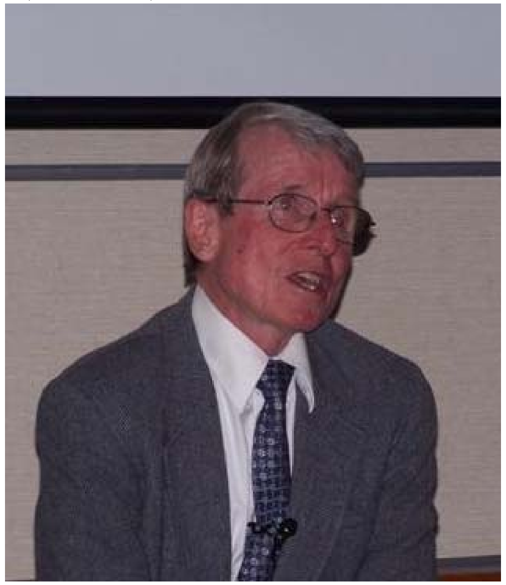
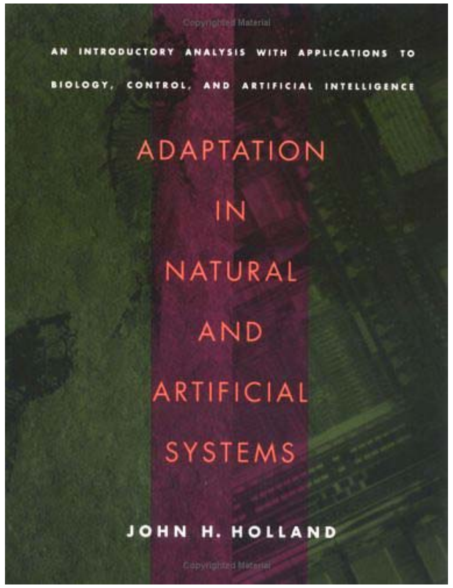
Theory and applications of simple Learning Classifier System (LCS)

Vladimír Kvasnička
Institute of Applied Informatics
FIIT STU

History of Learning Classifier System (LCS)?

Initially introduced by John Holland in his famous book *Adaptation in Natural and Artificial Systems* (1976)




AN INTRODUCTORY ANALYSIS WITH APPLICATIONS TO BIOLOGY, CONTROL, AND ARTIFICIAL INTELLIGENCE
ADAPTATION IN NATURAL AND ARTIFICIAL SYSTEMS
JOHN H. HOLLAND

In this book he introduced simultaneously **Genetic Algoritm (GA)** and **Learning Classifier Systems (LCS)**

Recently, LCE theory is very intensively developed and applied to different interesting examples. Main sources to LCE may be found in the following three books:

[1] Jan Drugowitsch: *Design and Analysis of Learning Classifier Systems. A Probabilistic Approach*. Springer, Berlin, 2008.

[2] L. Bull and T. Kovacs: *Foundations of Learning Classifier Systems: An Introduction*. Springer, Berlin, 2005.

[3] Martin V. Butz: *Rule-Based Evolutionary Online Learning Systems. A Principled Approach to LCS Analysis and Design.* Springer, Berlin, 2006.

An extensive bibliography may be found on ftp address:
ftp://math.chtf.stuba.sk/pub/vlado/LearnigClassifierSystem/Bibliography_lcs.pdf

Specification of LCS

- LCS is a machine learning symbolic ***approach*** which combines GA, reinforcement learning, and heuristics to produce adaptive systems.
- They are rule-based systems, where the rules are usually in the traditional production system form of “IF state THEN action”.
- GA and heuristics are used to search the space of possible rules, whilst a credit assignment algorithm is used to assign utility to existing rules, thereby guiding the search for better rules.
- The main superiority of LCS with respect to standard neural (symbolic) approaches is:

  (1) The produced symbolic rule system has ***simple straightforward interpretation***, and

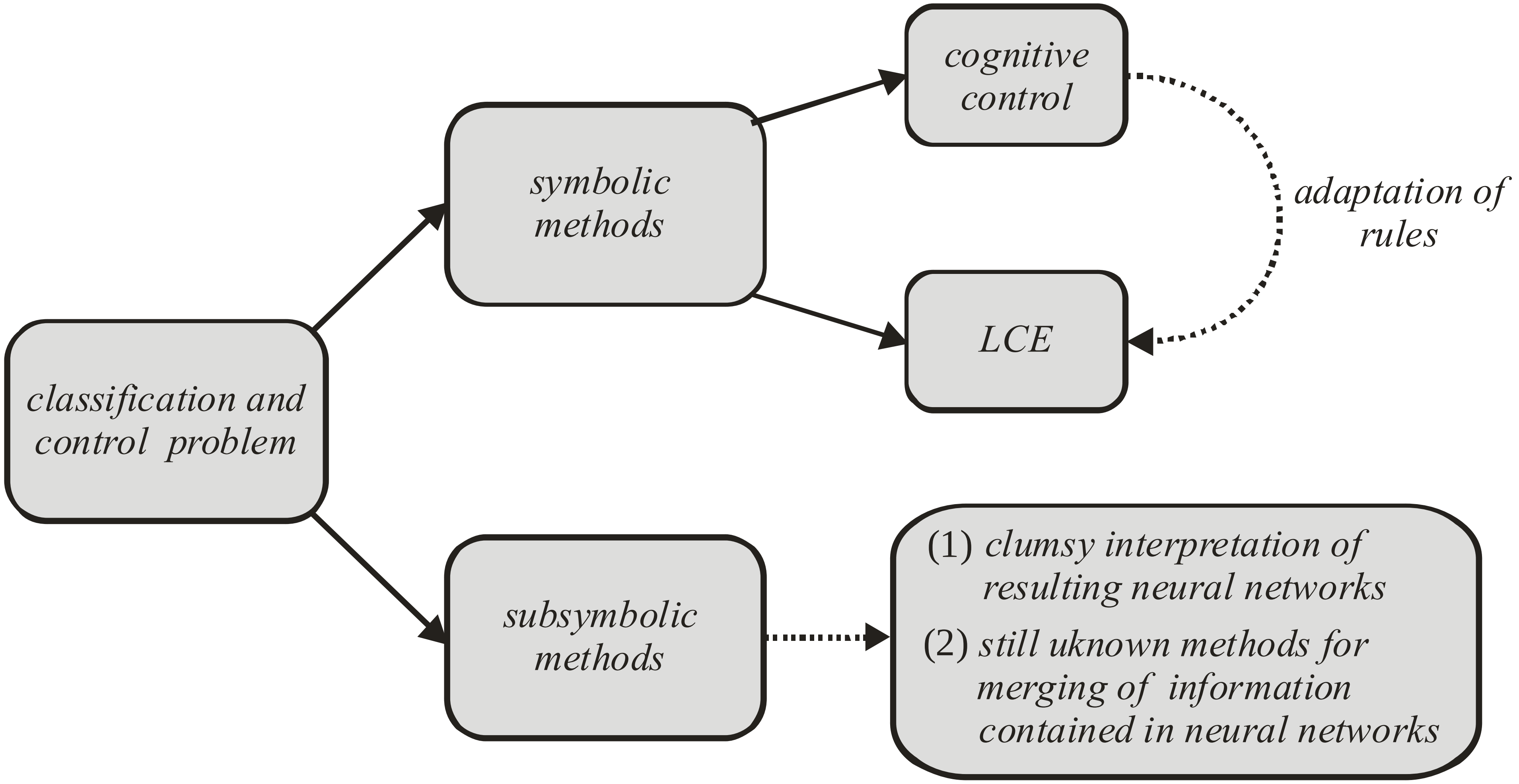
  (2) rule systems offered by two different agents controlled by LCE may by simply ***merged*** into one bigger consistent system.

Why it is used in recent days, when we have available many effective subsymbolic techniques?

- Contemporary AI is oriented mainly to solution of higher cognitive activities (reasoning, planning,…), where subsymbolic approaches are clumsy and not very effective.
- This is a main reason why in contemporary AI exists a renaissance of symbolic techniques that are combined with modern ***non-neural*** subsymbolic techniques (e. g. evolutionary algorithms, reinforcement learning, and fuzzy logic).
- In the course of last a few years the so-called ***cognitive control*** is created, where a standard control by neural networks is substituted by symbolic expert system (a set of rules *if*…, *then* …), which specifies what has to do a supervisor controlling a given plant.

- LCE offers for cognitive control very effective and robust technique for classification and control complex environments.
- The main problem in the cognitive control is an implantation of adaptation method for updating of the expert system for changing environment.
- LCE technique may be considered as a type of cognitive control, where
  (1) the problem of updating of rule system is solved by a rudimentary form of GA and reinforcement learning, and
  (2) the merging of two rule sets that were produced by two agents is simply done by their union and then by removing rule that are inconsistent with other dominant rules.

