars for natural language analysis," *Communications of the Association for the Computational Machinery*, vol. 13, no. 10, pp. 591–606, 1970.

[12] Z. Guessoum, *Un environnement opérationnel de conception et de réalisation de systèmes multi-agents*. PhD thesis, LIP6 – Université Paris VI, may 1996.

[13] S. Landau and S. Picault, "Modeling Adaptive Multi-Agent Systems Inspired by Developmental Biology," in *Proc. of the Workshop on Adaptability and Embodiment using Multi-Agent Systems of ACAI'01*, 2001.

[14] R. Salomon, "Increasing Adaptivity through Evolution Strategies," in *From animals to animats 4, Proc. of ther fourth international conference on simulation of adaptive behaviour* (P. Maes, M. J. Mataric, J.-A. Meyer, J. Pollack, and S. W. Wilson, eds.), pp. 411–420, MIT Press, 1996.