A relationship between symbolic and subsymbolic methods of classification and control


cognitive control
symbolic methods
adaptation of rules
LCE
classification and control problem
subsymbolic methods
(1) clumsy interpretation of resulting neural networks
(2) still uknown methods for merging of information contained in neural networks

Formal description of LCE

1. Environment (states and actions)

The environment is specified by two sets and one mapping

(a) Set of environment states

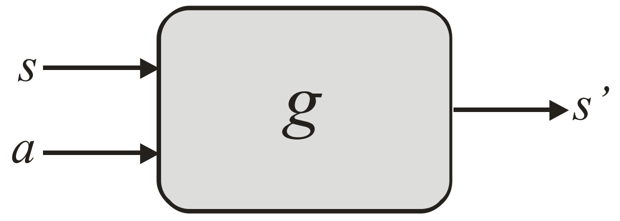
$$S = \{s_1, s_2, \ldots, s_a\}$$

(b) Set of actions onto system states

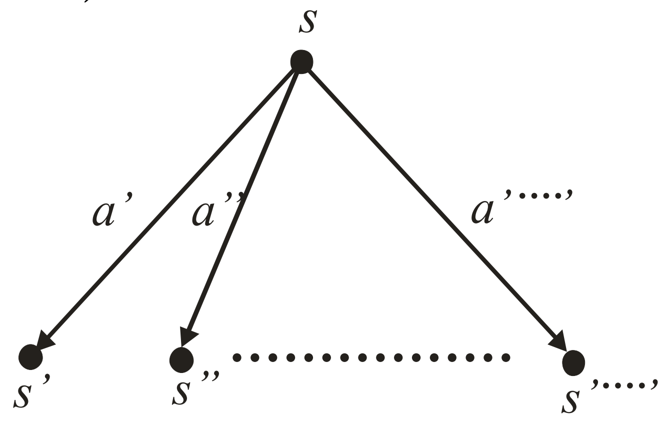
$$A = \{a_1, a_2, \ldots, a_b\}$$

(c) Relationship between states and actions are specified by a function

$$s' = g(a, s)$$



A



B

2. Set of classifiers (an expert system)

- ***Set of classifiers***

$$R = \{r_1, r_2, \ldots, r_p\}$$

is composed of rules $r \in R$ that are specified as ordered pairs of ***condition*** $\tilde{s}$ and ***action*** a

$$r = (\tilde{s}, a)$$

where the condition $\tilde{s} \in \{0,1,\#\}^u$ is a string of length u composed of three symbols 0, 1, and # (called "wild card"), and the action $a \in \{0,1\}^v$ is a string of length v composed of symbols 0 and 1.

- Standard specification of conditions is that these entities are string of symbols 0 and 1 with the fixed length *u,* i. e. $s \in \{0,1\}^u$, already used condition $\tilde{s}$ from the rule $r = (\tilde{s}, a)$ contains third symbol #, which is interpreted as an auxiliary symbol (wild card) that can be substitute either by 0 or 1.

- We say that a condition $\tilde{s} = (\tilde{s}_1, \tilde{s}_2, \ldots, \tilde{s}_u) \in \{0,1,\#\}^u$ is the ***scheme*** of a condition $s = (s_1, s_2, \ldots, s_u) \in \{0,1\}^u$, $s \leq \tilde{s}$, iff one of the following two conditions is satisfied (but not both)

$$(1) \quad s_i = \tilde{s}_i, \; or_{exclusive} \; (2) \quad \tilde{s}_i = \#$$

or formally

$$(s \leq \tilde{s}) =_{def} (\forall i)(\alpha_i = \tilde{s}_i \quad or_{exclusive} \quad \tilde{s}_i = \#)$$

- A scheme $\tilde{s} = (\tilde{s}_1, \tilde{s}_2, \ldots, \tilde{s}_u) \in \{0,1,\#\}^u$, where $s \leq \tilde{s}$, may be interpreted as a set composed of conditions that are created from α̃ in such a way that it "wilds" # are substituted by 0 or 1. For example, let $\tilde{s} = (01\#1\#)$ be a scheme composed of two symbols #, then this scheme specifies a set composed of 2^2=4 conditions

$$\tilde{s} = (01\#1\#) \Rightarrow set(\tilde{s}) = \{(01010),(01011),(01110),(01111)\}$$

- We may say that a scheme $\tilde{s}$ (or its set $set(\tilde{s})$) represents an ***abstraction*** of the class of standard conditions $s \in set(\tilde{s})$.

- Each rule $r \in R$ is evaluated by a positive number, $F(r)$, called the ***fitness***

$$F : R \to (0,\infty)$$

  As will be demonstrated latterly, the process of evaluation of rules by fitness belongs to the main part of the suggested method LCE.

3. An application of rules to environment

Let us have a rule $r=\left(\tilde{s},a\right)\in R$ and let the environment be in a state $s\in S$. Our goal is to demonstrate a method how to apply a rule to an environment; this method is composed of the following elementary steps:

(1) Let be the environment in state *s*, then we determine a subset of *R* as follows

$$R\left(s\right)=\left\{r=\left(\tilde{s},a\right)\in R; s\in set\left(\tilde{s}\right)\right\}=match\left(s,R\right)$$

This subset of rules is composed of those ones *r* with state parts $\tilde{s}$ specified as an abstraction of the given state s, i. e. $s\in set(\tilde{s})$.

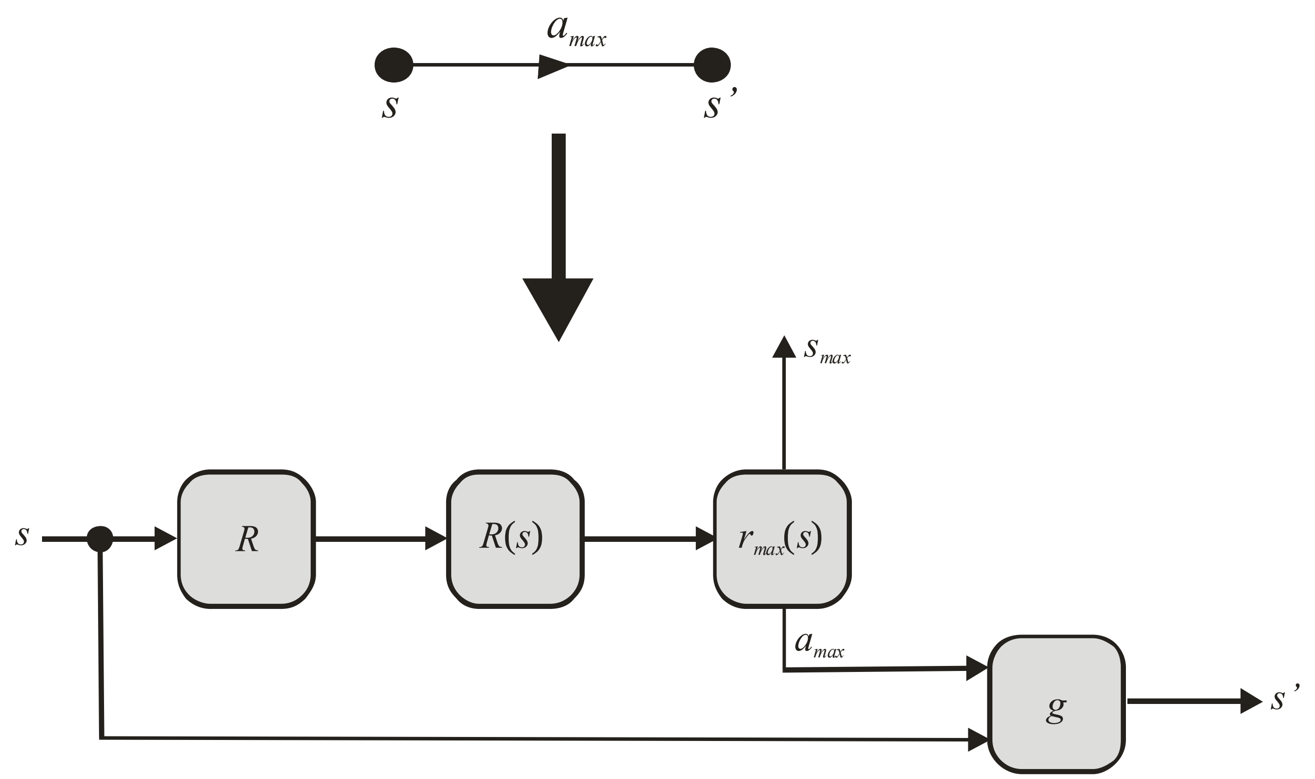
(2) From the subset *R*(*s*) we select a rule $r_{max}\left(s\right)\in R\left(s\right)$ such that

$$r_{max}\left(s\right)=arg\ \max_{r\in R\left(s\right)} F\left(r\right)$$

Where this optimal rule (with respect to a state *s* and for an actual evaluation *F*) has a form

$$r_{max}\left(s\right)=\left(\tilde{s}_{max},a_{max}\right)$$

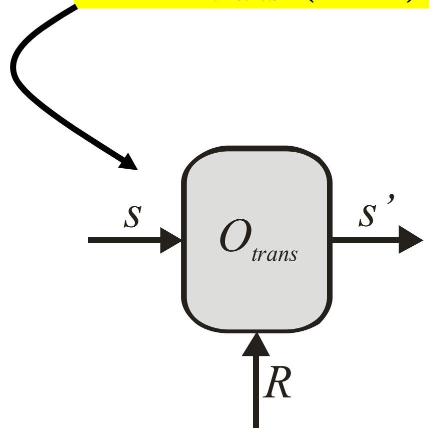
(3) The action a_{max} is applied to the environment in state s, we arrive at the environment in a new state $s' = F(a,s)$.



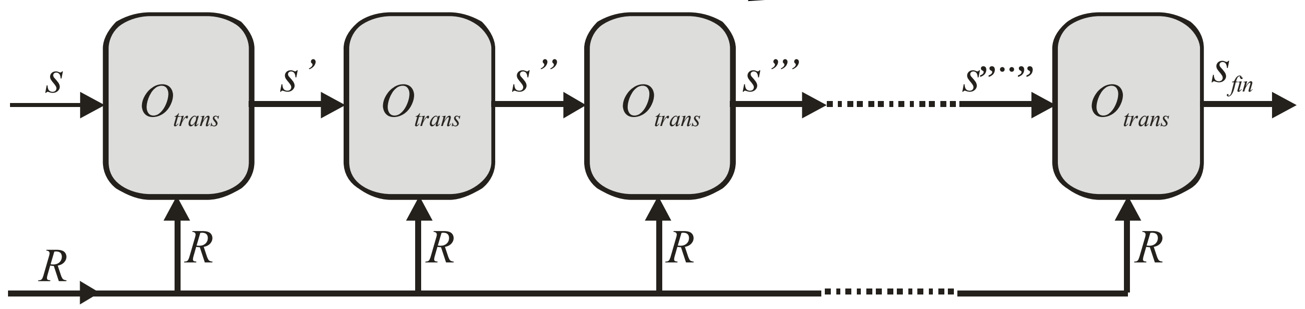
The whole process of transformation of environment from an "input" state *s* to "output" state *s'* (with respect to the rule system *R*) is formally expressed by an operator O_{trans}

$$s' = O_{trans}(s, R)$$

or diagrammatically



A sequence of transformations that transform an initial environment state *s* onto a final state s_{fin}.



4. Simple reinforcement learning

The above process of application of an action a_{max} to system states creates a sequence of states

$$s = s^{(ini)} \xrightarrow{a_{max}} s' \xrightarrow{a'_{max}} s'' \xrightarrow{a''_{max}} \ldots . s^{(fin)}$$

In such a way we may transform an initial state $s^{(ini)}$ onto a final state $s^{(fin)}$

$$s^{(ini)} \rightarrow \ldots . \rightarrow s^{(fin)}$$

Goal: To create such rule system R_{opt}, which transform initial state $s^{(ini)}$ onto a required final state $s^{(fin)}_{req}$

How to achieve this goal?
We use the method of ***reinforcement learning***!

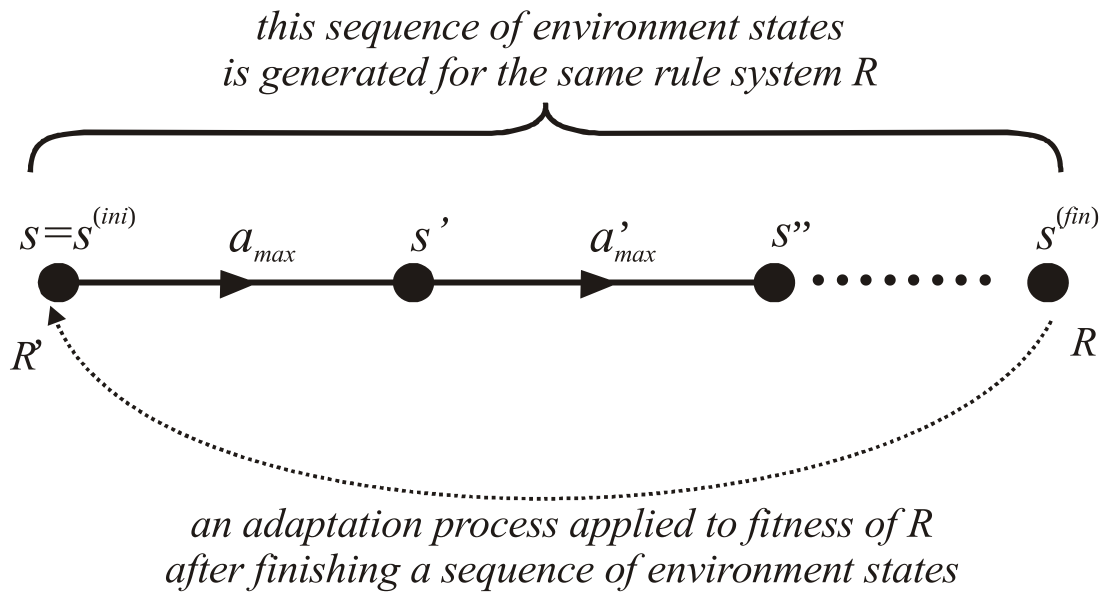
Learning Classifier System

In order to be adapted (learned) a software system; it must have variable (flexible) parameters that specify its resulting properties. E. g., a neural network system is adapted (learned) with respect to its weight coefficients that specify neural-network output.

- In LCS a role of the mentioned variable parameters ***play fitness*** parameters that specify an applicability of LCS to a given environment state.

- Reinforcement learning offers effective technique how to systematically change fitness of rules in such a way that an application of LCS will produce the final required environment state.

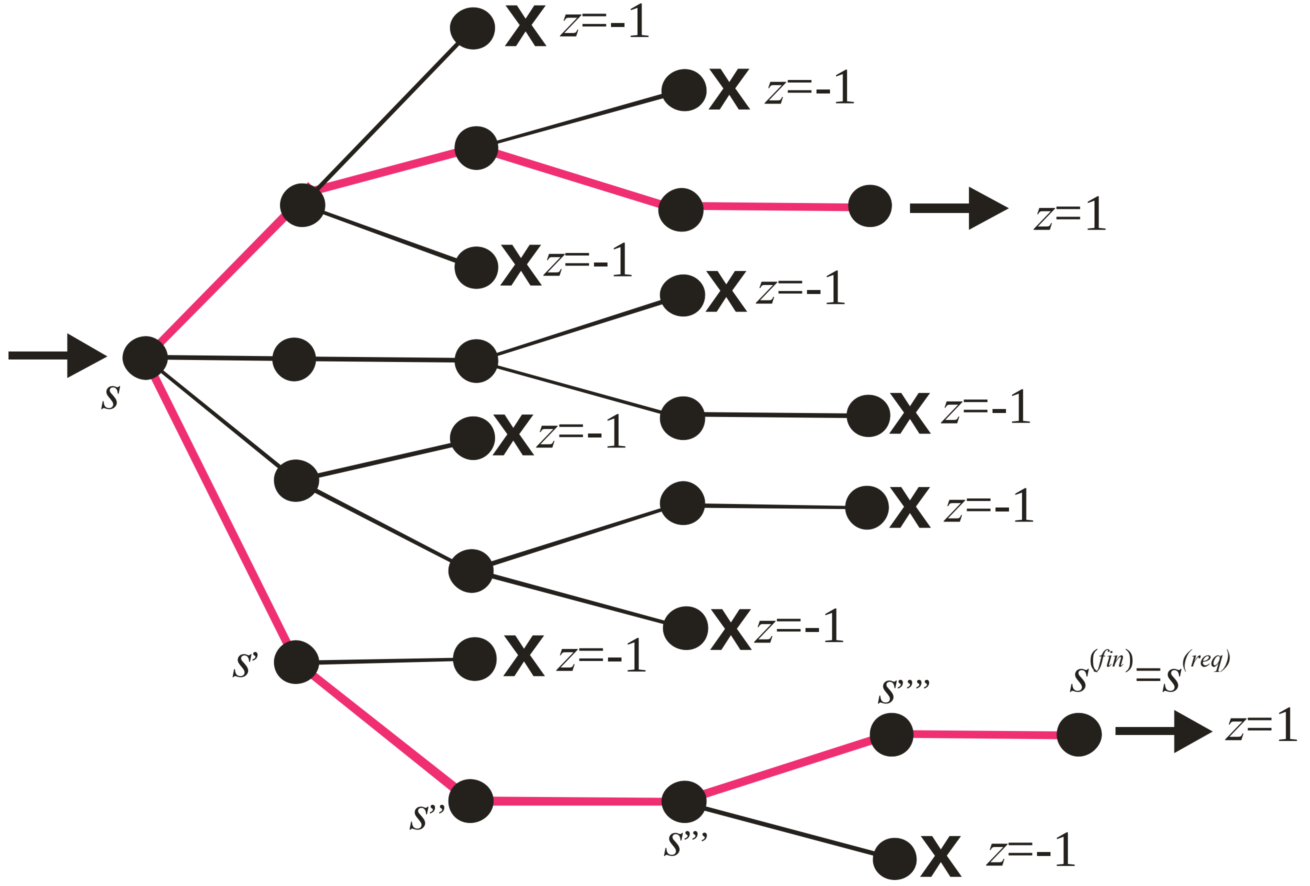
Diagrammatic interpretation of adaption process applied to rule system *R* when a sequence of environment states was created.


this sequence of environment states
is generated for the same rule system R
$s=s^{(ini)}$
a_{max}
s'
a'_{max}
s''
$s^{(fin)}$
R'
R
an adaptation process applied to fitness of R
after finishing a sequence of environment states

$\left(s,s',s'',\ldots,s^{(fin)};z\right)$…a sequence of environment states initialized by *s* and finished by a state $s^{(fin)}$; real number "reward" *z* specified whether the expected-required final state was achieved.

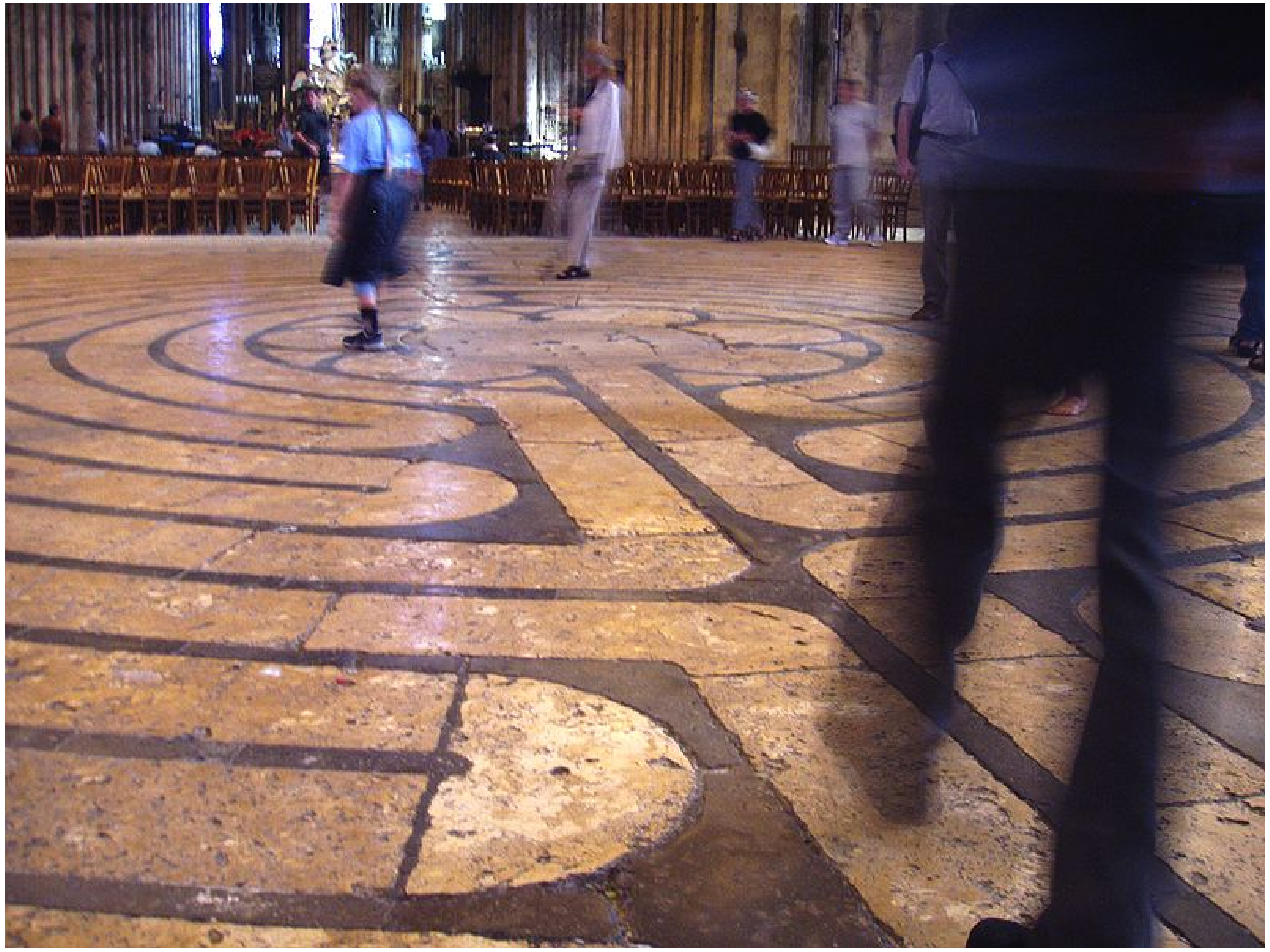
reward, $-1 \le z \le 1$

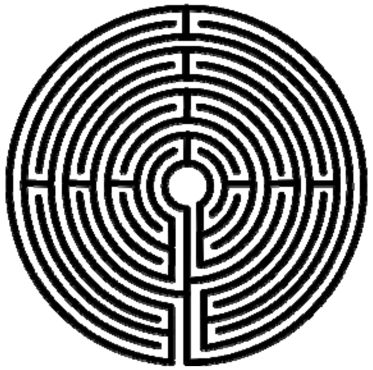
Illustrative example of moving through the labyrinth



Roman mosaic picturing Theseus and the Minotaur.

Famous labyrinth on floor of Cathedral of Our Lady of Chartres





It is not a genuine labyrinth, the path from start point to the goal point is unambiguously determined (a linear labyrinth)

Alternative description of creation of a sequence of states of environment

$$s = s^{(ini)} \xrightarrow{a_{max}} s' \xrightarrow{a'_{max}} s'' \xrightarrow{a''_{max}} \dots \xrightarrow{a''^{\dots\prime}{}_{max}} s^{(fin)}$$

$$\Big\downarrow$$

$$s = s^{(ini)} \xrightarrow{r^{(1)}{}_{max}(s)} s' \xrightarrow{r^{(2)}{}_{max}(s')} s'' \xrightarrow{r^{(3)}{}_{max}(s'')} \dots \xrightarrow{r^{(q)}{}_{max}\left(s''^{\dots\prime}\right)} s^{(fin)}$$

It means that a sequence of environment states $\left(s, s', s'', \dots, s^{(fin)}\right)$ is substituted by a sequence rules $\left(r_1, r_2, r_3, \dots, r_q\right)$ from the fixed rule system R

$$\left(s, s', s'', \dots, s^{(fin)}\right)$$

$$\Big\downarrow$$

$$\left(r_1, r_2, r_3, \dots, r_q\right)$$

We shall postulate that the rule sequence $\left(r_1, r_2, r_3, \ldots, r_q\right)$ is evaluated by a reward z, we get

$$\left(r_1, r_2, r_3, \ldots, r_q; z\right)$$

reward, $-1 \le z \le 1$

This is a basic concept, which will be studied in our forthcoming considerations on LCE

Update of fitness for a sequence $\left(r_1, r_2, r_3, \ldots, r_q; z\right)$

$$F_i' = F_i + \lambda\gamma^{p-i+1}z$$

where $\lambda > 0$ is the learning rate and $0 \le \gamma \le 1$ is the discount factor. This modification of fitness satisfies the following obvious property

$$z > 0 \Rightarrow F_i' > F_i \quad \text{(reward)}$$

$$z < 0 \Rightarrow F_i' < F_i \quad \text{(punishment)}$$

The discount factor γ controls a deep of updating of fitness:

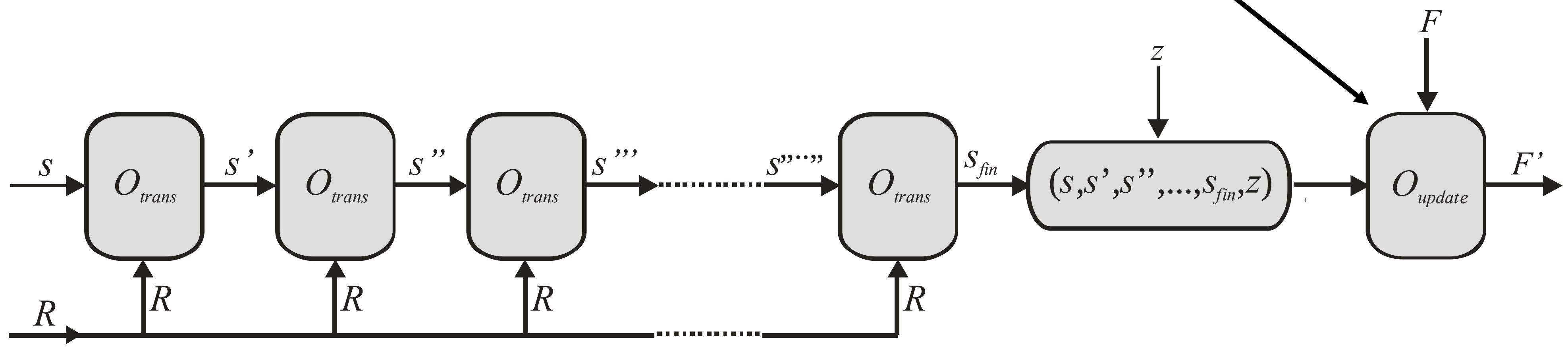
(γ=1) $\Rightarrow$ fitness of all rules from the sequence is updated in the same level,

(γ<1) $\Rightarrow$ fitness of all rules from the sequence is updated in a decreasing level going from the last rule to the first rule, i. e. the fitness medication are smaller as we go to the first initial rule.

(γ=0) $\Rightarrow$ fitness of all rules from the sequence remains unchanged.

Formally, a process of updating of fitness with respect to a sequence of environment states (which is evaluated by reward *z*) is expressed by an operator O_{update}

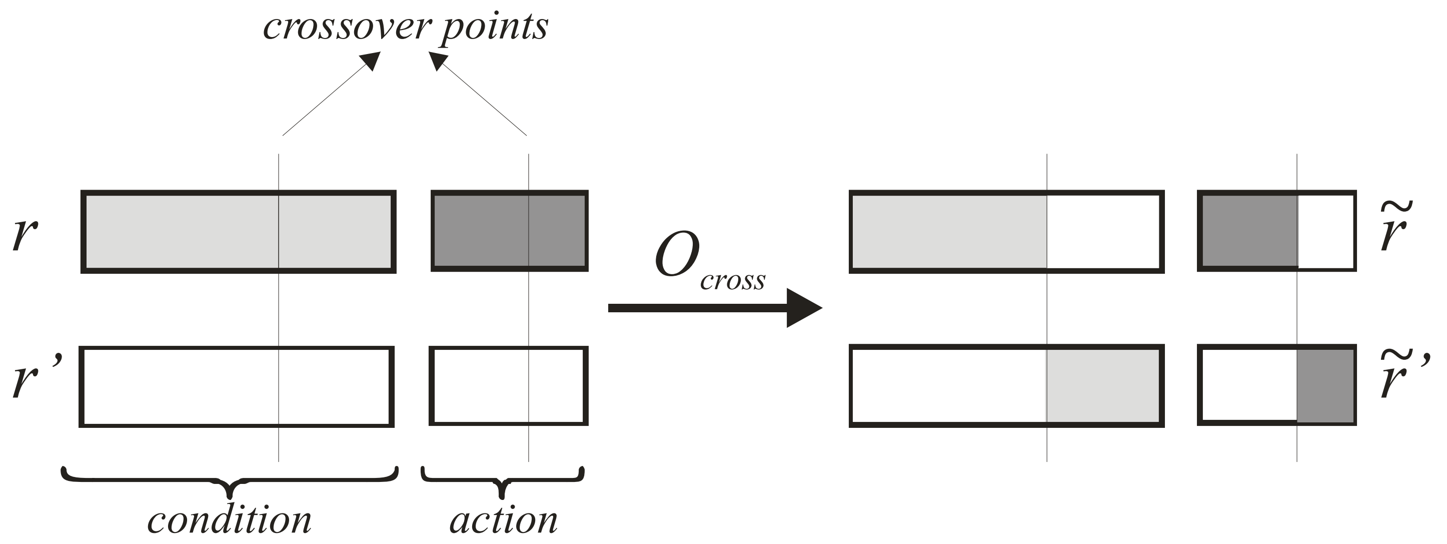
$$F' = O_{update}\left(F,\left(r_1,r_2,r_3,\ldots,r_q;z\right)\right)$$



4. GA innovation of rule set *R*

(1) ***Crossover operation***

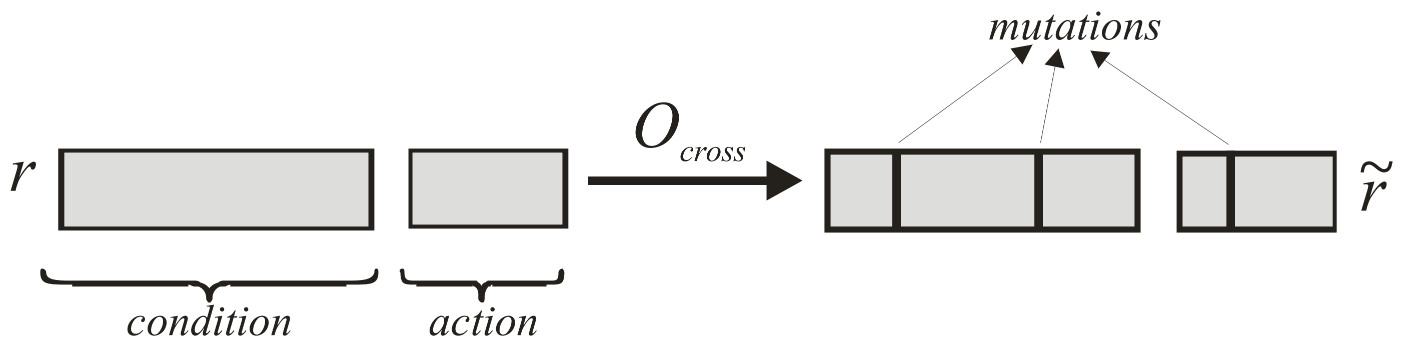
$$\left(\tilde{r},\tilde{r}'\right)=O_{cross}\left(r,r'\right)$$



$$\begin{Bmatrix}\left(s_1,\ldots,s_i,s_{i+1}\ldots,s_m\right)\left(a_1,\ldots,a_j,a_{j+1},\ldots a_n\right)\\\left(s'_1,\ldots,s'_i,s'_{i+1}\ldots,s'_m\right)\left(a'_1,\ldots,a'_j,a'_{j+1},\ldots a'_n\right)\end{Bmatrix}\xrightarrow{O_{cross}}\begin{Bmatrix}\left(s_1,\ldots,s_i,s'_{i+1}\ldots,s'_m\right)\left(a_1,\ldots,a_j,a'_{j+1},\ldots a'_n\right)\\\left(s'_1,\ldots,s'_i,s_{i+1}\ldots,s_m\right)\left(a'_1,\ldots,a'_j,a_{j+1},\ldots a_n\right)\end{Bmatrix}$$

(1) ***Mutation operation***

$$(\tilde{r}) = O_{cross}(r)$$



$$\underbrace{(s_1, s_2, \ldots, s_m)}_{s}\underbrace{(a_1, a_2, \ldots, s_n)}_{a} \xrightarrow{O_{mut}} \underbrace{(\tilde{s}_1, \tilde{s}_2, \ldots, \tilde{s}_m)}_{\tilde{s}}\underbrace{(\tilde{a}_1, \tilde{a}_2, \ldots, \tilde{a}_n)}_{\tilde{a}}$$

$$\tilde{s}_i = \begin{cases} comp(s_i) & (prob < P_{mut}) \\ s_i & (otherwise) \end{cases} \qquad \tilde{a}_i = \begin{cases} comp(a_i) & (prob < P_{mut}) \\ a_i & (otherwise) \end{cases}$$

probability of mutation
small positive number

Elementary GA step applied to an actual rule system such that an *innovated* rule system is produced

$$R' = O_{inno}\left(R\right)$$

This elementary step of LCS is composed of the following step:

(1) Select quasirandomly (with respect to fitness of rules) by roulette wheel two rules,

$$r = O_{select}\left(R\right),\quad r' = O_{select}\left(R\right)$$ ’

(2) Apply to these selected rules operations of crossover and mutation

$$\left(\hat{r},\hat{r}'\right) = O_{cross}\left(r,r'\right),\quad \tilde{r} = O_{mut}\left(\hat{r}\right),\quad \tilde{r}' = O_{mut}\left(\hat{r}'\right)$$

(3) Select by ***inverse*** roulette wheel (with respect to inverse fitness) two rules,

$$\overline{r} = O_{select}^{inv}\left(R\right),\quad \overline{r}' = O_{select}^{inv}\left(R\right)$$

(4) The original rule set *R* is updated to a new set *R'*

$$R' = \left(R - \{\bar{r}, \bar{r}'\}\right) \cup \{\tilde{r}, \tilde{r}'\}$$

(5) Finally, the new rules $\tilde{r}, \tilde{r}'$ are evaluated by fitness, e. g. by arithmetic mean

$$F(\tilde{r}) = F(\tilde{r}') = \frac{1}{2}\left[F(r) + F(r')\right]$$

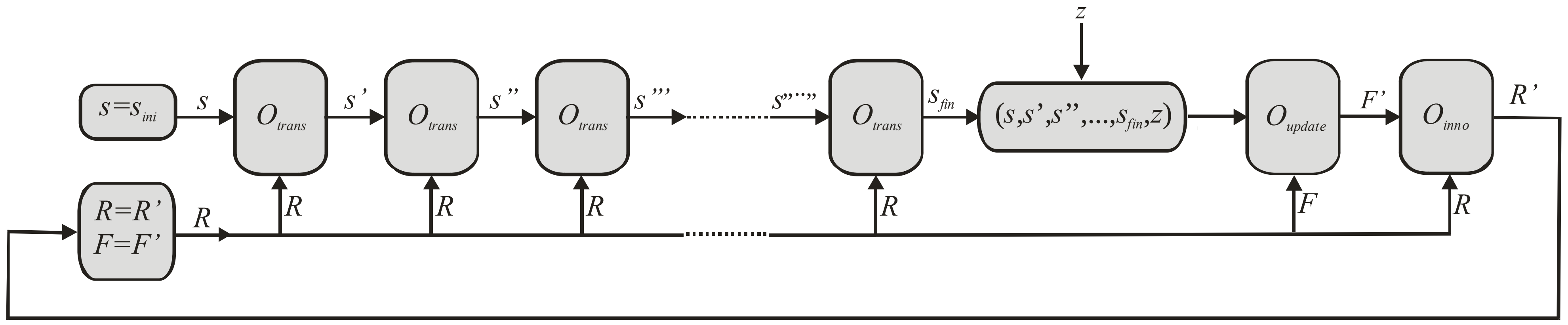
Note: This step may be realized by many alternative ways

5. Summary of LCS algorithm

Main components of LCE are:

- set of ***rules***,
- mechanism of ***interaction*** of rules with environment
- ***updating*** of rules fitness by RL,
- GA mechanism of ***innovation*** of rule sets.

An evolution of LCS for a given environment with specified pair of initial and final states



Hill Climbing with Learning (HCwL)

We have an objective function $f:\{0,1\}^r \to R$, which maps binary vectors of the length r onto real numbers. Our goal is to find a solution of the global minimization problem

$$x_{opt} = arg \min_{x\in\{0,1\}^r} f(x)$$

In the forthcoming we demonstrate a simple method, called the *hill climbing*, to solve this discrete optimization problem.

Basic ideas of the hill climbing consists in a simple approach how to systematically generate new approximate solutions of optimization problem such that for a given temporally solution $x\in\{0,1\}^r$ a new solution is constricted with a neighborhood $U(x)$ of the solution x

$$x' = arg \min_{y\in U(x)} f(y)$$

The neighborhood is o the form

$$U(x) = \{y; y = O_{mut}(x)\}$$

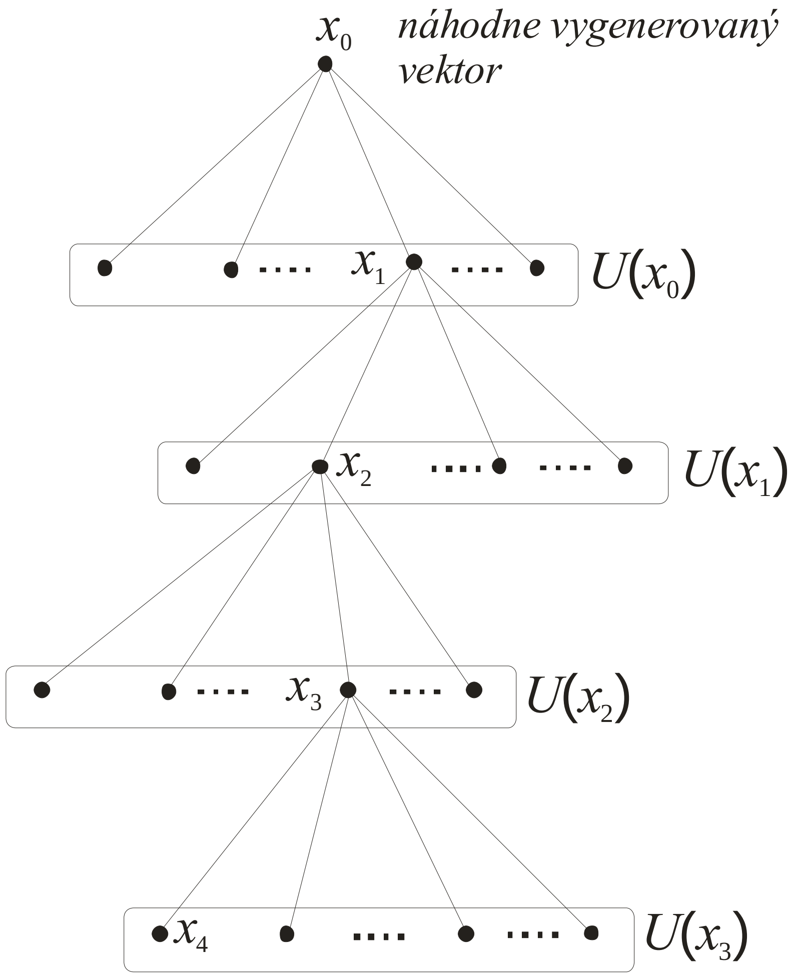
where $O_{mut}(x)$ is an application of mutation operator (well known in GA) to the binary vector x.

$$x' = (x_1', x_2', \ldots, x_r') = O_{mut}(x_1, x_2, \ldots, x_r)$$

$$x_i' = \begin{cases} 1 - x_i & (random(0,1) < P_{mut}) \\ x_i & (otherwise) \end{cases}$$

$$\lim_{P_{mut} \to 0} O_{mut}(x) = x, \quad \lim_{P_{mut} \to 1} O_{mut}(x) = \overline{x} = (\ldots, 1 - x_i, \ldots)$$

Found locally optimal solution *x'* is used in the forthcoming step aqn a centre of new neighborhood *U*(*x'*)



Though the hill climbing does not contain any evolutionary strategy, it is very effective and robust algorithm which is capable ***to find global minimum*** of many optimization problems.

Standard hill climbing algorithm may be simply modified in the so-called ***hill climbing with learning algorithm***. This modification consists in a modification of construction of the neighborhood $U(x)$. Its original specification uses stochastic mutation operator O_{mut} , but in the generalized version the neighborhood is defined as follows:

(1) First we introduce a probability vector $\boldsymbol{w} = \left(w_1, w_2, \ldots, w_r\right) = \left[0,1\right]^r$, its single elements $0 \le w_i \le 1$ specify a probability of appearing element '1' in the i-th position of solution $x = \left(\ldots, x_i, \ldots\right)$

$$x_i = \begin{cases} 1 & \left(random\left[0,1\right] < w_i\right) \\ 0 & \left(\text{otherwise}\right) \end{cases}$$

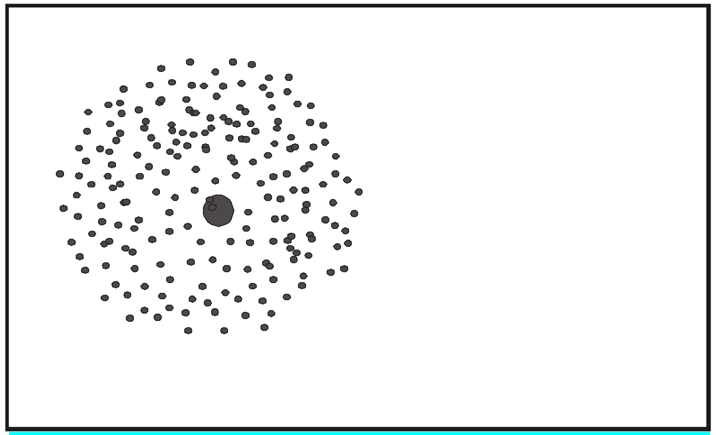
This stochastic construction of the binary string x we denote as

$$x = R\left(\boldsymbol{w}\right)$$

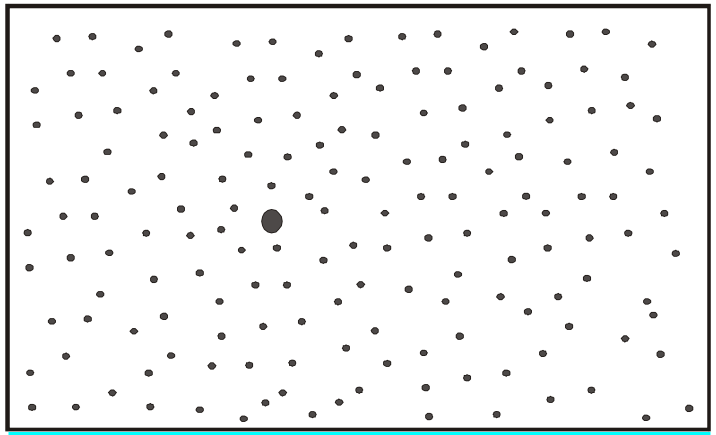
A neighborhood is constructed as follows

$$U(\boldsymbol{w}) = \{x; x = R(\boldsymbol{w})\}$$

If all components of probability vector are closely related either to zero or one, then a "diameter" of neighborhood is "small", in the opposite case, if components are closely related to ½, then the neighborhood is distributed through all binary vectors



All components of ***w*** are closely related either to 0 or 1



All components of ***w*** are closely related to ½

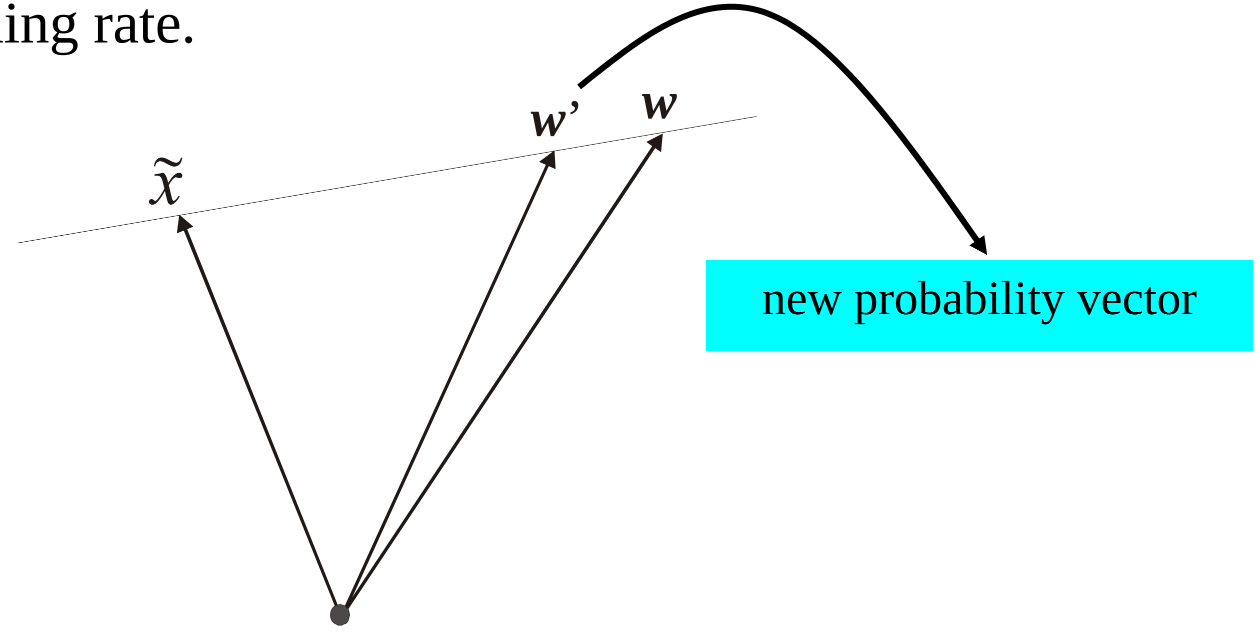
Second important item of the proposed method is the so-called learning. Let $\tilde{x}$ be a optimal solution of a local minimization within the neighborhood $U(\mathbf{w})$

$$\tilde{x} = arg \min_{y \in U(\mathbf{w})} f(y)$$

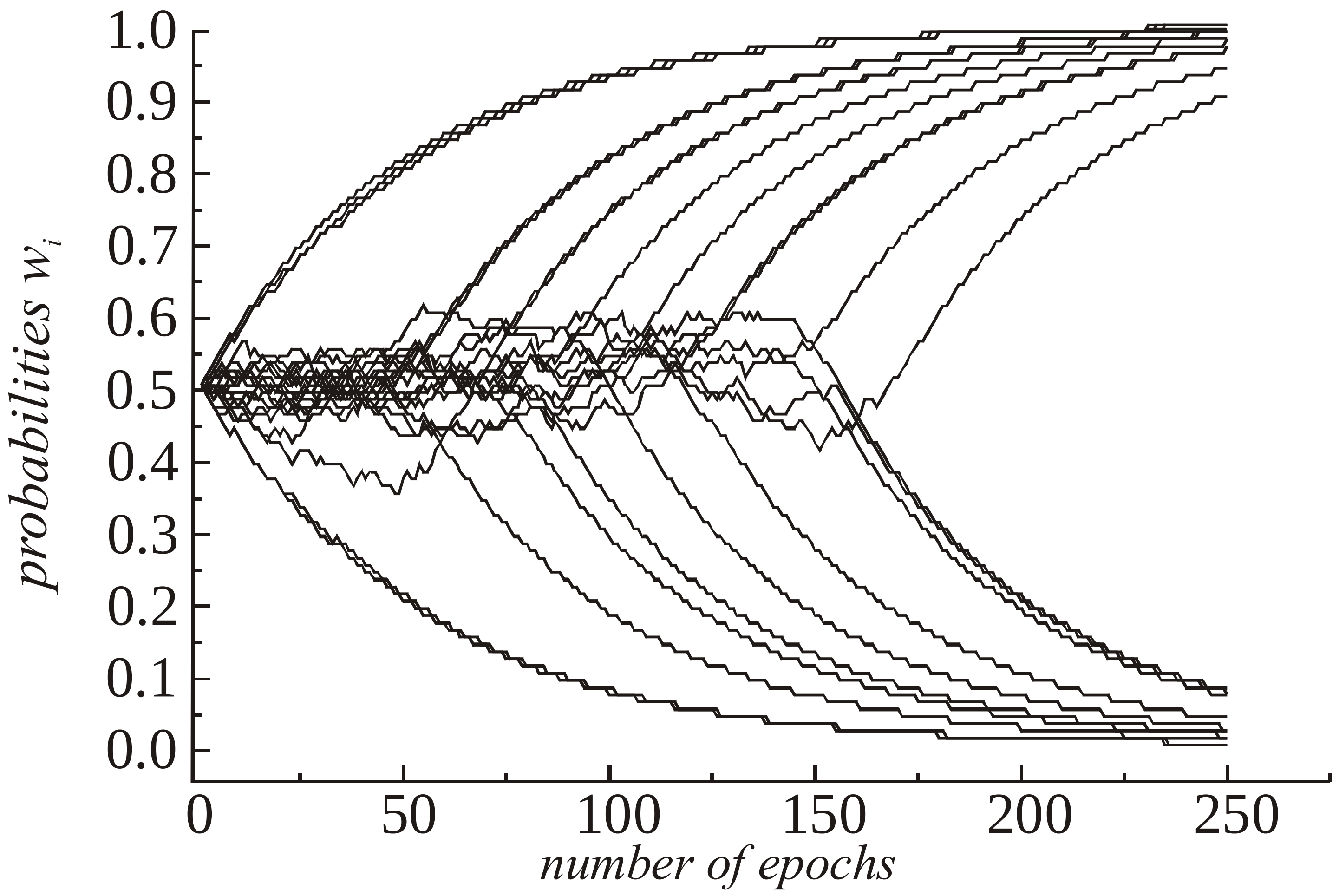
The learning process consists in the following updating of the probability vector }(an analogue of Hebbian learning)

$$w \leftarrow w + \lambda(\tilde{x} - w) = (1-\lambda)w + \lambda \tilde{x}$$

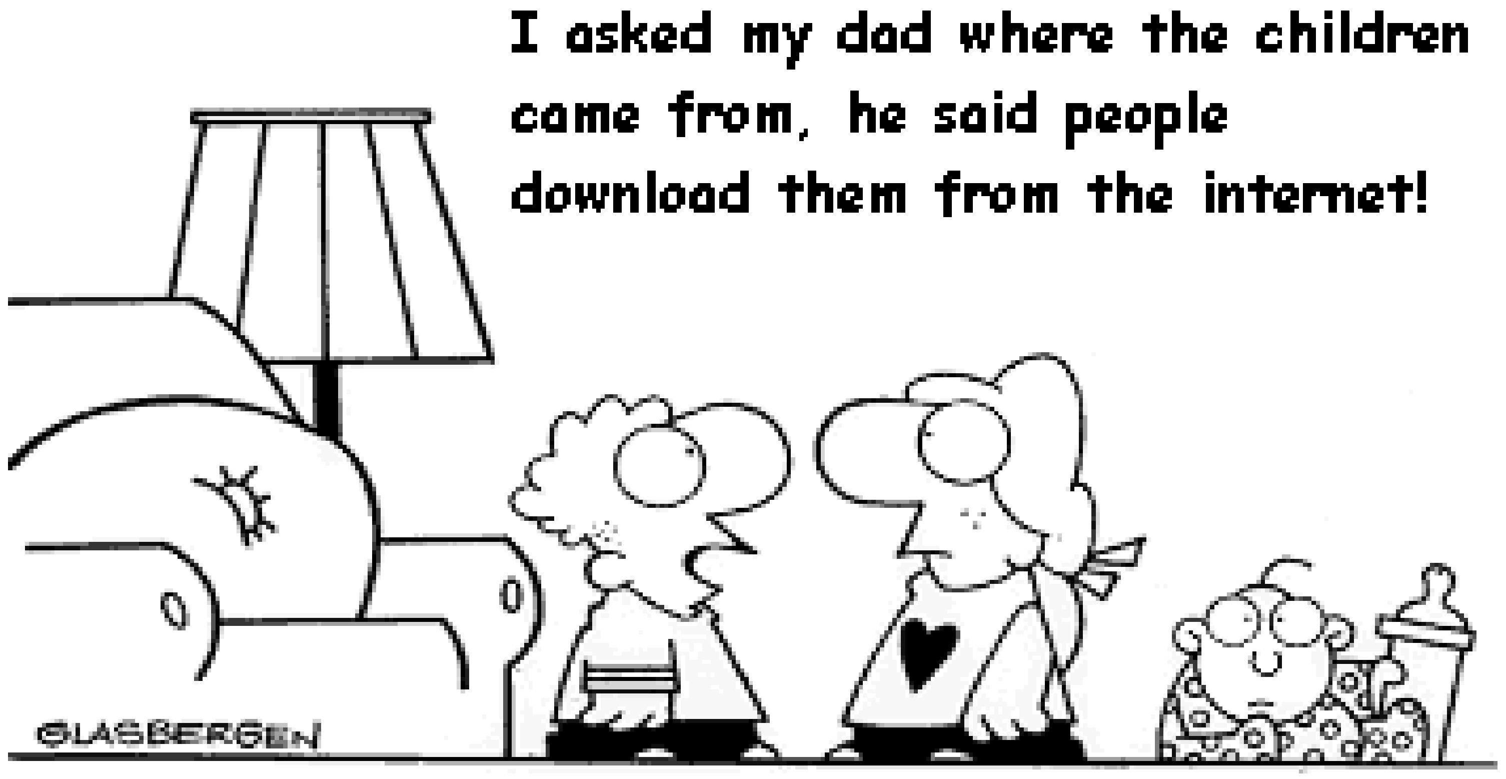
by a convex combination of weight vector ***w*** and local solution $\tilde{x}$, and λ (small positive number) is a learning rate.



Plot of components w_i of probability vector w

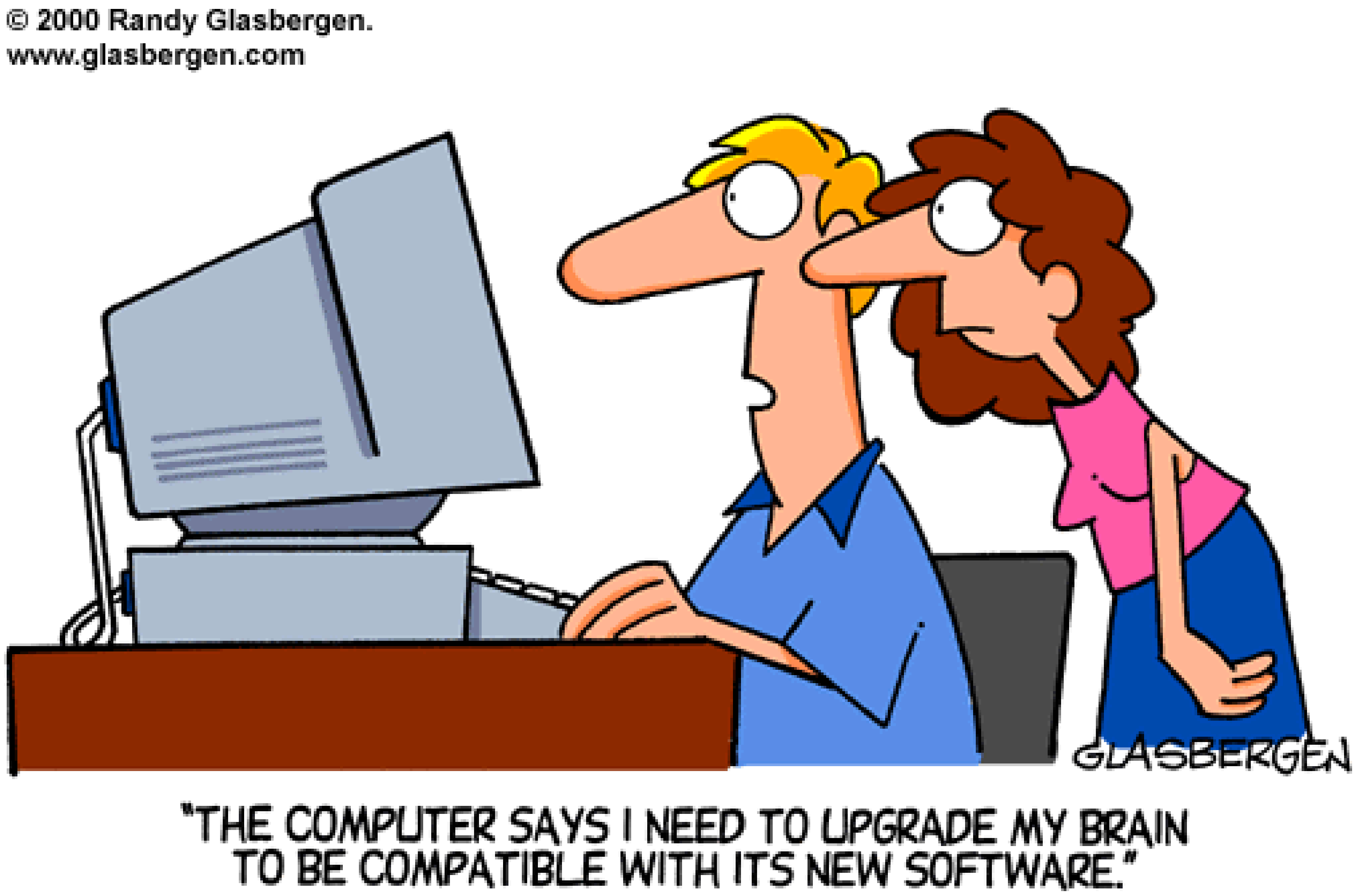


Some computer jokes at the end


I asked my dad where the children came from, he said people download them from the internet!
GLASBERGEN

Copyright 1996 Randy Glasbergen. www.glasbergen.com

"Sorry about the odor. I have all my passwords tattooed between my toes."


© 2000 Randy Glasbergen.
www.glasbergen.com
GLASBERGEN
"THE COMPUTER SAYS I NEED TO UPGRADE MY BRAIN TO BE COMPATIBLE WITH ITS NEW SOFTWARE."

The End